CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER'S THESIS

Title:	The Impossible Differential Cryptanalysis
Student:	Bc. Peter Poljak
Supervisor:	Ing. Josef Kokeš
Study Programme:	Informatics
Study Branch:	Computer Security
Department:	Department of Computer Systems
Validity:	Until the end of winter semester 2017/18

Instructions

Research the available papers dealing with the impossible differential cryptanalysis. Describe the current state-of-the-art of the field and explain the workings of this technique. Compare it to the other cryptanalytic techniques. Select a suitable cipher or its model, implement it, and demonstrate the execution of the attack. Evaluate the results.

References

Biham, E., Keller, N.: Cryptanalysis of reduced variants of Rijndael, 3rd AES Conference, 2000.
Phan, R. C.-W.: Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES). Elsevier: Information Processing Letters, Volume 91, Issue 1, 2004.
Kim, J., Hong, S., Sung, J., Lee, S., Lim, J., Sung, S.: Impossible Differential Cryptanalysis for Block Cipher Structures. Progress in Cryptology - INDOCRYPT 2003. Springer: Lecture Notes in Computer Science, Volume 2904, 2003.

L.S.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague June 30, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Master's thesis

The Impossible Differential Cryptanalysis

Bc. Peter Poljak

Supervisor: Ing. Josef Kokeš

10th January 2017

Acknowledgements

First I would like to thank my supervisor Ing. Josef Kokeš for bringing insightful ideas, quick responses to my email inquiries and willingness to frequently meet and talk. These things helped a lot. Furthermore, I would like to thank my family for support they provided during my work on the diploma thesis. Last but not least I would like to thank Anna Madliak for support, help needed to meet the schedule, reading through diploma thesis and giving ideas for improvement and being my driving force when I needed that. This diploma thesis would not be possible without these people, so once more – thank you.

Declaration

In Prague on 10th January 2017 . . . .

Citation of this thesis

Poljak, Peter. *The Impossible Differential Cryptanalysis.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

V našej diplomovej práci sme sa venovali kryptoanalýze pomocou nemožných diferenciálov ako mierke na testovanie slabín šifier. Kvôli relatívnej novosti, oproti lineárnej a diferenciálnej analýze, nie je táto analýza veľmi známa a existuje len málo jednoduchých "ako na to" návodov. Zaumienili sme si poskytnúť presne tento jednoduchý "ako na to" návod. Na testovanie tejto techniky sme si zvolili šifru Baby Rijndael. Podarilo sa nám zaútočiť na 4 rundy Baby Rijndaelu a poskytnúť detailný návod. Prišli sme na to, že potrebujeme len 13436 časových jednotiek oproti 32768 jednotiek potrebných na útok hrubou silou. Potvrdili sme užitočnosť kryptoanalýzy pomocou nemožných diferenciálov a s vyššie spomínaným detailným návodom sme k nej zjednodušili prístup.

Klíčová slova Baby Rijndael, Kryptoanalýza, Kryptoanalýza pomocou nemožných diferenciálov

Abstract

In our diploma thesis we focused on the impossible differential cryptanalysis as a benchmark to test weaknesses of ciphers. Because of its relative novelty, compared to linear and differential analysis, this analysis is not so well-known and there are few-to-none simple tutorials on how to do it. We set to provide exactly this simple how to tutorial. We chose Baby Rijndael cipher to test this technique. We performed a successful attack on 4 rounds of Baby Rijndael and provided a step-by-step tutorial. We found out that we need only 13436 units of time instead of 32768 needed for brute-force attack. We confirmed the usefulness of this technique and with the provided how to simplified an access to its workings.

Keywords Baby Rijndael, Cryptanalysis, Impossible differential cryptanalysis

Contents

List of Figures

List of Tables

Introduction

People have been developing tools to hide information from the dawn of time. This allowed them to gain an advantage over another people (societies). The most noticeable example would be wars. Naturally these tools became better and better, century after century. These improvements resulted in the existence of Enigma during World War II. For a long part of World War II this cipher remained unsolved and Germans could transfer messages between each other without problems. Solving of Enigma cipher became the most important problem for Allies. When they did finally solve Enigma – it changed the tide of war drastically.

With gradual increase of popularity of computers, ciphers became more mathematical. We (as a society) started using them as a tool to secure almost every piece of information moving through a network of computers. We have recognized the importance of cipher design and have incorporated analysis of proposed ciphers to it.

It is crucial to know and perform various types of analyses of a cipher before making it a standard for securing information. Because of that, the analysts have developed a lot of methods. One of these methods is **the impossible differential cryptanalysis**. Because of its relative novelty, the analysis is not so well known. This makes harder to find study materials for future cryptanalysts – let alone find “how to“ tutorial like [3] for differential and linear cryptanalysis. As a result, we focus on providing this “how to“ in our diploma thesis.

Our aim is to acquaint ourselves with the workings of impossible differential cryptanalysis and its current state-of-the-art. This knowledge can be later used to formulate an easy to use tutorial. And finally implement and demonstrate the application of the tutorial to a real cipher.

Research gives us theoretical basis for further understanding of this technique. Part of the research provides a comparison of impossible differential cryptanalysis to other techniques which helps us to better understand the differences and the similarities between them. The emphasis on a tutorial is mostly because it is often the quickest way of explaining something new to another person. We have selected Baby Rijndael as a suitable cipher for demonstration. It is simple to understand so we can focus mainly on the analysis.

We divided our diploma thesis into two parts: theoretical and practical. The theoretical part consists of three chapters. The first one is **Baby Rijndael** which gives us details on the chosen cipher – Baby Rijndael. The second one is **Cryptanalysis** which describes linear, differential and algebraic cryptanalysis – so that we can compare impossible differential cryptanalysis to them. The third one is **Impossible differential cryptanalysis** which provides history and current development of this technique. After that we described the principle and offered a simple example. Following these chapters we continued with the practical part of diploma thesis which consists of one chapter **Impossible differential cryptanalysis of Baby Rijndael**, providing the implementation, analysis of the cipher and the execution of the attack.

Chapter 1

Baby Rijndael

Baby Rijndael is a *symmetric block cipher* derived from Rijndael cipher – the algorithm selected as the Advanced Encryption Standard (AES[1]). The reason behind the choice of this scaled-down version of Rijndael is simple: the behaviour and the structure of cipher are the same but the analysis is much less time and space consuming due to the reduced size of a block and key. Instead of 128 bit long block and 128 bit long key (one version of Rijndael), Baby Rijndael is using only 16 bit long block and 16 bit long key. This gives us an opportunity to even perform a *brute-force attack* which provides a good benchmark for any type of analysis.

1.1 Cipher structure

Baby Rijndael consists of only four rounds (instead of Rijndael's 10), identical in structure [4]. Each round consists of these four successive operations: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*.

The state, but also the plaintext, ciphertext and key of *Baby Rijndael* is 16 bits long, arranged in 2×2 matrix with every element consisting of 4 bits. This gives us $2 \times 2 \times 4 = 16$ bits. In contrast, *Rijndael* has a 128 bit long state and a 4×4 matrix with every element consisting of 8 bits (1 byte). However, the structure is the same, meaning: positions in columns are filled from top to bottom and columns are filled left to right.

$$A = \begin{pmatrix} a_0 & a_2 \\ a_1 & a_3 \end{pmatrix},\ a_i \in GF(2^4) \tag{1.1}$$

Note: We usually write the state as a sequence of four hexadecimal digits.

Suppose we have a *block* = 0110 1011 0101 1101; we translate it to $(6, b, 5, d)_{16}$ and *state* then looks like this: $\begin{pmatrix} 6 & 5 \\ b & d \end{pmatrix}$.

[1]AES – more at `http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard`.

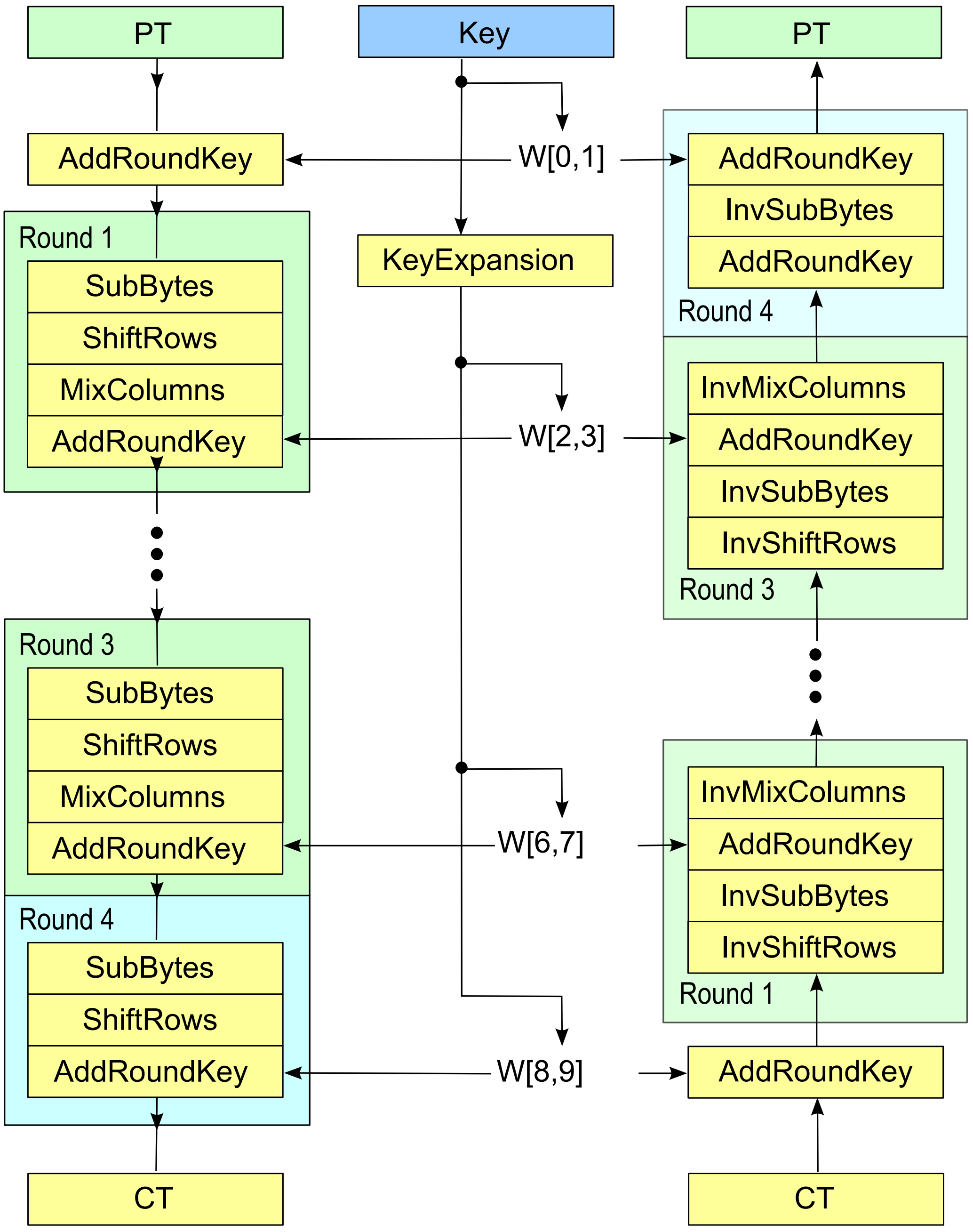

Figure 1.1: Structure of Baby Rijndael [1, page 14].

Figure 1.1 (cipher structure) can be written down as follows[2]:

$$E(a) = r_4 \circ r_3 \circ r_2 \circ r_1(a \oplus k_0) \tag{1.2}$$

where a stands for *state*, $k_0, .., k_4$ are the round keys and r_i is round

$$r_i(a) = (t \cdot \sigma(s(a))) \oplus k_i \tag{1.3}$$

where in r_4, multiplication by t is absent. The component operations are:

$$\begin{gathered} s\ (\mathit{SubBytes}) \\ \sigma\ (\mathit{ShiftRows}) \\ t\cdot\ (\mathit{MixColumns}) \\ \oplus k_i\ (\mathit{AddRoundKey}) \end{gathered}$$

1.2 SubBytes

Operation *SubBytes* can be imagined as a substitution table where every position (one hexadecimal digit) is translated to a new value.

