Analyzing MILPs for Linear and Differential Cryptanalysis

Aura Sofia Lohr

December 2022
Bachelor Thesis

Prof. Dr. Holger Dell
Theoretical Computer Science Group
Department of Computer Science
Goethe University Frankfurt

Advised by
Leo Wennmann

Abstract

The security of encryption algorithms is more important than ever because of their ubiquity in critical infrastructure like banking, defense, energy, and public administration. Two widely known and significant types of attacks on encryption are linear and differential cryptanalysis. When designing a cipher, i.e. an algorithm for encryption, resistance against these attacks is crucial. In order to evaluate if a cipher is secure and will resist these attacks, the security bound, which is based on the minimum number of active s-boxes in a cipher, has to be determined. Mouha et al. [29] present a new method to calculate this bound using mixed integer linear programs (MILPs) for the ciphers AES and Enocoro-128v2. To calculate these MILPs, Mouha et al. use an MILP solver. The goal of this thesis is to analyze these MILPs and to search for inherent structures that can be potentially solved more efficiently than with an MILP solver. As a matter of fact, new structures were found with a Python framework that generates the constraint matrix and the related visualizations. The analysis revealed that the linear and differential cryptanalysis MILP for the cipher AES can be described with the 4-block structure. This is the first real-world occurrence of the 4-block which is a widely researched theoretical structure. As for the Enocoro-128v2 MILP, a new structure, called stair-fold, has been identified. These findings could help in accelerating the time spent searching for the minimum number of active s-boxes in a cipher.

Contents

1 Introduction

Contrary to the meaning of a crypt, this thesis has nothing to do with the analysis of underground chambers. Instead, the 'crypt' in cryptanalysis comes from 'crypto', derived from the Greek word *kryptós* which means 'hidden, secret'. Cryptanalysis belongs to the field of cryptology which examines encryption and decryption of information. Cryptology can be organized into two directions: cryptography and cryptanalysis. Cryptography studies the development of encipherment systems that are secure against manipulation and unwanted readers. The field of cryptanalysis on the other hand attempts to break those encryptions and gain access to the messages. But to this day, no one knows where the 'o' in cryptanalysis went.

Even though the goals of cryptography and cryptanalysis appear to be complete opposites, the fields depend on each other: For example, cryptanalysts help identify problems in encryption algorithms made by cryptographers. These algorithms can then be built stronger than before. This thesis looks at the attacks of differential and linear cryptanalysis and how to prove resistance against them. Proving resistance against cryptanalysis is often done by calculating the minimum number of active s-boxes in a cipher. With this number, one can find out the number of rounds required to be resistant. Efficiently calculating a bound for the minimum number of active s-boxes is a central research question in the field of linear and differential cryptanalysis. Before Mouha et al. [29] found an approach using MILPs, different methods were already established. The 'wide trail design strategy' was used by Daemen and Rijmen [16] to prove a security bound for the block cipher AES. With other methods that involve a lot of manual work and programming, a minimum number of s-boxes or a differential characteristic (which is used for the minimum number of active s-boxes) was calculated in several papers for other ciphers like feistel structures [6, 7, 24, 33, 8]. Some of these works already use integer linear programs (ILPs) to calculate the bound. Bogdanov [6, 7] also presented an approach using MILPs, though the method requires solving many ILPs alongside and therefore is not as efficient as the method of Mouha et al. published in 2011. Mouha et al. introduce new variables in order to capture the problem in one big MILP, thereby fully automating the calculations. This approach accelerates the time spent finding these bounds. The generated MILP is then solved by putting it into an MILP solver. However, from an optimization perspective one could examine these MILPs and see if they can be solved more easily by considering their inherent structure, since there are algorithms to solve certain structures efficiently. This thesis analyzes the MILPs that Mouha et al. built for differential and linear cryptanalysis with the ciphers AES and Enocoro-128v2 from an optimization point of view. To understand the use of MILPs for cryptanalysis we first define MILPs and their structures in Section 1.1. Since we will look at cryptanalysis for the ciphers AES and Enocoro-128v2, a description of their working is given in Section 1.2. Then, the attacks of linear and differential cryptanalysis are explained with the help of examples in Chapter 2. With this information, we can finally make the connection of how MILPs can analyze the security of a cipher against the attacks of linear and differential cryptanalysis in Chapter 3. After that, a Python framework to generate the MILP for a given cipher and analyze the structure of the constraint matrix is presented in Section 4.1. With the help of this framework, new matrix structures for the constraint matrix were found, inter alia, the first real-world example of the 4-block structure. The new-found structures are then depicted in detail in Section 4.2 and 4.3.

1.1 Mixed Integer Linear Programs

In the field of optimization, linear programs are commonly used to find an optimal solution to a problem. They consist of an objective function that has to be minimized or maximized over given variables and constraints that restrict the values of the variables. Problems where every variable in the solution vector can take fractional values are called linear programs (LPs). In contrast to LPs, the solution to an ILP can take only integer values. Normally, for the analysis of attacks ILPs were used. But, since MILPs have a better practical application in real-world, the ILPs are later transformed into MILPs. The objective function of an MILP contains integer variables but also variables that can take fractional values.

▶ Definition 1.1 (MILP). *Consider the matrices $M_1 \in \mathbb{Z}^{m \times n_1}$, $M_2 \in \mathbb{Z}^{m \times n_2}$ and the vectors $c \in \mathbb{Z}^{n_1}$, $d \in \mathbb{Z}^{n_2}$, $b \in \mathbb{R}^m$. A MILP consisting of an objective function and constraints has the form*

$$\begin{aligned} &max\ c^T x + d^T y \\ &(M_1\ M_2) \begin{pmatrix} x \\ y \end{pmatrix} = b \end{aligned}$$

where $x \in \mathbb{Z}^{n_1}_{\geq 0}$ and $y \in \mathbb{R}^{n_2}_{\geq 0}$.

Regarding complexity, LPs can be solved efficiently in polynomial time. ILPs on the other hand are NP-complete and not believed to be generally solvable in polynomial time. When comparing an MILP and an ILP with the same size, the MILP can be solved more efficiently since it contains less integer variables, but not efficient like LPs. MILPs exist naturally in many different occasions. One example is agricultural production planning, where some parameters can take only integer values (labor) and others have no integral restrictions (land, fertilizer). In practice, the solution of an MILP can be found with a solver like Gurobi [1], which was used by Mouha et al. as well. Gurobi finds the optimal solution for arbitrary MILPs without considering the constraint matrix structure. But, analyzing the constraint matrix structure might have an advantage: Depending on the structure, more efficient algorithms can be used. Through permutation of the rows and columns, a constraint matrix can take the form of different structures. One of these structures is the 4-block structure, which consists of four types of blocks.

▶ Definition 1.2 (4-block). *Consider the matrices $A \in \mathbb{Z}^{h_A \times w_A}, B_1, ..., B_r \in \mathbb{Z}^{h_B \times w_B}$, $C_1, ..., C_r \in \mathbb{Z}^{h_C \times w_C}$ and $D_1, ..., D_r \in \mathbb{Z}^{h_D \times w_D}$ with $h_A = h_B$, $w_A = w_C$, $w_D = w_B$ and $h_D = h_C$. An MILP is called a 4-block if the constraint matrix has the following structure*

$$\begin{pmatrix} A & B_1 & \cdots & B_r \\ C_1 & D_1 & & \\ \vdots & & \ddots & \\ C_r & & & D_r \end{pmatrix} \begin{pmatrix} x \\ y_1 \\ \vdots \\ y_r \end{pmatrix} \geq \mathbf{0},$$

where mixed solution vector consists of $x \in \mathbb{Z}^{w_A}_{\geq 0}$, $y_1 \in \mathbb{Z}^{w_B}_{\geq 0}$, $y_i \in \mathbb{R}^{w_B}_{\geq 0}\ \forall i \in 2, ..., r$ and $\mathbf{0}$ denotes an all-zero vector.

From now on, if an entry in a matrix is empty, it corresponds to a zero. The symbol $\mathbf{0}^{m \times n}$ denotes an all-zero matrix with the dimensions $m \times n$. In the 4-block structure, block A has the same height as all of the B-blocks and the same width as all of the C-blocks. Each D-block has the same width as the B-blocks and the height of the C-blocks. The D-blocks form a diagonal in the matrix and are independent from each other. Since the 4-block describes the constraints of an MILP, it contains integer

[1] https://www.gurobi.com/

and fractional values. The variables corresponding to the A-block and first B-block B_1 have to take integer values and the remaining variables can take fractional values. The 4-block is a combination of two well-researched block structures, namely n-fold and two-stage [13, 19, 12, 26, 22, 25, 23]. While those two structures admit an 'efficient' algorithm exploiting their structure, it is unknown whether there exists one for 4-block. Section 4.3 contains the first real-world 4-block example.

Further, we newly introduce the stair-fold structure, partly inspired by the n-fold structure. As opposed to an n-fold, the blocks on the diagonal are not independent.

▶ Definition 1.3 (Stair-fold). *Consider the matrices $C_1 \in \mathbb{Z}^{h_C \times w_{C1}}$, $C_2 \in \mathbb{Z}^{h_C \times w_{C2}}$, $A_1, ..., A_r \in \mathbb{Z}^{h_A \times w_A}$ and $B_i \in \mathbb{Z}^{h_B \times w_i}$ for $i = \{1, ..., q\}$ and $B_i \in \mathbb{Z}^{h_B \times w_B}$ for $i = \{q+1, ..., r\}$ with $h_A = h_B$. An MILP is called a stair-block if the constraint matrix has the following structure*

$$\begin{pmatrix} C_1 & C_2 & & & & \\ A_1 & B_1 & & & & \\ A_2 & B_2 & & & & \\ \ddots & \vdots & & & & \\ & B_{q-1} & & & & \\ & B_q & B_{q+1} & & & \\ & & & B_{q+2} & & \\ & & & & \ddots & \\ A_r & & & & & B_r \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \geq \mathbf{0},$$

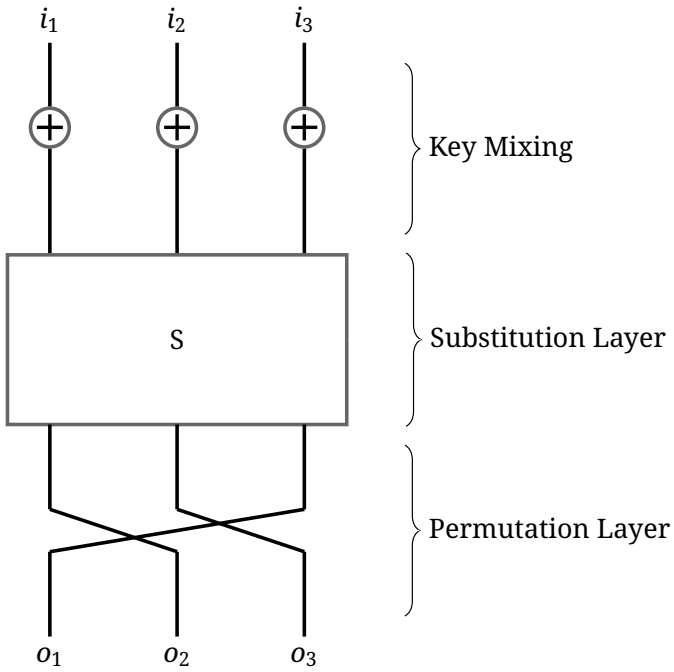
where the mixed solution vector consists of $x \in \mathbb{Z}^{w_{C1}+r \cdot w_A}$, $y \in \mathbb{Z}^j$ and $z \in \mathbb{R}^{w_{C2}-j}$.

The first j variables corresponding to the C_2 block have to take integer values as well as the variables corresponding to the A-blocks and C_1 block. The rest of the variables is free.

1.2 Symmetric Ciphers

The word cipher derives from the Arabic word for zero *sifr* which turned into the medieval Latin word *cifra*, when the Arabic numerical system was adapted in Europe in the Middle Ages. Europeans at the time used the roman number system where the number zero did simply not exist. The concept of zero was therefore confusing and mystical to Europeans, and the term was then used to refer to a secret message [17]. In fact, our modern ciphers do exactly this: they transform easily understood messages into hard to unravel code. A cipher is an algorithm used to encrypt or decrypt a message. It takes a key and the plaintext as an input, and returns the ciphertext, which (hopefully) makes no sense by reading it. This cryptic message can then be decrypted with a key in order to get the original message, the plaintext. Ciphers can be asymmetric or symmetric. In an asymmetric cipher like the RSA-Cryptosystem, two separate keys are needed for the encryption and decryption. Symmetric ciphers use the same key for both the encryption and decryption of a message. In order to extract the message from the ciphertext, one uses the same key and the reverse cipher, and gets the original plaintext. Since linear and differential cryptanalysis are attacks against symmetric ciphers, the thesis will focus on them.

For symmetric ciphers, there are different methods of encrypting messages. Stream ciphers encrypt the message one-by-one for each digit in the plaintext. Enocoro-128v2, for example, is a stream cipher. In block ciphers, the input message is computed through the operations with a fixed number of bits, a block. A significant type of block cipher is the substitution-permutation-network (SPN), which has substitution and permutation layers. SPNs are the basis for AES and other ciphers. Since



■ Figure 1.1. Substitution Permutation Network

linear and differential cryptanalysis will be explained with an SPN and they are the basis for AES, a description of them follows. In an SPN, the input bits are first XOR-ed with the key in the key-mixing step. The resulting bits go through the substitution layer, which often is an s-box. The s-box is a bijective nonlinear mapping of the bits. The output of the s-box is then permutated in the permutation layer. This describes one round. The output of the current round will be the input for the next round. A cipher always has a specified number of rounds where the operations are repeated. This ensures a stronger encryption of the message since for every round the message will diffuse more and more. Figure 1.1 shows one round of a simple, generic SPN, while real-world applications usually have more than three inputs and one s-box per round.

1.2.1 AES

In 1997, the National Institute of Standards and Technology of the United States (NIST) announced that a new encryption standard was needed [30]. Fifteen different ciphers were submitted and the cipher Rijndael was chosen to be the new advanced encryption standard with a few alterations [31]. Rijndael, which is now called the advanced encryption standard (AES), is a block cipher designed in 1998 by Joan Daemen and Vincent Rijmen. Today, AES is still being used, and it is the only publicly available cipher that has been approved to transfer top secret information by the U.S. national security agency (NSA) [34].

AES has a block size of 128 bits for the message, and a key size of 128, 192, or 256 bits. Depending on this key size, there are 10, 12, or 14 rounds. 10 rounds for 128-bit key size, 12 for 192-bit key, and 14 for 256-bit key. The more rounds a cipher uses, the safer it is against attacks. Every operation in the cipher is calculated on a 4×4 array of 16 bytes called the state array and in the first round the state is filled with the input, the plaintext. The operations are all computed over the finite field $GF(2^8)$. AES is such a significant cipher because with just a few operations the message will be completely

convoluted. That is why we explain every operation separately. In the beginning, the round key is added to the state bytes with *AddRoundKey()*. Then, every round except for the last one consists of the same operations:

$$\underbrace{\textit{SubBytes()}}_{\textit{Substitution}} \rightarrow \underbrace{\textit{ShiftRows()} \rightarrow \textit{MixColumns()}}_{\textit{Permutation}} \rightarrow \textit{AddRoundKey()}$$

In the last round, the *MixColumns()* step is not executed. The output of one operation is the input for the next operation. The cipher is based on an SPN as depicted in Figure 1.1, thus there is a substitution and permutation layer in every round, which both contribute to the diffusion of the message. In the following, each operation will be explained in more detail with a corresponding visualization in Figure 1.2, where the left (right) side depicts the state array before (after) the operation.

SubBytes()

Every state byte b_i goes through the s-box and is replaced by $S(b_i)$. The s-box is a 8×8 bit non-linear mapping. The corresponding byte $S(b_i)$ to the state byte b_i can be found in the lookup-table for the s-box. As explained later, s-boxes are a crucial step to linear and differential cryptanalysis. This is because they represent the only non-linear function. Without the s-boxes, the cipher would be much less secure.

ShiftRows()

Each row of the state array is now shifted cyclically to the left. The first row stays the same, the second row shifts one to the left, the third two to the left and the last row three to the left. This assures that after the operation each column contains bytes that were previously in every other column, avoiding an independent encryption of every column in the next step. This would mean that AES consists of four smaller independent ciphers, which is not the goal. The shift is emphasized by the colored columns in Figure 1.2.