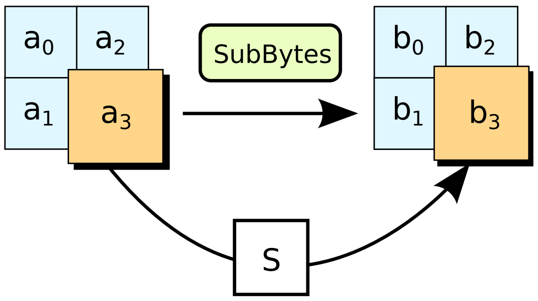

Figure 1.2: Baby Rijndael's SubBytes [1, page 15].

The substitution table (*SubBytes*):

Input	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Output	a	4	3	b	8	e	2	c	5	7	6	f	0	1	9	d

Table 1.1: Baby Rijndael's SubBytes.

[2]Symbol $\circ$ stands for composition of functions.

An important feature of *SubBytes* is its *non-linearity*. This operation is the only *non-linear* one in the cipher.

A proof that Baby Rijndael's *SubBytes* has the same properties as Rijndael's can be found in diploma thesis [1, page 16-18] and paper [5, page 16-17].

1.3 ShiftRows

Operation *ShiftRows* in Baby Rijndael is just swapping of two positions in the second row. It is based on Rijndael's *ShiftRows*, where every row (from top to bottom) is shifted $i-1$ positions to the left where i is row number starting with 1. In Baby Rijndael it means that we get the same operation, because swapping of two positions in the second row is just shifting one position to the left.

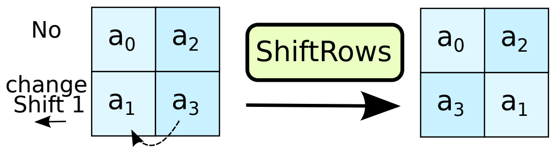

Figure 1.3: Baby Rijndael's ShiftRows [1, page 18].

1.4 MixColumns

Operation *Mixcolumns* performs a matrix multiplication modulo 2 of the (transformed – see below) state by t on the left.

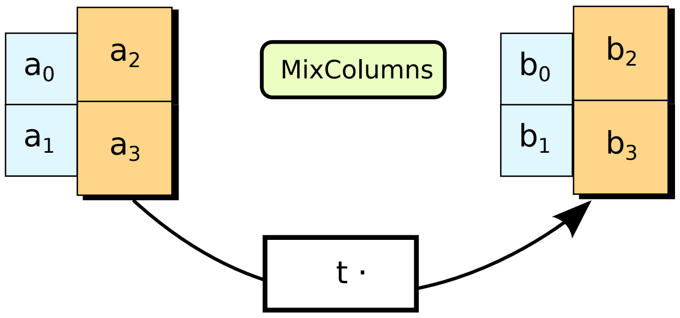

Figure 1.4: Baby Rijndael's MixColumns [1, page 20].

Where matrix t is:

$$t = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{1.4}$$

And the *state* is considered not to be a 2 × 2 matrix with 4 bit long positions, but rather a **1 × 2 matrix with 8 bit long positions**. When we recall an example of *state* from section 1.1, we can transform it to:

$$A = \begin{pmatrix} 6 & 5 \\ b & d \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{1.5}$$

A proof that Baby Rijndael's *MixColumns* has the same properties as Rijndael's can be found in diploma thesis [1, page 19-22].

1.5 AddRoundKey

Operation *AddRoundKey* performs an addition (modulo 2) of *key* and *state* in corresponding positions.

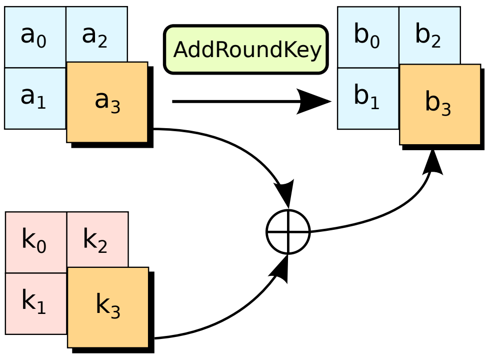

Figure 1.5: Baby Rijndael's AddRoundKey [1, page 22].

1.6 Key schedule

Key schedule's purpose is to generate several round keys from one main key. This is done by introducing an array of 8 bit long columns of the round keys – w.

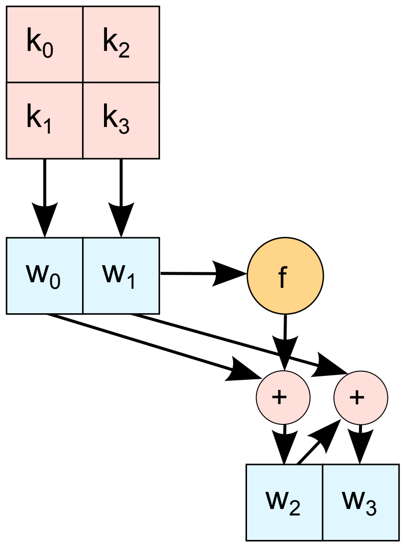

Figure 1.6: Baby Rijndael's Key schedule [1, page 23].

From figure 1.6 we can notice that w_0 and w_1 are columns (respectively) of *main key*.

$$w_0 = \begin{pmatrix} k_0 \\ k_1 \end{pmatrix} \quad w_1 = \begin{pmatrix} k_2 \\ k_3 \end{pmatrix} \tag{1.6}$$

Function f operates as follows:

$$\begin{aligned} w_{2i} &= w_{2i-2} \oplus s(\text{rotation}(w_{2i-1})) \oplus y_i \\ w_{2i+1} &= w_{2i-1} \oplus w_{2i} \end{aligned} \tag{1.7}$$

- **i** $= 1, 2, 3, 4$
- **rotation** – function which shifts a given w_i half-byte to the left.
- $\mathbf{y_i}$ – stands for x^{i-1} reduced by the polynomial $m(x)$ [3]. In other words, we can write down y_i as:

$$y_i = \begin{pmatrix} 2^{i-1} \\ 0 \end{pmatrix} \tag{1.8}$$

Finally, the other *round keys* k_i are obtained from w array as follows ($i = 1, 2, 3, 4$):

$$k_i = \begin{pmatrix} w_{2i} & w_{2i+1} \end{pmatrix} \tag{1.9}$$

[3] $m(x) = 1 + x + x^4$

1.7 Implementation

We implemented Baby Rijndael cipher in a **C++** environment, using version **C++11** of the standard. All source codes and executable files are available in the attached CD.

Implementation of Baby Rijndael is handled by class **`BabyRijndael`** defined in **`babyrijndael.h`** and implemented in **`babyrijndael.cpp`**.

Our public methods:

- **`BabyRijndael()`** – is an empty constructor.
- **`sbox()`** – performs the operation *SubBytes* on *state*.
- **`shiftRows()`** – performs the operation *ShiftRows* on *state*.
- **`mixColumns()`** – performs the operation *MixColumns* on *state*.
- **`addRoundKey(int round)`** – performs the operation *AddRoundKey* on *state*.
- **`makeKeys()`** – performs the *Key schedule* operation.
- **Inverse methods** – inverse versions of Baby Rijndael's operations.
- **Printing methods** – print various information about cipher's state.

Our private variables:

- **`state[]`** – represents the *state* of Baby Rijndael. We chose to implement it as an array of two 8 bit long integers.
- **`key[]`** – represents the *main key* of cipher. In other words, `w[0]` and `w[1]` of the key schedule.
- **`w[]`** – represents the array of *columns of round keys*. Retrieved from *Key schedule*. This means we have an array of ten 8 bit long integers.

There are some other *helper functions* and versions of Baby Rindael's operations (`SubBytes`,...) not bounded by class. Methods are not optimized for speed. Generally, operations are implemented as defined in the Baby Rijndael specification. `SubBytes` (`sbox()`) uses a table look-up.

Chapter 2

Cryptanalysis

2.1 Definition

Cryptanalysis is focused on analyzing systems and their secrets. The secret can be a key with which the system is encrypted.

Cryptanalyst, the person performing the cryptanalysis, is not limited to study cipher only from a mathematical angle, but from every direction including the use of side-channel attack (e.g. power consumption), which is an attack on a specific implementation.

2.2 Availability of Plaintext and Ciphertext

Depending on the extent of access to cipher by analyst, we have numerous types of analyses:

- **Known-plaintext analysis** – the analyst has access to some plaintexts and their corresponding ciphertexts. The secret key is then deduced from this information.
  - *Simple known-plaintext analysis could be performed on the **Caesar cipher**[4], where each letter of plaintext is shifted a defined number of places (that is a key) alphabetically and that shifting resulting in final ciphertext.*
- **Chosen-plaintext analysis** – the analyst can choose any plaintext and put it through the cipher to get the corresponding ciphertext. The key is still unknown.
  - *The best known chosen-plaintext analyses are part of **differential cryptanalysis** category; a detailed description is presented in section 2.4.4.*
- **Ciphertext-only analysis** – the ciphertext is known to analyst but the plaintext is unknown. Most of the time the style or phrasing of the plaintext is known, too. The analyst has to work with this information to decrypt the ciphertext or deduce the key.

[4]Caesar cipher – more on this can be found at: `http://practicalcryptography.com/ciphers/caesar-cipher/`

– *The best known case of a successful ciphertext-only analysis is the breaking of the* ***Enigma*** *code.*

2.3 Notation

In this section we introduce the notation which is used throughout whole thesis. For:

Input we will use $\mathbf{X}$, which consists of individual bits $[\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}..\mathbf{X_U}]$.
Output we will use $\mathbf{Y}$, which consists of individual bits $[\mathbf{Y_1}, \mathbf{Y_2}, \mathbf{Y_3}..\mathbf{Y_U}]$.

Exclusive or we will use symbol $\oplus$ or word **XOR**.
Logical conjunction we will use symbol $*$ or word **AND**.

Difference between two texts (plaintexts or ciphertexts) we will use $\mathbf{\Delta}$ with the meaning $\mathbf{\Delta X''} = \mathbf{X} \oplus \mathbf{X'}$.

2.4 Types of cryptanalysis

Common types of cryptanalysis for *symmetric block ciphers* are the **linear**, **differential** and **algebraic** cryptanalysis. In the following subsections we briefly explain these types. For a better understanding of these analyses we define **brute-force** and **side-channel** attacks as well.

2.4.1 Brute-force attack

A type of attack in which an analyst is trying every possible option (key). This approach is infallible, but extremely time consuming.

The main obstacle to this approach is the way time complexity grows with an increasing key size – if the key is **4 bits long** we have to go through **8 guesses in the average case** / 16 guesses in the worst case. If the key is **8 bits long** (2 times longer than the previous key) we have to go through **128 guesses in the average case** / 256 guesses in the worst case. The increase is, as we can notice, not linear (2 times larger, 2 times more time consuming), but exponential (2 times larger, 16 times more time consuming).

2.4.2 Side-channel attack

A type of an attack where analysts are not exploiting the algorithm itself but its specific implementation, hardware or software.

Typically, we exploit the timing and power consumption of algorithms. Most of the time we use this information with statistical methods to build a successful attack. Further material on this subject can be found in the well-recognized paper by Paul C. Kocher: "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems" [6].

2.4.3 Linear cryptanalysis

Linear cryptanalysis was published in 1993 for the first time. It is a *known-plaintext analysis* in which an analyst is trying to approximate the linear behaviour of the cipher using equations where some bits of plaintext, ciphertext and subkeys are used. Generally, this equation can be written in the form:

$$X_1 \oplus X_2 \oplus ...X_U \oplus Y_1 \oplus Y_2 \oplus ... \oplus Y_V = 0 \tag{2.1}$$

Analyst's final goal is to find an equation with the same form as equation 2.1 and a high (or low) probability of occurrence [7].

Suppose we have 4 random bits X_1, X_2, X_3, X_4. Then the probability of this equation

$$X_1 \oplus X_2 \oplus X_3 \oplus X_4 = 0 \tag{2.2}$$

holding true is

$$\Pr(even\ number\ of\ ones) = 1/2, \tag{2.3}$$

provided that these bits are truly random. But if they are not, then the probability has a value different from **1/2**. Difference between 1/2 and the actual value is called a **bias (ϵ)**

$$\Pr(even\ number\ of\ ones) = 1/2 + \epsilon \tag{2.4}$$

and the *linear cryptanalysis* is working exactly with these **biases** where $\epsilon \in \langle -1/2, 1/2 \rangle$.

Suppose a **bias** is distant enough from 0, then the *non-linear component* of cipher can be replaced with a *linear equation* [1, page 28].

Simple example:
Imagine that our cipher consists only of one *substitution box* which needs three bits on *input* (X_1, X_2, X_3) and gives us three bits on *output* (Y_1, Y_2, Y_3). Substitution goes as follows:

Input	000	001	010	011	100	101	110	111
Output	011	110	000	001	111	010	100	101

Table 2.1: Linear cryptanalysis example – Substitution table.

We try some equations with same form as *equation 2.1*:

$$\begin{aligned} X_1 \oplus X_3 \oplus Y_2 &= 0 \\ X_1 \oplus X_2 \oplus Y_2 &= 0 \\ X_3 \oplus Y_1 \oplus Y_2 \oplus Y_3 &= 0 \\ X_1 \oplus Y_1 &= 0 \\ X_2 \oplus Y_2 &= 0 \end{aligned} \tag{2.5}$$

and we get this table with their truth values and biases:

Input	000	001	010	011	100	101	110	111	**Pr**	**Bias**
Output	011	110	000	001	111	010	100	101		
$X_1 \oplus X_3 \oplus Y_2 = 0$	0	1	1	0	1	0	0	1	1/2	0
$X_1 \oplus X_2 \oplus Y_2 = 0$	0	0	0	0	1	1	1	1	1/2	0
$X_3 \oplus Y_1 \oplus Y_2 \oplus Y_3 = 0$	1	0	1	1	0	1	0	0	1/2	0
$X_1 \oplus Y_1 = 0$	1	0	1	1	1	0	1	1	3/4	1/4
$X_2 \oplus Y_2 = 0$	0	0	0	0	0	0	0	0	0/8	-1/2

Table 2.2: Linear equations table.

Table 2.2 shows that our cipher is prone to *linear cryptanalysis*, because we found some equations with *bias* much higher than 0, the best one being the last one with *bias* equal to $-1/2$ a so-called affine approximation.

2.4.4 Differential cryptanalysis

We have now introduced the basic principle of *linear cryptanalysis*. **Differential cryptanalysis** was revealed in the late 1980's[5] and takes a slightly different approach.

First of all, this type of analysis is part of *chosen-plaintext analysis*, hence it is working with much more information than *linear cryptanalysis*. The core idea behind it is in checking *difference* between two plaintexts and their state in the last round of a cipher, resulting in the difference of *output* Y (ΔY) derived from the difference of *input* X (ΔX), in other words **differential** ($\Delta X, \Delta Y$). The occurrence of this differential among others has probability equal to:

$$\Pr(occurrence\ of\ (\Delta X, \Delta Y)) = 1/2^n \tag{2.6}$$

where n is the number of bits of input X.

However, in real world ciphers it is common to get this *probability* with much higher value then $1/2^n$.

Differential cryptanalysis is working exactly with those cases.

Simple example:
Assume that our cipher consist only of one *substitution box* which needs two bits on *input* and gives us two bits on *output*. Substitution goes as follows:

Input	00	01	10	11
Output	11	10	00	01

Table 2.3: Differential cryptanalysis example – Substitution table.

Consider the *input difference* = 01. It is possible to get this *difference* with 4 pairs of *input plaintexts*.

00 and 01
01 and 00
10 and 11
11 and 10

Putting these *inputs* through the *substitution box*:

$$\begin{array}{cccc} 00 \to 11 & 01 \to 10 & 10 \to 00 & 11 \to 01 \\ 01 \to 10 & 00 \to 11 & 11 \to 01 & 10 \to 00 \\ 11 \oplus 10 = 01 & 10 \oplus 11 = 01 & 00 \oplus 01 = 01 & 01 \oplus 00 = 01 \end{array}$$

always results in the *output difference* = 01.

[5]For more on history of *differential cryptanalysis* go to: `http://www.liquisearch.com/differential_cryptanalysis/history`.

Probabilities are as follows:

$$\begin{aligned} Pr(\Delta 01, \Delta 00) = 0 \\ Pr(\Delta 01, \Delta 01) = 1 \\ Pr(\Delta 01, \Delta 10) = 0 \\ Pr(\Delta 01, \Delta 11) = 0 \end{aligned}$$

By calculating the *probabilities* of other individual *differentials* we get:

		Out Δ			
		00	*01*	*10*	*11*
In Δ	*00*	1.0	0	0	0
	01	0	1.0	0	0
	10	0	0	0	1.0
	11	0	0	1.0	0

Table 2.4: Differential cryptanalysis example – probabilities of individual differentials.

This *substitution box* is prone to the *differential cryptanalysis* because *probabilities* are much higher than 1/4 of the ideal case. Specifically, in our case we always get just one *output difference* for given *input difference*. *Probability* in those cases is 1.

2.4.5 Algebraic cryptanalysis

The main idea behind this type of *cryptanalysis* is to rewrite the cipher into a form of system of polynomial equations over finite fields [8] and then solve these equations.

Substantial issue with this approach is the fact that solving a system of polynomial equations over finite fields is a NP-hard problem [8, page 199-202]. Recognizing this issue, analysts have to try to get the shortest possible system of equations.

Simple example:
Assume we have got two LFSRs[6]:

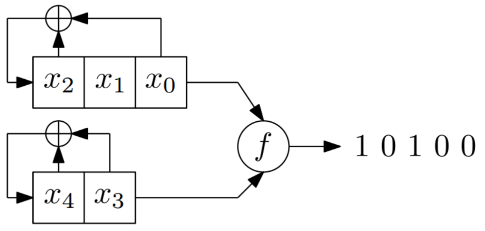

Figure 2.1: An example of two LFSRs [2, page 10].

and a non-linear function

$$f(o_1, o_2) = o_1 \oplus (o_1 * o_2) \tag{2.7}$$

where o_1 is *output* from first register (R_1) and o_2 is *output* from second register (R_2).

Our goal is to calculate $\mathbf{x_0..x_4} \in \mathbf{GF(2)}$, knowing the function's output 10100.

When we perform a step by step analysis of the states of this system, we get:

Step	**content of R_1**	**content of R_2**	o_1, o_2
1.	(x_2, x_1, x_0)	(x_4, x_3)	x_0, x_3
2.	$(x_0 \oplus x_2, x_2, x_1)$	$(x_3 \oplus x_4, x_4)$	x_1, x_4
3.	$(x_0 \oplus x_1 \oplus x_2, x_0 \oplus x_2, x_2)$	$(x_3, x_3 \oplus x_4)$	$x_2, x_3 \oplus x_4$
4.	$(x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2, x_0 \oplus x_2)$	(x_4, x_3)	$x_0 \oplus x_2, x_3$
5.	$(x_1 \oplus x_2, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2)$	$(x_3 \oplus x_4, x_4)$	$x_0 \oplus x_1 \oplus x_2, x_4$

Table 2.5: States of registers and their outputs [2, page 11].

[6]LFSR – Linear-feedback shift register; more at `https://en.wikipedia.org/wiki/Linear-feedback_shift_register`.

Now we can substitute *outputs* from the equation 2.7 with calculated ones from table 2.5 and we get **a system of five equations**:

$$
\begin{aligned}
x_0 \oplus (x_0 * x_3) = 1 \\
x_1 \oplus (x_1 * x_4) = 0 \\
x_2 \oplus (x_2 * x_3) \oplus (x_2 * x_4) = 1 \\
x_0 \oplus x_2 \oplus (x_0 * x_3) \oplus (x_2 * x_3) = 0 \\
x_0 \oplus x_1 \oplus x_2 \oplus (x_0 * x_4) \oplus (x_1 * x_4) \oplus (x_2 * x_4) = 0
\end{aligned}
\tag{2.8}
$$

Figure 2.2: Algebraic cryptanalysis equations [2, page 12].

Solution of these gives us the desired *key*, or the initial state of the registers:

$$x_0 = 1, x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0 \tag{2.9}$$

Chapter 3

Impossible differential cryptanalysis

As the title suggests, *impossible differential cryptanalysis* is a special kind of *differential cryptanalysis*, where an analyst is not looking for *differentials* with high probability, but rather for those with **zero probability**. Hence the word *impossible*.

This variant of *differential cryptanalysis* was described for the first time by Biham, Biryukov and Shamir [9] in 1999 and was focused on the *Skipjack cipher*[7]. We should point out that *impossible events* in a cipher were used a few times before – so it was not a totally new idea, but the combination of those *impossible events* and methods of a *differential cryptanalysis* was formally described only then.

3.1 History

After the first success with this technique, the team of Biham, Biryukov and Shamir continued analyzing other ciphers. In the same year (1999) they applied *impossible differential cryptanalysis* to IDEA and Khufu in a paper: "Miss in the Middle Attacks on IDEA and Khufu" [10]. They achieved considerable improvement over then-current cryptanalytic results:

IDEA

Before:	**Paper:**
3.5 rounds	3.5 rounds
truncated-differential	*impossible differential*
time: 2^{67} units	time: 2^{53} units
2^{56} chosen plaintexts	$2^{38.5}$ chosen plaintexts

The following year (2000) Biham with Keller published a paper in which they described improved attacks on Rijndael, at that time an AES candidate, with the name: "Cryptanalysis of Reduced Variants of Rijndael" [11]. One of the improved attacks was *impossible differential.*

[7]Skipjack - more at `http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf`.

Rijndael

Before:	**Paper:**
5 rounds	5 rounds
square	*impossible differential*
time: 2^{40} units	time: 2^{31} units
2^{11} chosen plaintexts	$2^{29.5}$ chosen plaintexts

In December of 2001, a team of Cheon, Kim M., Kim K., Lee and Kang extended previous *impossible differential* attack on Rijndael to 6 rounds in a paper: "Improved Impossible Differential Cryptanalysis of Rijndael and Crypton" [12]. This attack was more time and space demanding than the best previously known 6 round attack (square), but still less demanding than brute-force attack.

Rijndael

Before:	**Paper:**
6 rounds	6 rounds
square	*impossible differential*
time: 2^{72} units	time: 2^{122} units
2^{32} chosen plaintexts	$2^{91.5}$ chosen plaintexts

Another team, Kim J., Hong S., Sung J., Lee S., Lim J. and Sung S. then (2003) wrote a paper about *impossible differential cryptanalysis* of block ciphers: "Impossible Differential Cryptanalysis for Block Cipher Structures" [13]. They focused on introducing method for finding impossible characteristics in ciphers with bijective round functions.

In the same year (2003), Raphael C.-W. Phan extended previous works on Rijndael cipher by introducing an attack on 7 rounds of Rijndael in paper: "Impossible differential cryptanalysis of 7-round Advanced Encryption Standard (AES)" [14]. It was possible due to exploiting weaknesses in the AES key schedule.

Rijndael

Paper (AES-192):	**Paper (AES-256):**
7 rounds	7 rounds
impossible differential	*impossible differential*
time: 2^{186} units	time: $2^{250.5}$ units
2^{92} chosen plaintexts	$2^{92.5}$ chosen plaintexts

This attack (*impossible differential*) was then further improved by many papers. We would like to mention one other paper from past few years, specifically from year 2010, in which Mala H., Dakhilalian M., Rijmen V. and Modarres-Hashemi M. greatly improved time complexity of the attack to 7 rounds of Rijndael (AES-128): "Improved Impossible Differential Cryptanalysis of 7-Round AES-128" [15]. Results:

Rijndael
Paper (AES-128):
7 rounds
impossible differential
time: $2^{110.2}$ units
$2^{106.2}$ chosen plaintexts

3.2 Basic principle

As we mentioned, *impossible differential cryptanalysis* builds on never-occurring differences. Which means that if particular cipher is decrypted with the correct key, there is no chance of that difference arising. Suppose we have a pair of texts (plaintexts and corresponding ciphertexts). Then we are guessing some (round) keys and decrypting those ciphertexts to a predetermined place in cipher. If the difference of those texts belongs to the set of *impossible differentials* then we can surely say that the guessed key is not the right one. This way we eliminate all the incorrect keys and we are left with only correct one.