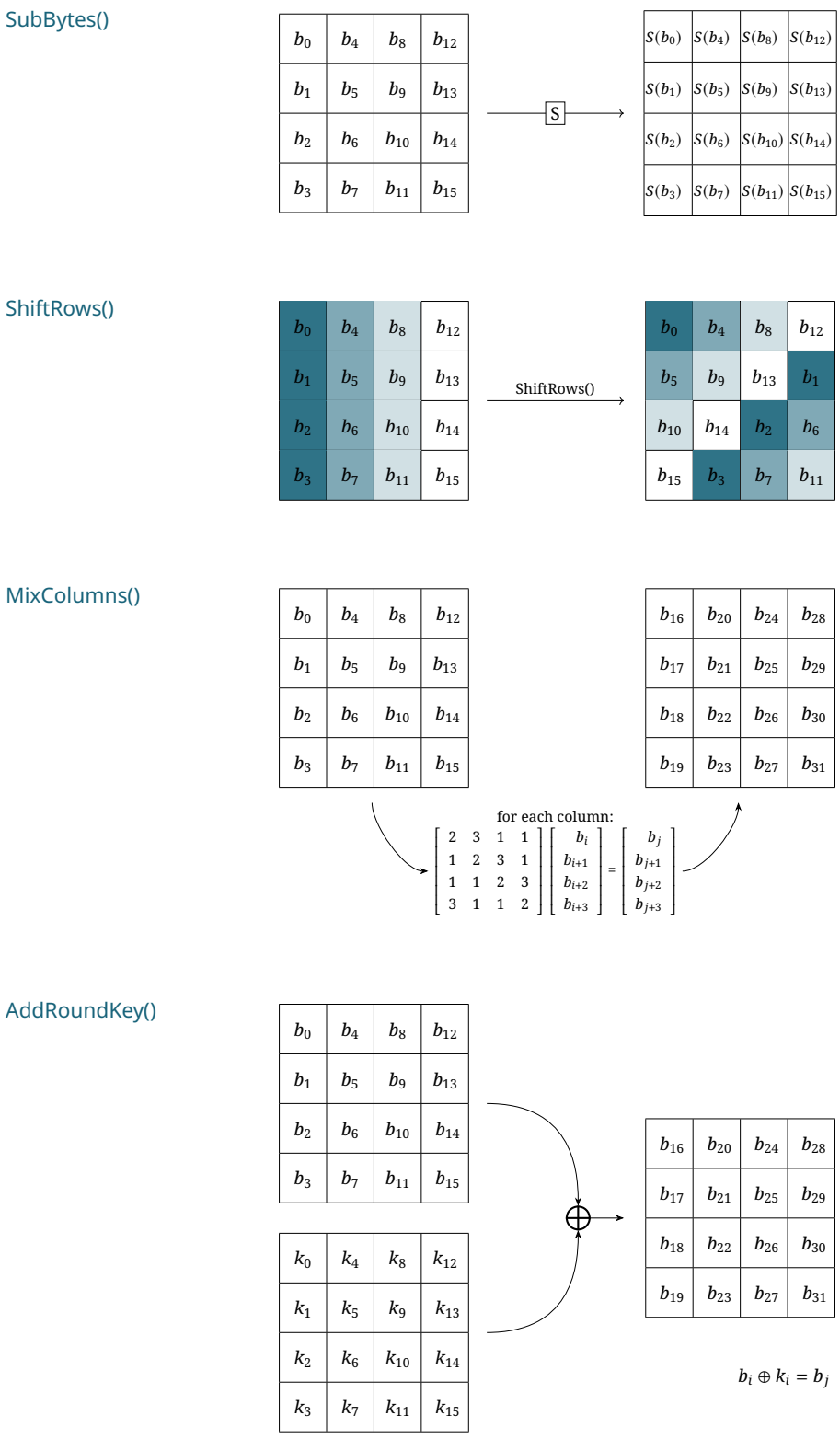
MixColumns()

Every column is multiplied with a fixed matrix. This linear function results in a new column where every output byte is influenced by all four input bytes. Together with *ShiftRows()*, *MixColumns()* is mainly responsible for the diffusion in the cipher. The step can be described as the multiplication with the MDS matrix below in the finite field $GF(2^8)$.

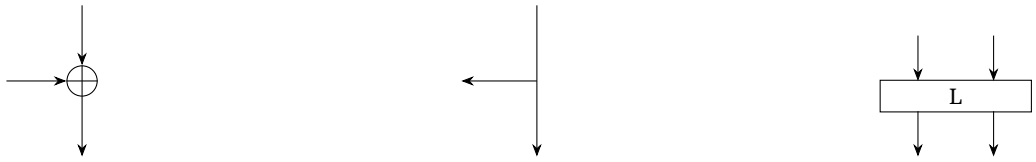
AddRoundKey()

With the help of a key schedule, the key is expanded into many round keys. In this step, the state array is combined with the corresponding round key through a XOR operation. Since the round key has the same size as the state array, every byte from the state array is XOR-ed with a byte from the round key.

When analyzing an attack on the cipher, we do not consider how exactly these operations work, but only how they change the state array. Therefore, an example of an encryption would not contribute to the understanding of the following chapters and is out of the scope of this thesis.



■ Figure 1.2. Visualization for the operations in the cipher AES [14]. Each operation is executed once in a round. The input of the operations is always the state array that was the output of the last operation.



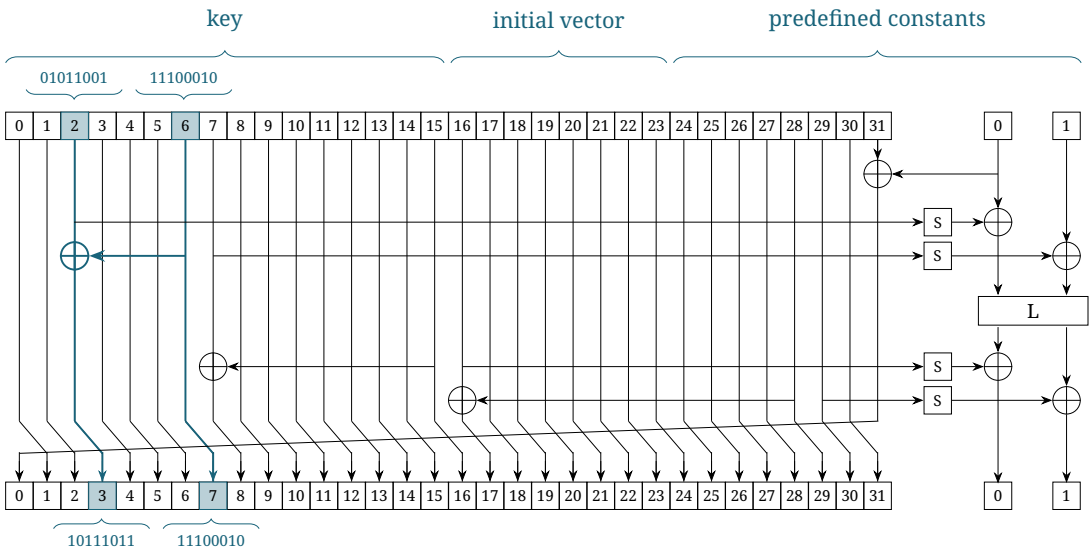
a. XOR operation b. Three forked branch c. Linear function

■ Figure 1.3. Operations for Enocoro-128v2. The XOR operation takes two bytes as an input and outputs the byte that results from XOR-ing the input. The three forked branch takes one input and duplicates it, thus the two output bytes correspond to the input byte. The linear function is a linear transformation of two input bytes multiplied with a 2×2 matrix over $GF(2^8)$.

1.2.2 Enocoro-128v2

The cipher Enocoro-128v2 was developed by the Japanese company Hitachi [21]. Published in 2010, it was a contender in the process for CRYPTREC, a Japanese government project similar to the search for the new Advanced Encryption Standard in the USA. Although it was not selected, the cipher, which is a pseudorandom number generator, is considered safe in terms of security [32, 18]. Enocoro-128v2 is a stream cipher, so the input bytes are encrypted one at a time. The following is a short description of the cipher.