3.2.1 Example

Imagine that one round of a cipher consist of only one *substitution box* which needs three bits on *input* and gives us three bits on *output*, a simple *permutation* and finally the *addition of a round key.* Our cipher has two rounds. The substitution goes as follows:

Input	000	001	010	011	100	101	110	111
Output	011	110	000	001	111	010	100	101

Table 3.1: Impossible differential cryptanalysis example – Substitution table.

The *permutation* goes as follows:

$$\begin{aligned} 1 &\rightarrow 2 \\ 2 &\rightarrow 3 \\ 3 &\rightarrow 1 \end{aligned}$$

In other words, the *permutation* shifts one bit to the right.

Our secret key is 6 bits long and it is divided into two 3 bit long parts. The First (Second) part serves as the round key for first (second) round.

Suppose we choose **010101** as our key. The first round key is therefore **010** and the second round key is **101**. Then we got two plaintexts **111** and **001**. To write this information down more clearly:

$$\mathbf{P_0 = 111}$$
$$\mathbf{P_1 = 001}$$
$$----$$
$$\mathbf{k_0 = 010}$$
$$\mathbf{k_1 = 101}$$

1st round:
We are starting with the *input difference* equal to:

$$\boldsymbol{\Delta}_{\mathbf{in}} = P_0 \oplus P_1 = \mathbf{110}$$

Then we transform the plaintexts with the **substitution**:

$$P_0^{'} = \mathrm{S}(P_0) = \mathrm{S}(111) = 101$$
$$P_1^{'} = \mathrm{S}(P_1) = \mathrm{S}(001) = 110$$

Next we put the result of previous step through **permutation**:

$$P_0^{''} = \mathrm{Perm}(P_0^{'}) = \mathrm{Perm}(101) = 110$$
$$P_1^{''} = \mathrm{Perm}(P_1^{'}) = \mathrm{Perm}(110) = 011$$

And finally we perform the **addition of a round key**:

$$P_0^{'''} = P_0^{''} \oplus k_0 = 110 \oplus 010 = 100$$
$$P_1^{'''} = P_1^{''} \oplus k_0 = 011 \oplus 010 = 001$$

Difference after the first round:

$$\Delta_{btwn} = P_0^{'''} \oplus P_1^{'''} = 101$$

Note: We can see that the **difference did not change with the key addition**.

2nd round:
Substitution:

$$R_0^{'} = \mathrm{S}(P_0^{'''}) = \mathrm{S}(100) = 111$$
$$R_1^{'} = \mathrm{S}(P_1^{'''}) = \mathrm{S}(001) = 110$$

Permutation:

$$R_0'' = \mathrm{Perm}(R_0') = \mathrm{Perm}(111) = 111$$
$$R_1'' = \mathrm{Perm}(R_1') = \mathrm{Perm}(110) = 011$$

Addition of the round key:

$$R_0''' = R_0'' \oplus k_1 = 111 \oplus 101 = 010$$
$$R_1''' = R_1'' \oplus k_1 = 011 \oplus 101 = 110$$

Output difference:

$$\Delta_{out} = R_0''' \oplus R_1''' = 100$$

Now we try to determine which differences are even possible after the first round of our cipher. For this we should analyze *substitution* and *permutation*.
Note: Addition of a round key does not change *difference*. Because we **XOR** same values of the key to both plaintexts. For the further explanation see section 4.4.

Note: We perform the same analysis as in the *simple example* from section 2.4.4.

		Δ_{out}							
		000	*001*	*010*	*011*	*100*	*101*	*110*	*111*
Δ_{in}	*000*	1.0	0	0	0	0	0	0	0
	001	0	0.5	0	0	0	0.5	0	0
	010	0	0	0	0.5	0	0	0	0.5
	011	0	0	0.5	0	0	0	0.5	0
	100	0	0	0	0	1.0	0	0	0
	101	0	0.5	0	0	0	0.5	0	0
	110	0	0	0	0.5	0	0	0	0.5
	111	0	0	0.5	0	0	0	0.5	0

Table 3.2: Impossible differential cryptanalysis example – probabilities of individual differentials (*Substitution box*).

Permutation just shifts the difference one position to the right.

Notice (from the table 3.2) that we have **fifty impossible differentials** for *substitution*. Recall that *difference* from *substitution box* is then shifted with *permutation*. If we have (after *substitution*) *difference* 101, then (after *permutation*) we obtain *difference* 110.

Finally, after the whole first round we still have **fifty impossible differentials**, but shifted one position to the right because of the *permutation*.

Now we **guess a second round key** and decrypt the ciphertexts to the start of the second round. After that we compare real obtained difference with those from the *impossible differentials* (with corresponding *input difference*) and in case of a match, it is apparent that the guessed key is wrong.

Decryption:

$$C_0 = 010$$
$$C_1 = 110$$

Addition of a guessed round key:

$$\mathbf{k_g} = \mathbf{100}$$
$$----$$
$$R_0^{''} = C_0 \oplus k_g = 010 \oplus 100 = 110$$
$$R_1^{''} = C_1 \oplus k_g = 110 \oplus 100 = 010$$

Inverse permutation:

$$R_0^{'} = \text{Perm}^{-1}(R_0^{''}) = \text{Perm}^{-1}(110) = 101$$
$$R_1^{'} = \text{Perm}^{-1}(R_1^{''}) = \text{Perm}^{-1}(010) = 100$$

Inverse substition:

$$P_0^{'''} = \text{S}^{-1}(R_0^{'}) = \text{S}^{-1}(101) = 111$$
$$P_1^{'''} = \text{S}^{-1}(R_1^{'}) = \text{S}^{-1}(100) = 110$$

Difference:

$$\mathbf{\Delta_{btwn}} = P_0^{'''} \oplus P_1^{'''} = \mathbf{001}$$

We know that after the first round with the **input difference = 110**, we can possibly obtain only these differences:

$$\Delta_0 = 101$$
$$\Delta_1 = 111$$

Everything else is **impossible**. We list the *impossible differentials* for this *input difference (110)* after one round for better clarity:

$$(110, \mathbf{000}), (110, \mathbf{100}), (110, \mathbf{001}),$$
$$(110, \mathbf{010}), (110, \mathbf{110}), (110, \mathbf{011})$$

At this point we are able to determine whether the calculated difference belongs to set of *impossible differences.*

Comparing differences:

$$\begin{gathered}\Delta_{btwn} = 001\\ \Delta_{btwn} \in \textit{impossible differences}\\ \Downarrow\\ \text{The } \textit{guessed key (100)} \text{ is } \textbf{wrong}.\end{gathered}$$

We guessed a **wrong key**, which means we can delete it and guess another. This way we narrow down the key space. If we exhaust all possibilities, then we can choose another *input difference* or the same *input difference* with other plaintexts and perform the analysis again.

3.2.2 Notes about example

Clarifying the selection of example: *substitution* and *permutation* were chosen at random, but the existence of the *permutation* in our cipher was substantial.

Imagine this example without *permutation*, but with the same *substitution.* We could have guessed whatever key, but we would never get to the *impossible difference.* As follows:

1. We would guess the key. **Difference remains the same**, i.e. is still possible.
2. We would then do *inverse substitution.* But the difference would change only in possible way (see table 3.2). **Difference remains possible**.

It is necessary to have some operation like *permutation* to set off the *difference.* To explain it differently: go to the *inverse substitution* with *unanticipated difference* (considering input difference).

Chapter 4

Impossible differential cryptanalysis of Baby Rijndael

In this section we use information from previous chapters to deploy an attack on the chosen cipher – Baby Rijndael. We introduced Baby Rijndael cipher, including its implementation, in chapter 1. We characterized its operations: **`SubBytes`**, **`ShiftRows`**, **`MixColumns`** and **`AddRoundKey`**, which is significant for further understanding and cryptanalysis. This implementation is then used for **impossible differential cryptanalysis** in class **`Differential`**, which contains methods needed for an attack. These will be described more in depth in the following sections.

4.1 Analysis of SubBytes

We perform the same analysis as in the example from section 2.4.4 (Differential cryptanalysis) in table 2.4 and from section 3.2 (Impossible differential cryptanalysis – Basic Principle) in table 3.2. Simply put, we want to know the *possible / impossible differentials*. In other words: we are searching for **probabilities of individual differentials**.

We implemented this functionality in the method **`sboxDiffs(int inverse)`**, which tries every possible input pair (difference) and logs what difference we get on output. Recall that **`SubBytes`** in Baby Rijndael has 4 bits on input and gives us 4 bits on output. After performing the analysis, we obtain a table with 16 rows and 16 columns (4 bits give $2^4 = 16$ possibilities).

Information known before the result:

- **Differential (0000,0000)** – should have 100% probability, because every pair of a same input text (without difference) is transformed to same output text (without difference), due to bijectivity of operation `SubBytes`.
- **Probability will always take the form: (even number/16)** – we know, that the order of a pair does not affect output difference – that is, the *difference* of (A, B) is the same as the *difference* of (B, A) due to the symmetry of the XOR operation.

Final result:

		Δ_{in}															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Δ_{out}	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	2	2	0	2	0	2	0	2	2	0	0	0	0	4	0
	2	0	0	2	2	0	0	0	0	2	4	2	0	2	0	0	2
	3	0	4	0	0	2	0	0	2	2	0	2	0	2	2	0	0
	4	0	2	4	0	0	2	2	2	0	0	2	0	0	0	0	2
	5	0	0	0	0	4	0	2	2	0	2	0	2	2	0	0	2
	6	0	0	0	0	0	2	2	0	4	0	2	2	2	0	2	0
	7	0	0	0	2	0	0	4	2	2	0	0	0	0	2	2	2
	8	0	2	0	2	2	2	0	0	2	0	0	2	0	0	0	4
	9	0	2	0	0	0	2	0	0	0	2	0	0	2	4	2	2
	A	0	2	2	2	0	0	2	0	0	0	0	2	4	2	0	0
	B	0	0	2	2	2	4	0	2	0	0	0	0	2	0	2	0
	C	0	0	0	2	2	2	2	0	0	2	4	0	0	2	0	0
	D	0	0	2	0	2	0	0	0	0	0	2	4	0	2	2	2
	E	0	2	0	4	0	0	0	2	0	2	2	2	0	0	2	0
	F	0	0	2	0	0	2	0	4	2	2	0	2	0	2	0	0

Table 4.1: Probabilities of individual differentials (Baby Rijndael's SubBytes).

Legend:

- **All numbers (probabilities) are shown as nominator to denominator 16.**
- **For example:** $2 \rightarrow 2/16 = 0.125$

Because of the method of calculation of probabilities, this table could be used for *differential cryptanalysis* as well. However, for our purpose we only need to know if individual *differential* is possible or not. There is no need to know specific values of probability.

Observation:

Every *input difference* has precisely seven possibilities of *output difference.*

Original output file from method `sboxDiffs(int inverse)`, which was further processed to form the table 4.1, is located in `src/results/sbox-diffs.txt`.

4.2 Analysis of ShiftRows

Recalling the structure of a state of Baby Rijndael

$$A = \begin{pmatrix} a_0 & a_2 \\ a_1 & a_3 \end{pmatrix},\ a_i \in GF(2^4)$$

and the knowledge, that operation **`ShiftRows`** switches positions a_1 and a_3, then we can easily conclude, that same happens to difference of texts.

Therefore there is no need for a special method for handling difference after **`ShiftRows`**, but we need to keep it in mind for computation of *impossible differentials*.

Example:

Suppose we have a pair of plaintexts:

$$\begin{aligned} P_0 &= 0101\ 1010\ 1110\ 0100 \\ P_1 &= 0110\ 1110\ 0010\ 0001 \end{aligned}$$

and their *input difference*:

$$\begin{aligned} \Delta_{in}(P_0, P_1) &= P_0 \oplus P_1 \\ &= 0101\ 1010\ 1110\ 0100 \oplus 0110\ 1110\ 0010\ 0001 \\ &= 0011\ 0100\ 1100\ 0101 \end{aligned}$$

Now we put those plaintexts through operation **`ShiftRows`**:

$$\begin{aligned} \text{ShiftRows}(P_0) &= 0101\ 0100\ 1110\ 1010 \\ \text{ShiftRows}(P_1) &= 0110\ 0001\ 0010\ 1110 \end{aligned}$$

and calculate their *output difference* after this operation:

$$\begin{aligned} \Delta_{out}(\text{ShiftRows}(P_0), \text{ShiftRows}(P_1)) &= \text{ShiftRows}(P_0) \oplus \text{ShiftRows}(P_1) \\ &= 0101\ 0100\ 1110\ 1010 \oplus 0110\ 0001\ 0010\ 1110 \\ &= 0011\ 0101\ 1100\ 0100 \end{aligned}$$

Output difference is just *input difference* after operation **`ShiftRows`**.

$$\begin{aligned} \text{ShiftRows}(\Delta_{in}) &= 0011\ 0101\ 1100\ 0100 \\ \Delta_{out} &= 0011\ 0101\ 1100\ 0100 \end{aligned}$$

4.3 Analysis of MixColumns

Operation `MixColumns` is handled by method `mixColumnsDiffs(int inverse)`. Before further explanation, please recall the changed view of *state* from section 1.4. Meaning, we put **8 bits on input and we get 8 bits on output**.

For implementation we utilize the knowledge that we can put *difference* through operation `MixColumns` the same way as *state* and we get *output difference* (*differential*).

Note: Notation "5D" means two hexadecimal numbers 5 and D in a form of 8 consecutive bits. This example would be **5D** = **01011101**.

Example:

Suppose we have this pair of texts:

$$\mathbf{P_0} = \mathbf{10101010}\ (AA)$$
$$\mathbf{P_1} = \mathbf{01001000}\ (48)$$

They have *input difference*:

$$\mathbf{\Delta_{in}} = \mathbf{11100010}\ (E2).$$

Now we put them through operation `MixColumns`:

$$\mathbf{MixColumns(P_0)} = \text{MixColumns}(10101010) = \mathbf{11111111}$$
$$\mathbf{MixColumns(P_1)} = \text{MixColumns}(01001000) = \mathbf{01011111}$$

Output difference:

$$\mathbf{\Delta_{out}} = 11111111 \oplus 01011111 = \mathbf{10100000}$$

Now we try to put the *input difference* itself through operation `MixColumns`.

$$\mathbf{MixColumns(\Delta_{in})} = \text{MixColumns}(11100010) = \mathbf{10100000}$$

Apparently it does not matter whether we transfer a pair of texts (with some *input difference*) and then calculate their *output difference* or put the *input difference* itself through `MixColumns`.

Finally, because of 8 bits on input, we end up with list of 2^8 = **256** one-to-one transformations.

The result of analysis (transformation table) is shown in table 4.2. The original output file from method `mixColumnsDiffs(int inverse)` is in `src/results/mixDiffs.txt`.

How to read the table 4.2:

`MixColumns` requires 8 bits on input and these 8 bits are composed of two 4 bit long positions. We used those positions to better navigate through table.

Reading:

$$\begin{aligned}\textbf{Input A7} &= 1st\ position\ \mathbf{A}\\ &= 2nd\ position\ \mathbf{7}\\ \textbf{Output} &= \mathbf{13}\end{aligned}$$

		2nd position															
		0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***A***	***B***	***C***	***D***	***E***	***F***
1st position	***0***	00	D5	9A	4F	17	C2	8D	58	2E	FB	B4	61	39	EC	A3	76
	1	5D	88	C7	12	4A	9F	D0	05	73	A6	E9	3C	64	B1	FE	2B
	2	A9	7C	33	E6	BE	6B	24	F1	87	52	1D	C8	90	45	0A	DF
	3	F4	21	6E	BB	E3	36	79	AC	DA	0F	40	95	CD	18	57	82
	4	71	A4	EB	3E	66	B3	FC	29	5F	8A	C5	10	48	9D	D2	07
	5	2C	F9	B6	63	3B	EE	A1	74	02	D7	98	4D	15	C0	8F	5A
	6	D8	0D	42	97	CF	1A	55	80	F6	23	6C	B9	E1	34	7B	AE
	7	85	50	1F	CA	92	47	08	DD	AB	7E	31	E4	BC	69	26	F3
	8	E2	37	78	AD	F5	20	6F	BA	CC	19	56	83	DB	E	41	94
	9	BF	6A	25	F0	A8	7D	32	E7	91	44	0B	DE	86	53	1C	C9
	A	4B	9E	D1	04	5C	89	C6	13	65	B0	FF	2A	72	A7	E8	3D
	B	16	C3	8C	59	01	D4	9B	4E	38	ED	A2	77	2F	FA	B5	60
	C	93	46	09	DC	84	51	1E	CB	BD	68	27	F2	AA	7F	30	E5
	D	CE	1B	54	81	D9	0C	43	96	E0	35	7A	AF	F7	22	6D	B8
	E	3A	EF	A0	75	2D	F8	B7	62	14	C1	8E	5B	03	D6	99	4C
	F	67	B2	FD	28	70	A5	EA	3F	49	9C	D3	06	5E	8B	C4	11

Table 4.2: Transformation of the difference during Baby Rijndael's `MixColumns`.

4.4 Analysis of AddRoundKey

Analysis of operation `AddRoundKey` is unnecessary, because it does not change the *difference*.

Example:

Suppose we have a pair of plaintexts and their *difference*:

$$P_0 \text{ and } P_1$$
$$\Delta(P_0, P_1) = P_0 \oplus P_1$$

If we **XOR** the same key to both of them and then calculate their *difference* we obtain:

$$\begin{aligned}&(P_0 \oplus k) \oplus (P_1 \oplus k)\\ &= P_0 \oplus P_1 \oplus k \oplus k\\ &= (P_0 \oplus P_1) \oplus (k \oplus k)\\ &= P_0 \oplus P_1\end{aligned}$$

XOR of a key does not make any change to the *difference* of plaintexts.

4.5 Conjuction of SubBytes and MixColumns

For the purpose of our cryptanalysis we decided to join **`SubBytes`** with **`MixColumns`** into a single operation. This means we had to move operation **`ShiftRows`** to the start of a round. The reason why we are able to do that is justified in [1, page 42-43]. Now we explain what we gain from this move.

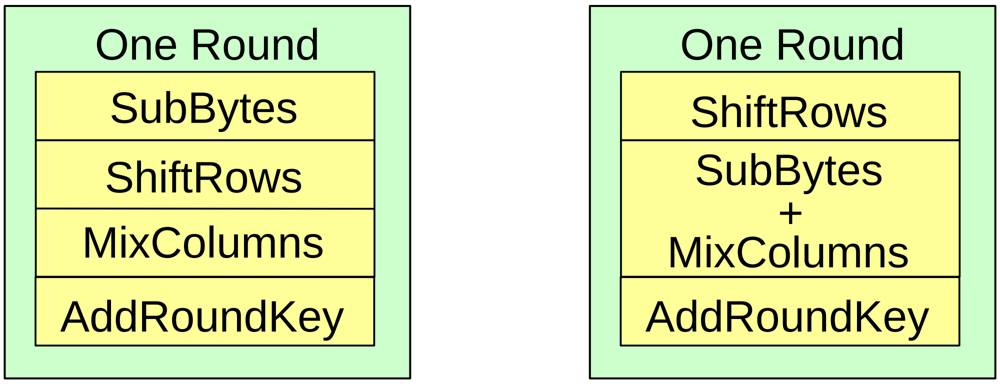

Figure 4.1: One round of Baby Rijndael for the purpose of our analysis – original and changed.

Suppose we think about *differentials* in the original sequence of operations, **`ShiftRows`** would have interchange *differences* between two halves and our table would have all $2^{16} = 65536$ possibilities on input. In contrast, we can easily compute *differentials* for operation **`SubBytes+MixColumns`** without thinking about **`ShiftRows`**, because this joint operation has only $2^8 = 256$ possibilities on input. Information gained from the result table of *differentials* can be used for the first half (8bits) and second half (8bits) of *state* of Baby Rijndael separately.

4.6 Analysis of SubBytes+MixColumns

Note: When we compute *differentials* we calculate the possible ones. ***Impossible differentials*** are computed as final product of cryptanalysis as a **complement** to them.

We implemented this joint operation in method **`sMixDiffs(int inverse)`** where we try every possible pair on input (256 × 256). Method first gets the *difference* of this pair, then puts it through SubBytes and MixColumns and finally gets the *difference* after these operations. Results (*possible differentials*) are saved in **`src/results/sMixDiffsValues.txt`**.

The results give us further understanding of which *input differences* should be analyzed. That is due to shorter list of *possible differentials* in some cases. Notice that this **shorter list** always has **seven differentials**. The reasons for this fact are:

- When we recall the outcome of the analysis of `SubBytes`, we notice that we always have **seven differentials** for a given *input difference.* Because of its bijective characteristic, `MixColumns` only transforms a *difference* to another one.
- Shorter lists are achievable due to one difference (of two 4 bit long positions going into `SubBytes`) being zero. This difference always gives us a zero difference on the output.
- If both 4 bits positions are non-zero, then we get seven possibilities for each of them, so combined it gives us final 7 × 7 = 49 possibilities for longer lists of *possible differentials.*
- **We can conclude that we are only interested in those *input differences to cipher* which have one of 4 bits positions equal to zero. This way we obtain shorter lists of *differentials*, which implies shorter computation time.**
- These interesting differences are: 01, 02, ... , 0F and 10, 20, ... , F0

4.7 Computation of differentials

At this stage of our work we have enough tools and pre-computed values to get *(im)possible differentials* for a whole round or for multiple rounds of Baby Rijndael. In this section we describe this procedure and its implementation, which can be found in method **`computePossible( uint16_t inDiff, int numOfRounds)`**, where arguments are:

- **inDiff** – *Input difference* at the beginning of cipher.
- **numOfRounds** – Number of rounds we want to get the *differentials* of.

Pseudo algorithm:

```
1: $possible\_differences \leftarrow input\_difference$ ; $start \leftarrow 0$ ; $end \leftarrow 1$
2: for $i = 0$ to $number\_of\_rounds$ do
3:   repeat
4:     $difference \leftarrow$ ShiftRows($possible\_differences[start]$)
5:     $first\_column \leftarrow$ first half of $difference$
6:     $second\_column \leftarrow$ second half of $difference$
7:     $start \leftarrow start + 1$
8:     $first\_differences \leftarrow$ analysis of SubBytes+MixColumns($first\_column$)
9:     $second\_differences \leftarrow$ analysis of SubBytes+MixColumns($second\_column$)
10:    for $j = 0$ to $size(first\_differences)$ do
11:      for $k = 0$ to $size(second\_differences)$ do
12:        $possible\_differences \leftarrow first\_differences[j] + second\_differences[k]$
13:      end for
14:    end for
15:  until $start < end$
16:  $end \leftarrow size(possible\_differences)$
17: end for
```

1 We **create a set** with one entry at the start – *input difference*. We create variables *start* and *end* which are used later for going through unprocessed differences from a set of *possible_differences*

2 **Loop a defined number of rounds** (numOfRounds).

3 **Loop** through all ***unprocessed differences***.

4 `Perform ShiftRows` on them.

5+6 We split a *difference* into two columns (of 8bits), **first column** and **second column**.

8+9 We define $first\_differences/second\_differences$ as sets of differences we have from the analysis of `SubBytes + MixColumns`; more precisely – possible output values of difference equal to $first/second\_column$.

10 **Loop** through all output differences of $first\_differences$.