As an input, the cipher takes a 128 bit key and 64 bit initial vector. The state (similar to the state array in AES) consists of buffer b with 32 bytes and buffer a with two bytes. The first 16 bytes of the buffer b consist of the key, and the following eight bytes consist of the initial vector. The rest of buffer b and buffer a is filled with predefined constants. The number of rounds is set to 96 for Enocoro-128v2. Though as we will see later on, it is resistant against certain attacks with fewer rounds. In one round, there are eight XOR operations with bytes from the state. A XOR operation can be seen in Figure 1.3a. Corresponding to these, there are eight three forked branches. A three forked branch, depicted in Figure 1.3b, takes a byte as an input and then outputs the same byte two times so that it can be used in two new operations. The linear function, which consists of multiplying a predefined matrix with the input over $GF(2^8)$, is executed one time. It has two inputs and two outputs and can be seen in Figure 1.3c. How the multiplication in the linear transformation exactly works is not relevant for this thesis, as we will see later on. In total, four bytes go through the 8×8 s-box per round. At the end, the bytes from the buffer are shifted to the right. In Figure 1.4 the exact order and use of those operations in one round is described. In the same figure, a small example of two bytes using a three forked branch and a XOR operation can be seen.



■ Figure 1.4. Visualization of one round for Enocoro-128v2 [21]. For the first round, the state is filled in the following way: The first sixteen bytes consist of the key. The next eight bytes are our initial vector, thus our message to be encrypted. The remaining bytes are filled with predefined constraints. The bytes then go through the operations and four bytes go through the s-box. At the end of the round, we have again 34 bytes which will be the input for the next round. Example for two bytes: Assuming at the beginning of the round the byte b_2 is (01011001) and b_6 is (11100010). The three forked branch duplicates the byte b_6 and one output goes to the XOR operation and the other one is put into the byte at position 7. The bytes at position 2 and 6 are XOR-ed and put into the byte at position 3. After the round the byte b_3 is (10111011) and b_7 is (11100010).

2 Linear and Differential Cryptanalysis

Even though linear and differential cryptanalysis are often compared and analyzed together, they were discovered independently. The method of differential cryptanalysis was first published by Biham and Shamir [4] in 1990 where they analyzed the attack with different DES-like cryptosystems. DES was the predecessor of AES designed by the company IBM. Rightfully, Shamir and Biham [5] remarked that DES is not as weak against differential cryptanalysis as they thought it would be. In fact, a member of the original IBM DES team revealed in 1994 that DES was designed to withstand differential cryptanalysis, which was known to them since 1974. But, in accordance with the NSA, IBM decided to not publish the attack in order to secure the advantage of the USA in the field of cryptanalysis [10]. The discovery of linear cryptanalysis was more calm: Matsui and Yamagishi [28] published a paper in 1992 where they presented the technique. In 1993, Matsui [27] applied linear cryptanalysis to the DES cipher. In this paper, he also states the Piling-up Lemma which is still relevant and applicable today. The Piling-up Lemma describes the probability that a linear boolean function holds and it is used for linear cryptanalysis as we will see in Chapter 3. Since then, linear and differential cryptanalysis became two of the most significant attacks.

2.1 Linear Cryptanalysis

Linear cryptanalysis works by finding a linear approximation between the input and the output of the s-box. The s-box is the only non-linear operation in a cipher and the approximation tries to find a linearity within the s-box. It is a known-plaintext attack, the attacker has access to a set of plaintexts and their corresponding ciphertexts. The structure and working of the cipher are known. The attacker wants to prove that they are not in a random-world setting, and maybe even extract the key. Contrary to a random permutation, a cipher has a structure. If the attacker knows that the procedures are not random, the steps of the cipher can be reenacted and dependencies can be found.

Thus, the goal is to find a linear connection between the input and output of an s-box in order to prove that the ciphertext is not a random permutation. To describe a connection, this linear equation can be used for an $n \times n$ bit s-box:

$$x_1 \oplus x_2 \oplus x_3 \oplus ... \oplus x_n = y_1 \oplus y_2 \oplus y_3 \oplus ... \oplus y_n$$

where $X = (x_1, x_2, x_3, ..., x_n)$ represent the input bits of the s-box and $Y = (y_1, y_2, y_3, ..., y_n)$ the output bits of the s-box. The symbol $\oplus$ describes the exclusive or (XOR). But the equation describing the connection between every input bit and every output bit is not the only interesting part, every other combination of input and output bits can also give information to the attacker. For these combinations, so-called linear masks are used. Every equation can be described by a linear input mask Γx and a linear output mask Γy. For the input/output of an s-box, a mask indicates for each bit if it is contained in the equation or not. Considering a 3×3 bit s-box, the equation for the input mask $\Gamma x = (101)$ and output mask $\Gamma y = (100)$ looks like the following

$$1x_1 \oplus 0x_2 \oplus 1x_3 = 1y_1 \oplus 0y_2 \oplus 0y_3.$$

x_1	x_2	x_3	y_1	y_2	y_3	$x_1 \oplus x_3$	$x_1 \oplus x_3 = y_1$
0	0	0	0	1	0	0	True
0	0	1	1	0	1	1	True
0	1	0	0	1	1	0	True
0	1	1	0	0	1	1	False
1	0	0	1	1	1	1	True
1	0	1	0	0	0	0	True
1	1	0	1	0	0	1	True
1	1	1	1	1	0	0	False

a. Outcomes for linear masks $\Gamma x = (101)$ and $\Gamma y = (100)$

	0	1	2	3	4	5	6	7
0	+4	0	0	0	0	0	0	0
1	0	0	-2	+2	0	0	-2	-2
2	0	0	0	0	0	+4	0	0
3	0	0	-2	-2	0	0	+2	-2
4	0	-2	0	-2	+2	0	-2	0
5	0	+2	-2	0	**+2**	0	0	+2
6	0	+2	0	-2	-2	0	-2	0
7	0	+2	+2	0	+2	0	0	-2

b. Linear Approximation Table (LAT)

■ Figure 2.1. Tables for linear cryptanalysis. With the Table 2.1a the entry of the LAT in Table 2.1b in the fifth row and fourth columns is calculated.

We only look at the first and last input bit and the first output bit, thus

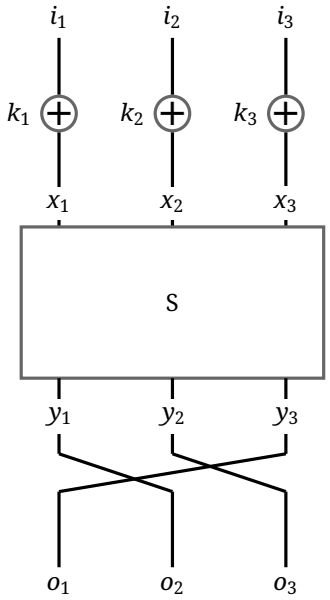
$$x_1 \oplus x_3 = y_1.$$

In the following example, we examine this equation and try to find out if there is a linearity. Consider this 3×3 bit s-box:

input (x_1, x_2, x_3)	000	001	010	011	100	101	110	111
output (y_1, y_2, y_3)	010	101	011	001	111	000	100	110

In a random world, the equation $x_1 \oplus x_3 = y_1$ should be true with probability $p = 1/2$. But applying the equation to all possible inputs of our s-box, we see in the table in Figure 2.1a that it holds true in 6 out of 8 cases. Thus, $p = 6/8$ and we know that we are not in a random world and can predict the working of the s-box. One could also find an equation that occurs with a lower probability than 1/2, since the goal is to have a probability that is as far away from 1/2 as possible. To describe this deviation from 1/2, the bias ϵ is used. The bias is defined as $\epsilon = p - 1/2$. The bias of the example above would be $6/8 - 1/2 = 2/8$. Since we go through many s-boxes during the attack, as explained in the next chapter, we want to know the probability of every possible equation. These probabilities/biases are stored in a linear approximation table (LAT). An LAT consists of 2^n columns and rows for a message of n bits. The rows describe the input linear mask, thus which input bits belong to the equation and the columns show the output linear mask, thus which output bits belong to the other side of the equation. The linear masks are depicted as decimal numbers. The LAT for our s-box can be found in Figure 2.1b. The equation from our example $x_1 \oplus x_3 = y_1$ will be at the fifth row and fourth column since the linear masks are $\Gamma x = (101)$ (5 in decimal notation) and $\Gamma y = (100)$ (4 in decimal notation). Every entry describes how often the related equation holds true in comparison to the expected number if we were in a random world. That means in our example we subtract the actual number of cases by 4, since if the equation holds true for 4 cases, the probability is $4/8 = 1/2$, thus random. Through dividing the entry with 2^n, one gets the bias of this linear mask combination. Thus, in the example above the entry is +2, because we have 4 + **2** cases in which the equation holds true. We can use this knowledge to derive the key. Consider the (not-so-secure) cipher in Figure 2.2 which consists of one round. The s-box is the same as in our previous example.