11 **Loop** through all output differences of $second\_differences$.

12 **For every** *output difference* from $second\_differences$: **add the combination** of differences from step 10 and this *output difference* to the end of the **set**.

We will use the knowledge of computation of the *differentials* in the following sections.

Note: We should stress the fact that we do not need to know the **key** for determining *differentials*. That is, we are able to pre-calculate *impossible differentials*, save them and then use them multiple times regardless of round keys which are actually used for encryption. This pre-calculation is a one-time operation.

4.8 Structure of attacks

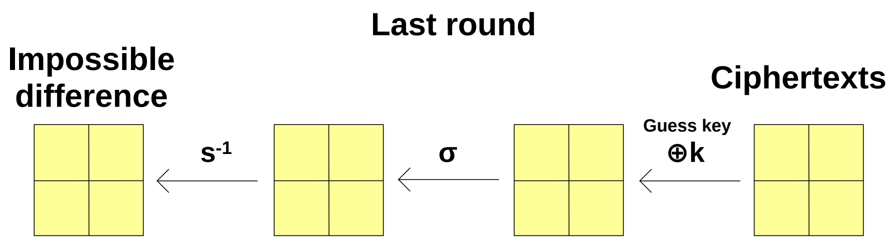

Figure 4.2: Decrypting through the last round of Baby Rijndael for the *impossible differential cryptanalysis.*

Note: We assume that we have performed the calculation of *impossible differentials* beforehand and that we have a pair of plaintexts and corresponding ciphertexts; the attack should recover the key.

All following attacks / analyses have the same structure. At the beginning we **guess a round key** and then we go backwards through the last round of Baby Rijndael, decrypting the ciphertexts. We stop after `SubBytes`$^{-1}$ with some *difference*. This *difference* is then compared with the pre–computed values of *(im)possible differences.* If the *difference* is *impossible*, then the **guessed key** is wrong and we drop it.

4.9 Attack on 2 rounds of Baby Rijndael

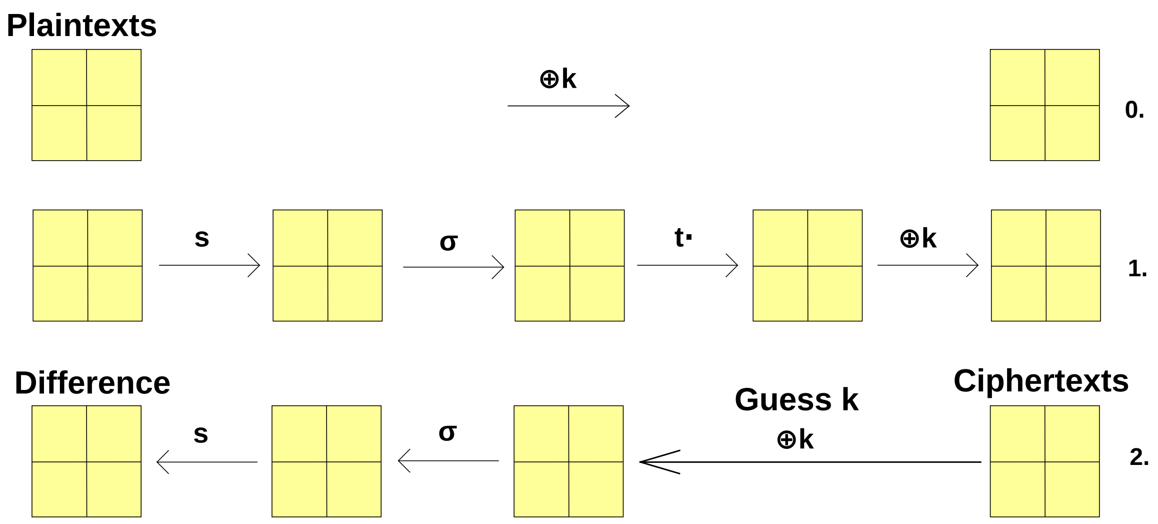

Figure 4.3: Attack on 2 rounds of Baby Rijndael (1st variant).

This simplification of an attack (2 rounds instead of all 4 rounds) allows us to further understand *impossible differential cryptanalysis*. We will later use knowledge gained in this way to mount an attack on full 4 rounds of Baby Rijndael.

In the demonstration of the attacks we always use these round keys:

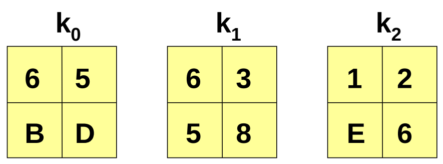

Figure 4.4: Attack on 2 rounds of Baby Rijndael – key schedule.

Note: The operation `ShiftRows` (σ) is, in the Baby Rijndael, equal to its inversion `ShiftRows`$^{-1}$ (σ^{-1}). Because of that, we always use `ShiftRows` (σ) without the inversion symbol.

4.9.1 1st variant of an attack:

We want to know the *(im)possible differentials* after the first round of Baby Rijndael. In our first variant we are working with the original sequence of operations: **`SubBytes, ShiftRows, MixColumns and AddRoundKey`**.

Attack:

1. **We calculate *impossible differentials* for the given *input difference*. See figure 4.5.**

   We start with this simple *input difference*:

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

   Now we use results of the previous analyses (sections 4.1 to 4.6) to determine transformation (or branching) of the *difference*.

   Note: We calculate *impossible differentials* as a complement to *possible differentials* (it is easier in this case).

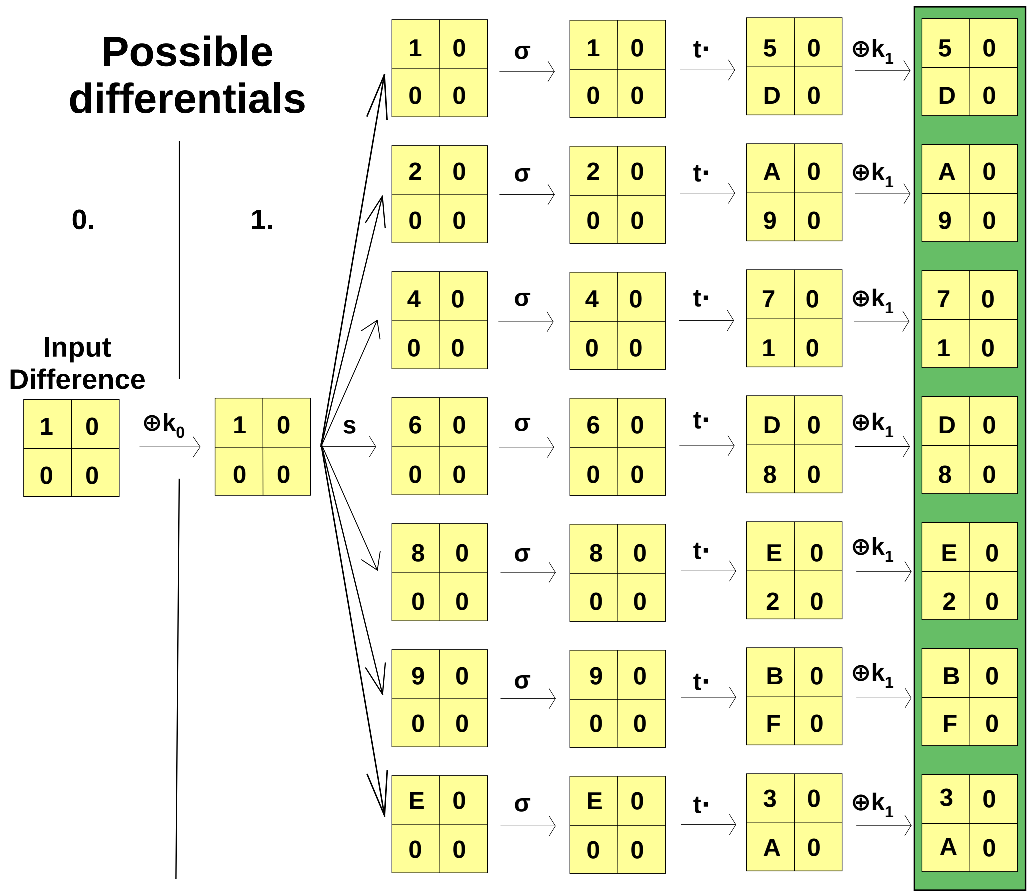

Figure 4.5: Attack on 2 rounds of Baby Rijndael – possible differentials (1st variant).

2. **We select a pair of plaintexts with the chosen *input difference.***

   Our choice:

$$P_0 = \begin{pmatrix} 2 & A \\ C & 5 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 3 & A \\ C & 5 \end{pmatrix}$$

3. **We run them through 2 rounds of Baby Rijndael. Remember that the second round is without `MixColumns`. We recieve a pair of corresponding ciphertexts.**

$$C_0 = \begin{pmatrix} 9 & 2 \\ 3 & 8 \end{pmatrix} \qquad C_1 = \begin{pmatrix} 6 & 2 \\ 3 & 7 \end{pmatrix}$$

4. **We guess the second round key.**

   Our choice:

$$k_2 = \begin{pmatrix} 0 & E \\ 0 & B \end{pmatrix}$$

5. **We decrypt these ciphertexts to the start of the second round with the guessed round key.**

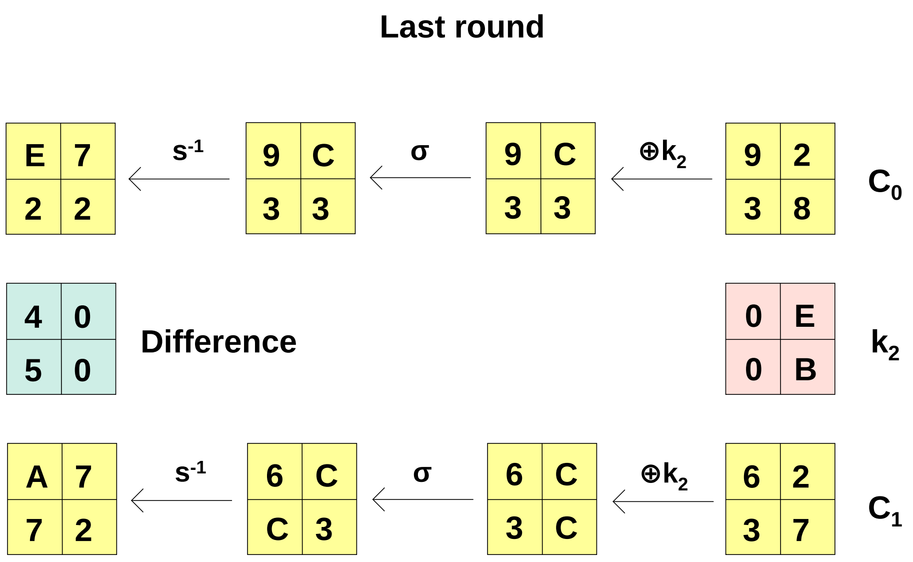

Figure 4.6: Attack on 2 rounds of Baby Rijndael – decrypting the last round (1st variant).

6. **We compare the difference of the decrypted ciphertexts with the pre-calculated set of *impossible differences.***

$$\Delta = \begin{pmatrix} 4 & 0 \\ 5 & 0 \end{pmatrix} \quad \notin \quad \textit{possible differences}$$
$$\Downarrow$$
$$\textbf{We have an } \boldsymbol{\textit{impossible difference.}}$$

7. **If we get an *impossible difference*, we drop this guessed key.**

$$\text{We drop the key } \begin{pmatrix} 0 & E \\ 0 & B \end{pmatrix}.$$

8. **We return to step 4 and guess another round key.**

   After we go through all possible keys, we end up left with 3072 keys[8].

9. **Now we choose different plaintexts and go through attack from step 2 once again.**

   To end up with 256 possible keys, we need to go through **2.45** plaintext pairs in average[9].

[8]Number 3072 is specific to this particular example.
[9]We ran program 100 times and calculated the average number.

4.9.2 2nd variant of an attack:

In the second variant of an attack we work with the changed sequence of operations: `ShiftRows (σ), SubBytes+MixColumns (s + t·) and AddRoundKey (⊕).`

The first noticeable change is in the computation of *(im)possible differentials*, where we use the analysis of operation `SubBytes+MixColumns` from section 4.6.

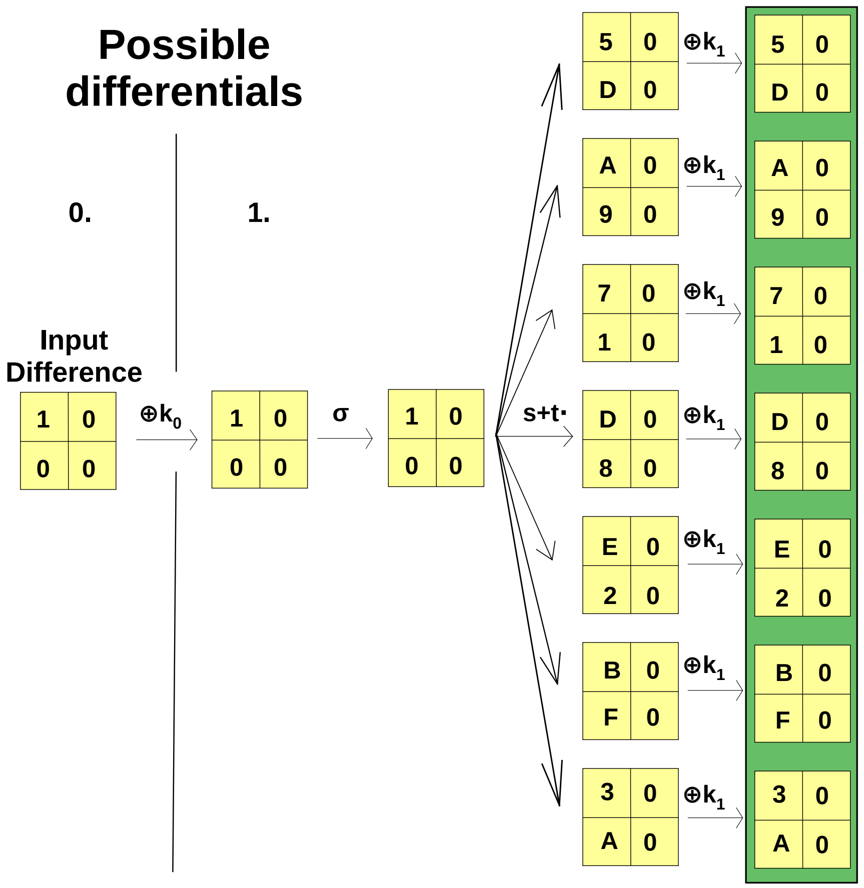

Figure 4.7: Attack on 2 rounds of Baby Rijndael – possible differentials (2nd variant).

Flow of the attack is altered from step 4 of the previous attack (1st variant) from section 4.9.1.

Note: Values, unless stated otherwise, are the same as in the attack from previous section 4.9.1.

Attack:

1. We calculate the *impossible differentials* for the given *input difference*. See figure 4.7.
2. We choose a pair of plaintexts with the defined *input difference*.
3. We run them through 2 rounds of Baby Rijndael. Do not forget that the second round is without `MixColumns`. We receive a pair of corresponding ciphertexts.
4. **We guess a half of the second round key.**

   We choose the first 4 bits (1st position) of the key and the last 4 bits (4th position) of the key. The reason behind is indicated in the figure 4.8 and further explained in the following text. The key only affects the positions printed in pink. As a result of this shorter guess, we decrease the time complexity.

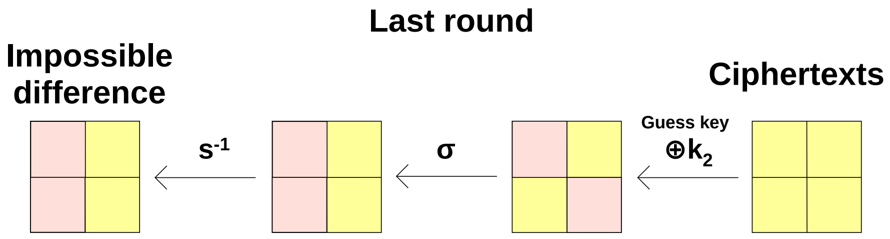

Figure 4.8: Attack on 2 rounds of Baby Rijndael – guessing half of the second round key (2nd variant).

   Our choice:

$$k_2 = \begin{pmatrix} E & 0 \\ 0 & B \end{pmatrix}$$

5. **We decrypt these ciphertexts to the start of a second round with the guessed half of round key.**

   Note: We get first 8 bits of difference at the start of a second round.

   The result is shown in figure 4.9.

6. **We compare the first half of a difference of the decrypted ciphertexts with first-halves of the pre-calculated set of *impossible differences*.**

$$\Delta = \begin{pmatrix} D & 0 \\ 5 & 0 \end{pmatrix} \notin \textit{possible differences}$$

$$\Downarrow$$

We have an *impossible difference*.

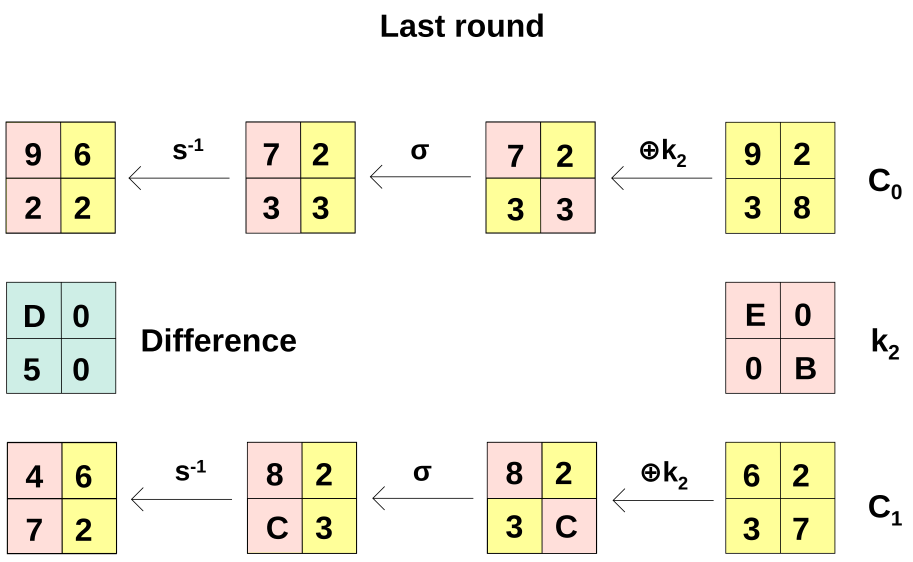

Figure 4.9: Attack on 2 rounds of Baby Rijndael – decrypting last round (2nd variant).

7. **If we get a match, we drop this guessed key and every other key with the matching 1st position and 4th position.**

   Note: In our case it means that we drop $2^8 = 256$ keys.

   All the dropped keys have this form (Guessed 1st position, Anything, Anything, Guessed 4th position).

   Reason: if we would consider every key in the mentioned form, we would get the same *half of difference* at the start of a second round. A change would only occur in the second-half. This means we can surely drop that key too, because a decryption with it results in an *impossible difference*.

$$\text{We drop the set of keys with this form } \begin{pmatrix} E & 0-F \\ 0-F & B \end{pmatrix}.$$

8. **We return to the step 4 and guess half of another round key (from remaining keys).**

   Note: We need to go through just $2^8 = 256$ keys instead of all $2^{16} = 65536$ needed in the previous variant of the attack.

We go through the remaining possible halves of keys. We end up with 3072 possible keys[10] left.

9. **Now we change plaintexts and go through the attack from step 2 again.**

   To end up with 256 keys, we need to go through **2.5** plaintext pairs on average[11].

10. **Notice the form of the result. We use it in the following attack for further improvement.**

    The possible keys take the form: $\begin{pmatrix} 1 & 0-F \\ 0-F & 6 \end{pmatrix}$. Notice that 1st and 4th position of the second round key are resolved.

[10]Number 3072 is specific to this particular example.

[11]We ran the program 100 times and calculated the average number.

4.10 Attack on 4 rounds of Baby Rijndael

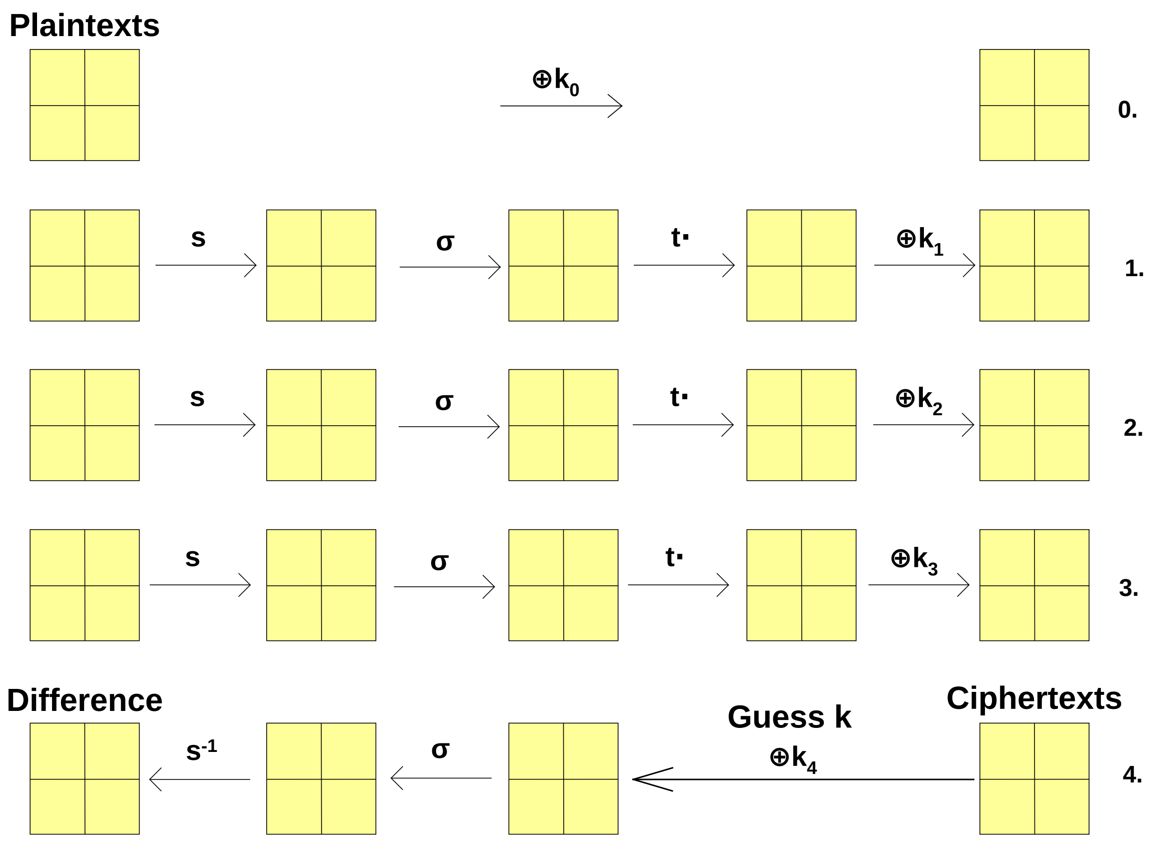

Figure 4.10: Attack on 4 rounds of Baby Rijndael.

In the following attack we always use these round keys:

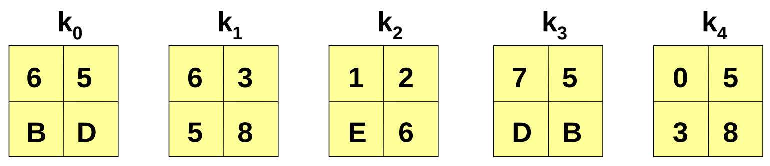
k0
6 5
B D
k1
6 3
5 8
k2
1 2
E 6
k3
7 5
D B
k4
0 5
3 8

Figure 4.11: Attack on 4 rounds of Baby Rijndael – key schedule.