We have a pair of plain- and ciphertext as the input $i_1, i_2, i_3 = 1, 0, 1$ and the output $o_1, o_2, o_3 = 1, 1, 1$. The key is not known to us. This time, we look at the input mask $\Gamma x = (001)$ and output mask $\Gamma y = (011)$. Looking at the LAT (row 1 and column 3), we know that with a probability of (4+2)/8 =



■ Figure 2.2. Example Cipher where i_1, i_2, i_3 is the plaintext and o_1, o_2, o_3 is the output in ciphertext. Further, k_1, k_2, k_3 is the key and x_1, x_2, x_3 and y_1, y_2, y_3 denote the in- and output of the s-box.

3/4 the equation $0x_1 \oplus 0x_2 \oplus 1x_3 = 0y_1 \oplus 0y_2 \oplus 0y_3$ will hold. Thus, we look at

$$x_3 = y_2 \oplus y_3,$$

where x_3 is the combination of the key k_3 and the input i_3. Thus, $x_3 = i_3 \oplus k_3$. Substituting x_3 for $i_3 \oplus k_3$, we get

$$i_3 \oplus k_3 = y_2 \oplus y_3.$$

By going through the permutation layer, y_3 becomes o_1 and y_2 becomes o_3, so $y_3 = o_1$ and $y_2 = o_3$. We can replace them in our equation and get

$$i_3 \oplus k_3 = o_3 \oplus o_1.$$

Since we are in possession of the input and output values, we can insert them. Finally, k_3 is the only unknown variable,

$$1 \oplus k_3 = 1 \oplus 1.$$

Subsequently, we know that $k_3 = 1$ with a probability of 3/4. This is how the attacker can extract the key. In a more secure (and realistic) cipher, there is more than one round and thus finding the key is more difficult.

The probabilities for each combination of linear masks can also be described with the linear probability. The linear probability is already the deviation from $1/2$, thus it describes the bias. For our simple example, the linear probability for a pair of input and output masks can be found in the LAT by dividing the absolute entry by 2^n. To find the most significant probability of the cipher, we use the maximum linear probability (MLP).

► Definition 2.1 (MLP). *Consider an $n \times n$ bit s-box. The MLP describes its biggest bias*

$$MLP = max_{v,w \neq 0} \left\{ \frac{|LAT(v,w)|}{2^n} \right\},$$

where $LAT(v,w)$ denotes the entry of the LAT in row v and column w.

It is computed by finding the biggest absolute value in the LAT and dividing it by 2^n. It depicts the farthest one can be from the random world in terms of probability. For the attacker, a high MLP is optimal, while the cryptographer wants to have a small MLP for their cipher to keep it as close to the random world as possible. To describe an s-box or a function in a cipher the branch number [1] can be very useful. It helps in seeing how many of the in- and output values of the s-box or function have to be active in the minimum case. An in- or output value is active, if the corresponding value in the linear mask is set to one.

▶ Definition 2.2 (Linear branch number). *The linear branch number b_L of an s-box or function is defined as*

$$b_L = min_{v \neq 0, w, LAT(v,w) \neq 0}\{wt(v) + wt(w)\},$$

where LAT(v, w) is the entry of the LAT in row v and column w and $wt()$ describes the hamming weight, which is the sum of nonzero bits, e.g. $wt(5) = wt(101) = 2$.

In other words, the linear branch number is the minimum number of nonzero entries in the input and output linear masks of a function excluding the case where both masks are zero. Thus, if one bit in the input or output linear mask has the value one, at least b_L input and output bits from the linear masks need to be active. For our s-box the linear branch number would be $b_L = 2$, since in the LAT in Figure 2.1b the column for an all-zero output mask ($w = 0$) has only entries with the value zero, except for the all zero case $v = w = 0$ which is excluded by definition. That means the linear branch number for this s-box cannot be $b_L = 1$. But, as $LAT(1, 2) \neq 0$ and $wt(1) + wt(2) = wt(001) + wt(010) = 2$, and the number smaller than two was excluded, one can conclude that the linear branch number is $b_L = 2$. In the design process of an s-box, the branch number is a significant metric. Therefore, the branch number can be found in the specifications of the corresponding cipher, since it can be menial to calculate it manually.

2.2 Differential Cryptanalysis

Differential cryptanalysis has a similar structure to linear cryptanalysis, but the difference lies in the approximation of the s-box. It is a chosen plaintext attack, which means that the attacker has access to specific plain- and ciphertexts. This is important because while building a path through the cipher, we will look at the difference of two inputs in order to find an approximation. Consider the inputs $X = (x_1, x_2, ..., x_n)$ and $X' = (x'_1, x'_2, ..., x'_n)$ and outputs $Y = (y_1, y_2, ..., y_n)$ and $Y' = (y'_1, y'_2, ..., y'_n)$ of an s-box. The symbol δX denotes the difference between two s-box inputs X and X'. Their corresponding outputs are Y and Y', and their difference is δY. The difference is calculated with an exclusive or, thus

$$\delta X = (x_1 \oplus x'_1,\ x_2 \oplus x'_2, ..., x_n \oplus x'_n),$$

and δY is constructed in the same way. Now we look at the possible output differences δY and how they relate to δX. In a random world, a certain output difference δY to a certain δX should occur with a probability of $1/2^n$ when the message is consisting of n bits. The difference δY should be completely independent of δX. But, as we will see in the following example, this is not always true. Attackers use this to their advantage. Consider the same s-box as before:

input (x_1, x_2, x_3)	000	001	010	011	100	101	110	111
output (y_1, y_2, y_3)	010	101	011	001	111	000	100	110

In a random world when all input pairs have a difference of $\delta X = 101$, they should all have different output differences δY, since every combination has a probability of $1/8$ of occurring in the

X	X'	δX	Y	Y'	δY
000	101	101	010	000	010
001	100	101	101	111	010
010	111	101	011	110	101
011	110	101	001	100	101
100	001	101	111	101	010
101	000	101	000	010	010
110	011	101	100	001	101
111	010	101	110	011	101

a. All possible input pairs for $\delta X = 101$ and their results δY

	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	0	4	0	0	0	0	4
2	0	2	0	2	2	0	2	0
3	0	2	0	2	2	0	2	0
4	0	0	0	0	0	4	0	4
5	0	0	4	0	0	4	0	0
6	0	2	0	2	2	0	2	0
7	0	2	0	2	2	0	2	0

b. Difference Distribution Table (DDT)

■ Figure 2.3. Tables for differential cryptanalysis. With Table 2.3a the entries of the DDT in Table 2.3b for the fifth row are calculated.

random world. But when we look at all possible δY in Figure 2.3a for our s-box, we see that this is not true. In four cases, $\delta Y = 010$ and in the other four $\delta Y = 101$. This means, they each have a probability of 1/2 while the rest has a probability of 0. This is not optimal since it is not behaving randomly, which in turn tells the attacker that a cipher was used. Which δX leads to which δY is stored in a difference distribution table (DDT). For a specific row (δX) it shows how the δY are distributed, thus the sum of a row is always 2^n. The DDT for our example s-box is depicted in Figure 2.3b. We calculated the entries for the row five (since 5=101 in binary) in Figure 2.3a. Since we had four output differences that were 010 and the other four were 101, the entry is four in the columns of two (2=010) and five (5=101). The rest of the row has the value zero. By repeating this for every row (every input difference), the DDT is completed. The DDT is useful because the difference of the inputs becomes important. When we look again at the cipher in Figure 2.2, consider two in- and outputs ($I = (i_1, i_2, i_3)$, $I' = (i'_1, i'_2, i'_3)$ and $O = (o_1, o_2, o_3)$, $O' = (o'_1, o'_2, o'_3)$). The key remains the same $K = K'$. Now we look at the differences during the process of encryption. We have δI as the difference between the plaintexts. We know that X (and similarly X') is constructed through XOR-ing I and K:

$$(x_1, x_2, x_3) = (i_1 \oplus k_1, i_2 \oplus k_2, i_3 \oplus k_3).$$

The difference δX is calculated with:

$$\delta X = (x_1 \oplus x'_1, x_2 \oplus x'_2, x_3 \oplus x'_3).$$

Hence, the combined equations:

$$\delta X = (i_1 \oplus k_1 \oplus i'_1 \oplus k_1, i_2 \oplus k_2 \oplus i'_2 \oplus k_2, i_3 \oplus k_3 \oplus i'_3 \oplus k_3).$$

When XOR-ing the same values, the result will always be zero ($k_1 \oplus k_1 = 0$). Thus, we can eliminate them from the calculation:

$$\delta X = (i_1 \oplus i'_1, i_2 \oplus i'_2, i_3 \oplus i'_3).$$

It follows directly that

$$\delta X = \delta I.$$

So if we have $\delta I = 101$ it follows that $\delta X = 101$. From the DDT we see that with a probability of 1/2 the output of the s-box will be $\delta Y = 010$ and going through the permutation, $\delta O = 001$. Thus, if the attacker takes many input pairs with $\delta I = 101$ and approximately half of the output pairs have

$\delta O = 001$, they can distinguish the cipher from a random function and assume that we are in the real world.

In differential cryptanalysis, the key extraction process is a bit trickier than in linear cryptanalysis. It works by going through a differential path until the last round. Then, one tries every possible combination of keys and sees if it matches with the differential output of the last round. This process is repeated until the keys from every round are found as explained by Heys [20]. The probability that an input differential δX leads to a certain output differential δY is described as differential probability. For an s-box, the differential probability can be found in the DDT through dividing the entry by 2^n. The biggest differential probability is denoted by the maximum differential probability (MDP).

▶ Definition 2.3 (MDP). *Consider an $n \times n$ bit s-box. The MDP describes the biggest differential probability*

$$MDP = max_{v,w \neq 0} \left\{ \frac{|DDT(v,w)|}{2^n} \right\},$$

where $DDT(v,w)$ denotes the entry of the DDT in row v and column w.

Just like the MLP, the smaller the MDP is, the better is the security of the cipher. As the duality of linear and differential cryptanalysis demands, there exists a differential branch number b_D [1] similar to the linear branch number.

▶ Definition 2.4 (Differential branch number). *The differential branch number of a function or s-box f is defined as*

$$b_D = min_{v,v \neq w} \{wt(v \oplus w) + wt(f(v) \oplus f(w))\},$$

where $wt()$ denotes the hamming weight, in this context the sum of all nonzero bits.

Thus, the differential branch number is the smallest sum of the input and output differential bits which are one, except for the cases where the input and output differentials contain no differences. The differential branch number of our s-box is $b_D = 2$. This can be verified easily by looking at the DDT in Figure 2.3b. Again, the differential branch number is provided in the specifications of the cipher.

3 Obtaining Security Bounds using MILPs

In this chapter we answer the following questions: How can we use MILPs to calculate bounds? Why do we even need bounds? And how does this relate to linear and differential cryptanalysis? Well, when attacking with linear or differential cryptanalysis a path through the cipher is detected. The attacker or cryptanalyst knows that there is a high chance that the bytes in this path are dependent of each other. Every s-box contained in this path is considered an active s-box, since it plays a role in the attack. The example cipher in the previous chapter was very small in order to simplify the principle of the attacks, so the only s-box was obviously an active one. In a bigger cipher, a path through an SPN could look like Figure 3.1, where the active s-boxes are colored in blue. In linear cryptanalysis, an s-box is active if it is involved in a linear approximation. For differential cryptanalysis, the s-box has to have a nonzero input difference in order to be considered active. In both contexts, the approximations lead to a path through the cipher. The more active s-boxes are in a path, the harder it is for the attacker to use cryptanalysis. This is why ciphers are designed to have paths that involve many active s-boxes. Daemen and Rijndael designed AES with the 'wide trail design' [16] so that the paths in those attacks would be as wide as possible. Attackers, on the other hand, try to find a path that uses the minimum number of active s-boxes.

For each cipher there is a specific number of how many active s-boxes are needed so that linear or differential cryptanalysis become useless. In the following, we will call it the resistance bound. This number can be calculated with the MDP/MLP, as we will see in this chapter. If the number of active s-boxes is higher or equal to this resistance bound, the cipher is safe against the attack. Therefore, it is crucial to determine how many active s-boxes our cipher minimally contains. This is because the simple look at a cipher does not reveal how many s-boxes are active. In the following, we will refer to the minimum number of active s-boxes in a cipher as the security bound. The security bound is normally calculated for every number of rounds in a cipher. When we find that a cipher with a specific amount of rounds has as many minimum active s-boxes as the resistance bound, we know that the cipher is secure.

Mouha et al. calculate the security bound using MILPs. The objective is to minimize the number of active s-boxes. The constraints consist of the operations in the cipher and how the linear masks or differentials change. Since the output of one round is the input of the next round, if an output bit is active (set to one), then the input bit of the next round is also active and therefore part of a path.

Note that is it irrelevant if the path through the s-box has a high or low probability of actually happening. The MILP just looks for paths regardless of the precise specifications of the s-box. As an example, consider a cipher with r rounds and a resistance bound of x, i.e. it needs x active s-boxes in order to be resistant against the attack. Then, the security bound is calculated r times with an MILP for $1, ..., r$ rounds. This can result for example in the fact that the smallest path using four rounds has $x - 1$ active s-boxes, but the smallest path with five rounds has at least x active s-boxes, then the cipher is resistant against the attack only with five or more rounds. Since such a path does not exist, the attacker cannot find one with less than x active s-boxes, and thus the attack becomes useless to execute. Mouha et al. calculate exactly these security bounds in their paper for the ciphers AES and