We attained some previous knowledge about analysis of Baby Rijndael from two attacks described in previous subsections. We use them to construct an attack on all 4 rounds.

Attack:

1. We calculate the *impossible differentials* for the given *input difference*.

$$\text{We choose } \Delta_{in} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

   The result of calculation of the *(im)possible differentials* with this *input difference* is in `src/results/1000.txt`. The result was calculated using the algorithm described in section 4.7.

2. We choose a pair of plaintexts with the given *input difference*.

   Our choice:

$$P_0 = \begin{pmatrix} F & 1 \\ C & E \end{pmatrix} \qquad P_1 = \begin{pmatrix} E & 1 \\ C & E \end{pmatrix}$$

3. We run them through 4 rounds of Baby Rijndael. Remember that the fourth round is without `MixColumns`. We receive a pair of corresponding ciphertexts.

$$C_0 = \begin{pmatrix} F & 1 \\ 1 & 8 \end{pmatrix} \qquad C_1 = \begin{pmatrix} 3 & 1 \\ F & 4 \end{pmatrix}$$

4. **We guess a half of the fourth round key.**

   Our choice:

$$k_2 = \begin{pmatrix} E & 0 \\ 0 & B \end{pmatrix}$$

5. **We decrypt these ciphertexts to the start of a fourth round with the guessed half of the round key.**

   Result is shown in figure 4.12.

6. **We compare the first half of the difference of the decrypted ciphertexts with first-halves of the pre-calculated set of *impossible differences.***

   Note: For the purpose of comparing only first-halves we made a special file with half-sized *differences* and no duplicate values. The file is in `src/results/1000-halves.txt`.

$$\Delta = \begin{pmatrix} 2 & 0 \\ 9 & 0 \end{pmatrix} \quad \in \quad \textit{possible differences}$$
$$\Downarrow$$

We have a *possible difference.*

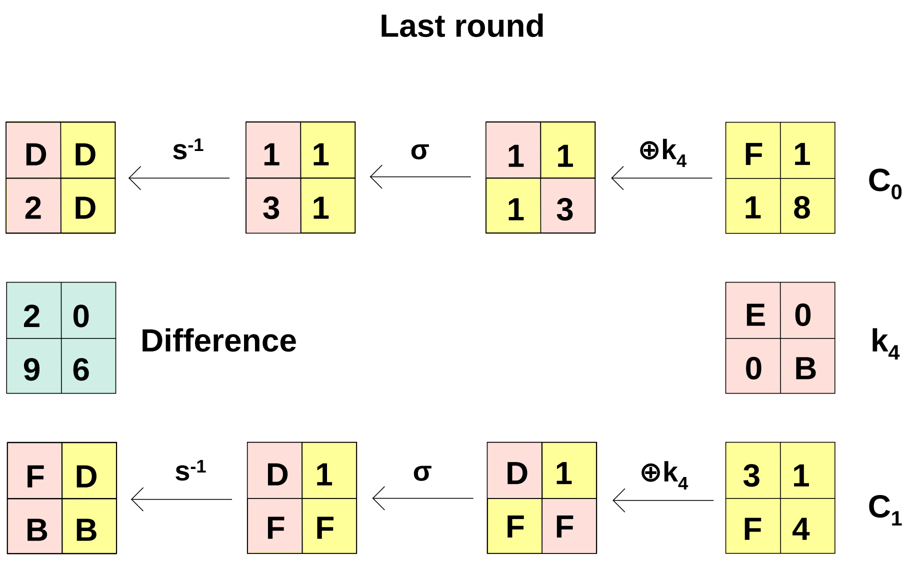

Figure 4.12: Attack on 4 rounds of Baby Rijndael – decrypting last round.

7. **If we get an impossible difference, we drop this guessed key and every other key with the matching 1st position and 4th position.**

   We got possible difference. So we do not cross anything.

8. **We return to step 4 and guess half of another round key (from the remaining keys).**

   We go through all remaining possible halves of keys. We end up with 55296 keys[12] left. We dropped 10240 ($40 * 2^8$) keys by checking just 256 key halves.

9. **Now we change plaintexts and go through attack from step 2 again.**

   To end up with 256 keys we need to go through $2^{5.7} = \mathbf{51}$ plaintext pairs on average[13].

   The possible keys take this form: $\begin{pmatrix} 0 & 0-F \\ 0-F & 8 \end{pmatrix}$

10. **Eventually we end up with solved 1st and 4th positions.**

[12] Number 55296 is specific to this particular example.
[13] We ran the program 100 times and calculated the average number.

11. **Now we have two possible options. Either perform the *impossible differential cryptanalysis* on the remaining 256 possible keys with full *differentials*** (a) **or apply brute-force attack on the rest** (b)**. We evaluate both options.**

    a) **We go through all the remaining 256 keys with full *differences* (*differentials*). This means we use method from 1st variant of an attack on 2 rounds (section 4.3).**

    Note: We use the original file mentioned in step 1 (`src/results/1000.txt`). We guess all bits of the fourth round key.

    To end up with **one key** we need to go through $2^{5.81} = \mathbf{56}$ plaintext pairs on average[14].

    The one possible key left is: $\begin{pmatrix} 0 & 5 \\ 3 & 8 \end{pmatrix}$ which matches $\mathbf{k_4}$. This means we successfully performed an attack on all 4 rounds of Baby Rijndael and recovered the correct key.

    b) **We go through all 256 left possible keys and decrypt ciphertexts to the start of a fourth round (= the end of a third round). Then we check, if we get a match of texts after three rounds.**

    Note: We are able to do so with just one pair of plaintexts.

    After going through $2^7 = 128$ possibilities in the average case we end up with the correct fourth round key $\begin{pmatrix} 0 & 5 \\ 3 & 8 \end{pmatrix}$.

[14]We ran program 100 times and calculated the average number.

Conclusion

In our thesis we described the important parts which, combined, provide view on impossible differential cryptanalysis. From the beginning where we outlined cipher on which this technique was implemented, through a general look at cryptanalysis and comparison of other techniques to build foundations on which the impossible differential cryptanalysis is constructed. After this theoretical background we moved to a practical example, because it is better to see something even once in reality than hear hundred times about it (in this case: read about it). All the theoretical parts came together so that we could perform a proper analysis and devise an algorithm. Finally we were able to execute complete attack on all 4 rounds of Baby Rijndael. In the last chapter (4.10) we provided what we set out to do in the introduction – a tutorial on how to perform a basic attack with this technique.

Once again we saw, that is better to look at things from various different perspectives. What the original differential cryptanalysis started (and did well), the impossible differential cryptanalysis took and applied from another point – from the other side. We looked at the cipher and its impossibilities which gave us much more information than we thought at the start. **At the end we performed a successful attack on all 4 rounds of Baby Rijndael with:**

1. **Time complexity** equal to $2^{5.7} * 2^8 + 2^{5.81} * 2^8 =$ **27671 units** and **space complexity** equal to $2^{16} =$ **65536 units** with type ***a*** of the attack – two consecutive impossible differential attacks (one on halves of keys and second one on remaining keys).

2. **Time complexity** equal to $2^{5.7} * 2^8 + 2^7$ units = **13436 units** and **space complexity** equal to $2^8 =$ **256 units** with type ***b*** of the attack – an impossible differential cryptanalysis followed by an exhaustive search of the remaining possibilities.

It is apparent that time needed for this type of attack is significantly smaller than for the brute-force attack where we have to go through $2^{15} = 32768$ possibilities in the average case. For that reason we are able to conclude that this type of cryptanalysis is effective. Because of that, we think it can be beneficial to study this technique in more detail and show its workings to students. It can at least stimulate other points of view on analysis and the design of ciphers. After all, what we want for the future is a safe environment and that is not possible without safe communication.

Bibliography

[1] Kokeš, J. *Kryptoanalýza šifry Baby Rijndael.* Master's thesis, ČVUT, Prague, 2013.

[2] Jureček, M. Pokročilá kryptologie: Algebraická kryptoanalýza. ČVUT FIT, 2013. Available from: `https://edux.fit.cvut.cz/courses/MI-KRY/_media/lectures/06/ac.pdf`

[3] Heys, H. M. A Tutorial on Linear and Differential Cryptanalysis. *Cryptologia*, volume 26, no. 3, July 2002: pp. 189–221, ISSN 0161-1194, doi:10.1080/0161-110291890885. Available from: `http://dx.doi.org/10.1080/0161-110291890885`

[4] Bergman, C. *A Description of Baby Rijndael.* Iowa State University, 2005. Available from: `http://orion.math.iastate.edu/cbergman/crypto/homework/babyr/babyr.pdf`

[5] Wrolstad, J. *A differential cryptanalysis of Baby Rijndael.* Iowa State University, 2009. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.155.1547&rep=rep1&type=pdf`

[6] Kocher, P. C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, London, UK, UK: Springer-Verlag, 1996, ISBN 3-540-61512-1, pp. 104–113. Available from: `http://dl.acm.org/citation.cfm?id=646761.706156`

[7] Lórencz, R. Pokročilá kryptologie: Lineární kryptoanalýza. ČVUT FIT, 2011. Available from: `https://edux.fit.cvut.cz/courses/MI-KRY/_media/lectures/04/prednaska4.pdf`

[8] Bard, G. V. *Algebraic Cryptanalysis.* Springer Publishing Company, Incorporated, first edition, 2009, ISBN 0387887563, 9780387887562.

[9] Biham, E.; Biryukov, A.; et al. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, Berlin, Heidelberg: Springer-Verlag, 1999, ISBN 3-540-65889-0, pp. 12–23.

[10] Biham, E.; Biryukov, A.; et al. Miss in the Middle Attacks on IDEA and Khufu. In *Proceedings of the 6th International Workshop on Fast Software Encryption*, FSE '99, London, UK, UK: Springer-Verlag, 1999, ISBN 3-540-66226-X, pp. 124–138.

[11] Biham, E. Cryptanalysis of reduced variants of Rijndael. In *3rd AES Conference*, New York, USA, 2000.

[12] Cheon, J. H.; Kim, M.; et al. Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. In *Proceedings of the 4th International Conference Seoul on Information Security and Cryptology*, ICISC '01, London, UK, UK: Springer-Verlag, 2002, ISBN 3-540-43319-8, pp. 39–49.

[13] Kim, J.; Hong, S.; et al. Impossible Differential Cryptanalysis for Block Cipher Structures. In *Progress in Cryptology - INDOCRYPT 2003: 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003. Proceedings*, edited by T. Johansson; S. Maitra, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ISBN 978-3-540-24582-7, pp. 82–96, doi:10.1007/978-3-540-24582-7_6.

[14] Phan, R. C.-W. Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES). *Inf. Process. Lett.*, volume 91, no. 1, July 2004: pp. 33–38, ISSN 0020-0190, doi:10.1016/j.ipl.2004.02.018.

[15] Mala, H.; Dakhilalian, M.; et al. Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, edited by G. Gong; K. C. Gupta, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-17401-8, pp. 282–291, doi:10.1007/978-3-642-17401-8_20.

Appendix **A**

Contents of CD attachment

- `readme.txt` .... short description of CD contents
- `exe` .... folder with the executable files of the implementation
  - `babyRijndael-original.exe` .... the executable file of the implementation of the Baby Rijndael cipher
  - `impdiff.exe` the executable file of the impossible differential cryptanalysis on 4 rounds of Baby Rijndael (type *b* of the attack)
- `src`
  - `impl` .... source codes of the implementation
  - `results` .... files containing the results
    - `1000.txt` .... file containing the impossible differentials for input difference 1000
    - `1000-halves.txt` .. file containing the first halves of impossible differentials for input difference 1000
    - `mixDiffs.txt` .... file containing the results of analysis of `MixColumns`
    - `sbox-diffs.txt` .... file containing the results of analysis of `SubBytes`
    - `sMixDiffsValues.txt` .... file containing the results of analysis of `SubBytes+MixColumns`
  - `thesis` .... source form of the thesis in LaTeX
- `text` .... thesis text
  - `thesis.pdf` .... thesis text in pdf format