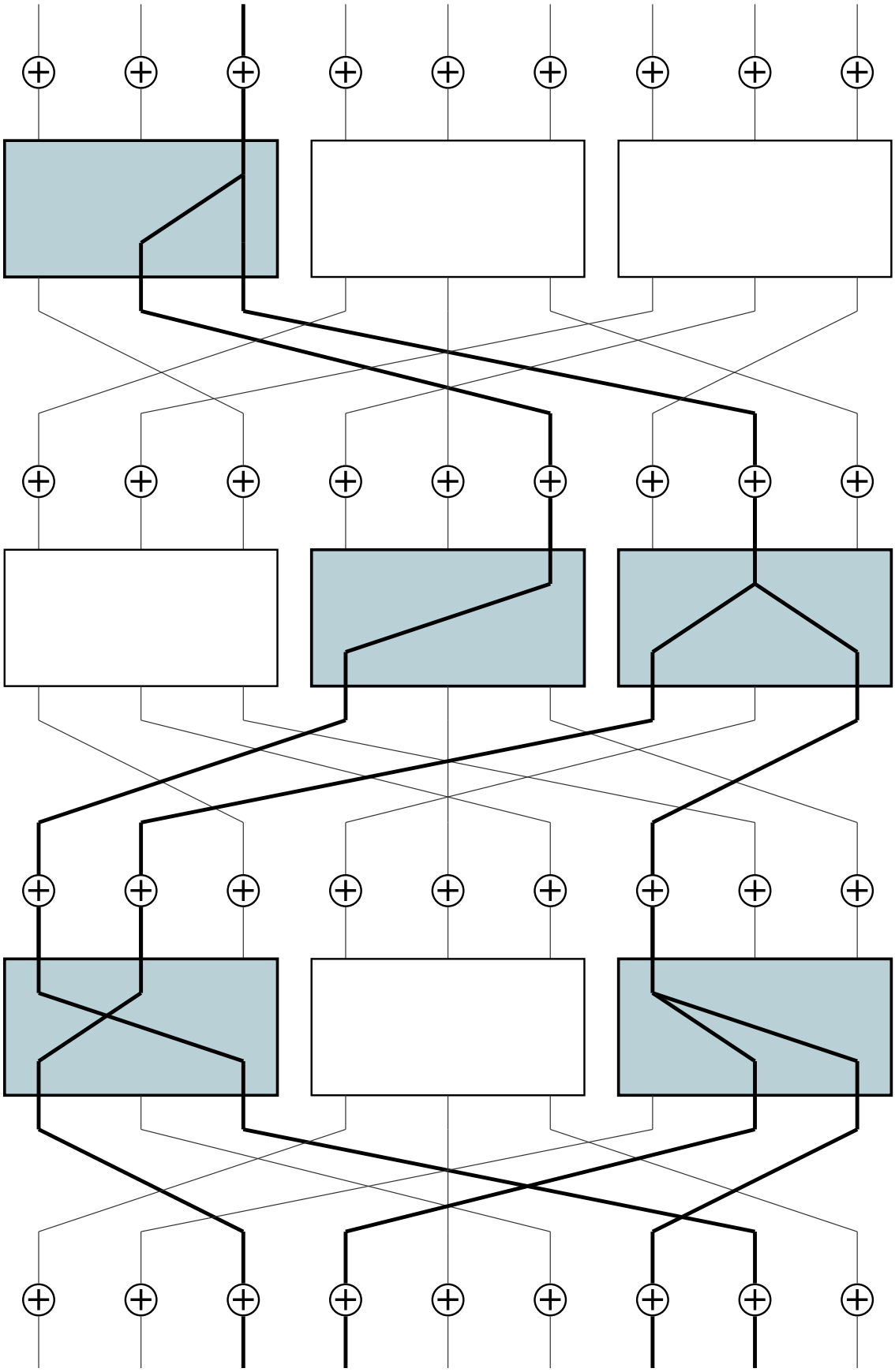


Figure 3.1. Example of a path through a cipher. The blue boxes describe an active s-box.

Enocoro-128v2. By using MILPs, the authors discovered a fast method to compute these bounds. But, since Mouha et al. are cryptanalysts, they put the generated MILP into a commercial solver without considering that there are structures for MILPs that can potentially accelerate the computation. In the next chapter, those generated MILPs will be analyzed to see if they have an inherent structure. But first, the process of constructing the security bound MILPs for AES and Enocoro-128v2 is described.

3.1 Differential Cryptanalysis Bound for Enocoro-128v2

For this cipher, an idealized variant is considered where the s-box and linear function can map any nonzero input difference to any nonzero output difference. Additionally, the branch number of the linear function is set to three. The minimum number of active s-boxes will be a lower bound for the real Enocoro-128v2. This is because Mouha et al. assume that the s-boxes in the cipher are independent as explained later, and also because of the idealized variant of the cipher. Therefore, it could be that the cipher is resistant against the attack with a lower number of rounds than Mouha et al. calculate, but this bound works in any case. For specifications about Enocoro-128v2, see Section 1.2.2 or the specification documents [21]. In one round, there are eight XOR operations and one linear function, as can also be seen in Figure 3.2. These nine operations are later portrayed as constraints of the MILP. We now go through the cipher with the difference vector and see how the active bits could behave. Instead of the input bytes $b_1, b_2, ..., b_{32}$ and a_0, a_1, we look directly at the difference vector $(x_0, x_1, ..., x_{33})$ of the initial value, which consists of binary variables. It is constructed from two input values in this way: Assuming $\delta = (\delta_0, \delta_1, ..., \delta_{33}) = (b_0 \oplus b_0', b_1 \oplus b_1', ..., a_1 \oplus a_1')$ is the difference of two input values, the difference vector is defined as $x = (x_0, x_1, ..., x_{33})$ with

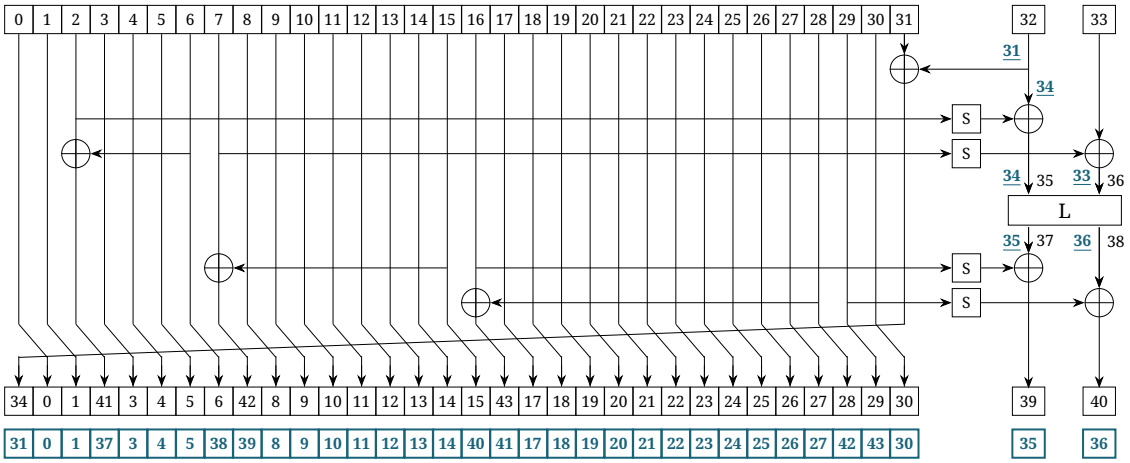
$$x_i = \begin{cases} 0, & \text{if } \delta_i = 0 \\ 1, & \text{otherwise.} \end{cases}$$

Thus, if in the difference of two bytes there is at least one bit different (one bit has value 1), the corresponding difference variable x_i will be set to $x_i = 1$. The corresponding x_i variables for the bytes in the first round of Enocoro-128v2 with differential cryptanalysis are depicted in Figure 3.2 in black.

We first look at the XOR operations. There, two bytes are XOR-ed, and we model the operation with our difference variables and see how the path goes through the operation. If both difference vectors are zero, then the difference of the output and therefore the new output difference vector is zero, and there is no path. If one of them is zero and the other is one, then the output difference vector is also one and there is a path. But, if the path is arriving from both sides, thus, both input difference vectors are one, the difference output vector is zero, since the differences cancel each other out. The path will stop there. By portraying this in the constraint of the MILP, the variables will be filled in the most optimal way. A XOR operation, e.g. for the XOR at the top of the cipher with x_{31}, x_{32} as input, can be described as

$$\begin{aligned} x_{31} + x_{32} + x_{34} &\geq 2d_0 \\ d_0 &\geq x_{31} \\ d_0 &\geq x_{32} \\ d_0 &\geq x_{34}. \end{aligned}$$

The branch number is set to two in order to display the XOR operation correctly. Mouha et al. introduce a dummy variable, which helps in modeling the XOR operation. This dummy variable, called d-variable, equals zero if all other values are zero, otherwise it takes the value one. To get an idea of why this works, consider a XOR operation as in Figure 1.3a. Assume that x_1, x_2 are the inputs and x_3



■ Figure 3.2. Differential and linear state in the first round of Enocoro-128v2. The number refer to the indices of the x-variables. The blue numbers depict the indices for the linear state whereas the black numbers correspond to the differential state.

is the output, and x_3 is the input for an s-box. Through previous rounds the variable x_2 is set to $x_2 = 1$. Since the output of the XOR operation x_3 is the input of an s-box, the MILP minimizes the value of x_3. Therefore, x_1, x_3 are set to $x_1 = 1$ and $x_3 = 0$ and the s-box will not be active.

The linear transformation is also portrayed through constraints. Since the branch number was constricted to three we know that at least three input and output bits have to be active (except for the all-zero case). To recall: we don't look at the specific values of the linear transformation, we just model the general behavior. Since we want to minimize the active bits, we want as many bits as possible to have the value zero. So, by taking the branch number which is the minimum number of active input and output bits, we are able to minimize the number of active bits while still having a correct depiction of the linear function. Because at least three bits will be active if one bit is. This linear transformation can be described as

$$\begin{aligned} x_{35} + x_{36} + x_{37} + x_{38} &\geq 3d_1 \\ d_1 &\geq x_{35} \\ d_1 &\geq x_{36} \\ d_1 &\geq x_{37} \\ d_1 &\geq x_{38}. \end{aligned}$$

where (x_{35}, x_{36}) is the input difference vector and (x_{37}, x_{38}) is the output difference vector in the first round. From now on, a constraint that contains the inputs and outputs of a function like the constraint in the first row will be called long constraint. A short constraint then describes a constraint consisting of a d-variable and an x-variable. The d-variable again equals zero if all variables are zero. Otherwise, if there is at least one variable that is nonzero, then the path goes through the linear function. For every round, there are eight XOR operations that have each four equations and one linear function with five equations, thus $8 \cdot 4 + 1 \cdot 5 = 37$ equations. Additionally, ten new x-variables and nine new d-variables are created in a round. Furthermore, a constraint is added in the end to ensure that in the result there is at least one active s-box.

The objective function is the sum of the input difference of all s-boxes. In the first round for example, the s-box input variables are x_2, x_7, x_{16} and x_{29}. If the input difference is one, then the s-box is active. Therefore, the sum of these variables has to be minimized. Consider D_i to be the set of the indices of all input variables of the s-boxes in round i.

Then,

$$D_1 = \{2, 7, 16, 29\}$$
$$\vdots$$
$$D_{96} = \{954, 941, 902, 863\}.$$

And with

$$I_N = \bigcup_{1 \leq i \leq N} D_i,$$

the objective function can be described as

$$minimize \sum_{j \in I_N} x_j.$$

The last constraint that ensures that at least one s-box is active is

$$\sum_{j \in I_N} x_j \geq 1.$$

If all variables are set to be binary, the program would be an ILP. We know that when an ILP and an MILP have the same amount of variables, the MILP can be solved more efficiently since it contains less integer variables. For this reason we transform the ILP into an MILP by setting only the d-variables and the input variables of the first round to be binary values. The rest of the variables can take any value. The resulting values will actually still be binary because they are created from the input variables, which are binary. This way, we improve the efficiency of the problem but still have the same result.

Mouha et al. have inserted the equation into an MILP solver and calculated for every number of rounds the minimum number of active s-boxes. If there are 14 active s-boxes, the cipher is secure against differential cryptanalysis. This can be calculated in the following way: The MDP, introduced in Definition 2.3, for the Enocoro-128v2 s-box is $\frac{10}{256} = 2^{\log_2(\frac{10}{256})} = 2^{-4.678}$ as Mouha et al. stated. Since the initial value consists of 64 bits, there are 2^{64} pairs for any certain difference (because every value has exactly one value with which it reaches the difference). The number of required chosen plaintexts for the attack is $1/p$ as described by Biham [2]. The variable p denotes the probability for the characteristic of the cipher, thus the overall probability for a path in a cipher. Because our main goal is to be secure against attacks, we give the attacker the best advantages. In a realistic attack, the cipher would not always behave in the worst case. But by assuming exactly this, we can be absolutely safe that our calculations work in every case. Thus, we assume every active s-box takes the MDP and calculate the overall probability p by multiplying the MDP of all active s-boxes with each other. Normally, the s-boxes are not independent, but if they would be, it would be easier for the attacker. So, we give the attacker again the advantage and act as if the s-boxes are independent. Then the overall probability is $p = 2^{-4.678^{k_N}} = 2^{-4.678k_N}$. In order for differential cryptanalysis to work, $1/p = 2^{4.678k_N}$ chosen plaintexts are required. If the number of required chosen plaintexts is larger than the number of possible inputs, the attack becomes useless because the attacker could just brute-force the process. Therefore, in this case the cipher would be secure against differential cryptanalysis,

$$2^{4.678k_N} \geq 2^{64},$$

and from this follows

$$k_N \geq 64/4.678.$$

Thus, with $14 > 64/4.678$ active s-boxes the cipher is resistant against differential cryptanalysis. When calculating the MILP for every number of possible rounds, the conclusion is that with 38 rounds the cipher is secure because the minimum number of active s-boxes is 14, as Mouha et al. stated. There is not a path through this cipher which goes through less than 14 s-boxes in 38 rounds. In Chapter 4 we will see a detailed analysis of this differential cryptanalysis MILP.

3.2 Linear Cryptanalysis Bound for Enocoro-128v2

Similar to differential cryptanalysis, we use an idealized variant of Enocoro-128v that has maximally as many linear active s-boxes as the real cipher. Instead of the buffer $(b_0, b_1, ..., b_{31})$ and state (a_0, a_1) byte vectors, we consider the linear mask vector $(x_0, x_1, ..., x_{33})$ consisting of binary variables. Assuming $\Gamma = (\Gamma_0, \Gamma_1, ..., \Gamma_{33})$ is a linear mask of all 34 bytes, the linear mask vector is defined as $x = (x_0, x_1, ..., x_{33})$ with

$$x_i = \begin{cases} 0, & \text{if } \Gamma_i = 0 \\ 1, & \text{otherwise.} \end{cases}$$

In other words, the vector indicates for every byte if at least one bit is 'active' and therefore if the byte is being used in the path for the attack. Now we want to model possible paths through the cipher with a MILP. The indices for the x-variables corresponding to a byte can be found in Figure 3.2 in blue.

This time, we don't consider the XOR operations. This is because the input and output linear mask vectors are always the same. To recall: the linear masks indicate which bits are ignored. So, when XOR-ing two bytes, the result will not have impact on the output linear mask. The output linear mask just indicates which bits to ignore, which are the same as in the input. Thus, the bits that describe the input linear masks and output linear mask are equal. But, the three forked branch is relevant in finding a path. Biham describes this 'phenomena' as 'just the opposite to the usual operations in the cryptosystem' [3]. In differential cryptanalysis, we treated the three forked branches as 'duplications', thus the input bit matches the two output bits, and the XOR operation was treated like a XOR operation. Now, when describing these operations in the MILP, we treat the XOR operation as a three forked branch: the input bit is just duplicated and equals the two output bits. The three forked branch on the other hand is treated like a XOR operation and XORs the input bit with the original output bit to form the second output bit. As illustrated in Figure 3.2, every three forked branch has an output variable that already exists and one output variable that is new. This 'phenomena' describes one of the most significant differences in linear and differential cryptanalysis. As can be seen in Figure 1.3b, there are three variables for the three forked branch: one for the input and two for the output. If the input bit is active, then at least one output bit has to be active too, because the path cannot end abruptly. If the input bit is 0 and therefore not part of a path, the output bits will also be inactive. This can be described with the following equations. The case where the input bit is zero, but both output bits are one will be avoided because the MILP minimizes the number of active bits and therefore sets the output bits to zero. Here, x_{31} is the input bit and x_{32} and x_{34} are the output bits. The following inequalities describe the three forked branch that is at the top of the cipher in Figure 3.2.

$$\begin{aligned} x_{31} + x_{32} + x_{34} &\geq 2d_0 \\ d_0 &\geq x_{31} \\ d_0 &\geq x_{32} \\ d_0 &\geq x_{34}. \end{aligned}$$

The branch number is set to two. The d-variable is set to zero if every other variable is zero and otherwise is set to one.

The minimal number of nonzero linear input and output masks for the linear function is three when excluding the cases where every mask is zero. Thus, the linear branch number is three. So now, when one bit in the linear mask vector of the input or output is active, we know that at least two more variables have to take the value one. Of course, we do not know if the input/output linear mask combination chosen with the equations is existing in real life, but this is not relevant for us.

The important thing is to model a path through a cipher and see if an s-box is active, and not *which* bits make the s-box active. The linear function in the first round can be portrayed with the following constraints

$$\begin{aligned} x_{34} + x_{33} + x_{35} + x_{36} &\geq 3d_1 \\ d_1 &\geq x_{34} \\ d_1 &\geq x_{33} \\ d_1 &\geq x_{35} \\ d_1 &\geq x_{36} \end{aligned}$$

where (x_{34}, x_{33}) is the input vector and (x_{35}, x_{36}) is the output vector. The d-variable is there to ensure that at least three bits are active when at least one bit is active. In one round, the cipher has eight three forked branches and one linear function. Thus, one round can be described with $8 \cdot 4 + 1 \cdot 5 = 37$ equations. Since every three forked branch generates one new variable and the linear function generates two new variables, there are ten new x-variables at the end of each round. Additionally, nine new d-variables are created in a round. By summating every input linear mask variable of an s-box together, one gets the objective function. To get the minimum number of active s-boxes, this sum has to be minimized. In the first round, the input variables for the s-box are x_{34}, x_{33}, x_{35} and x_{36}. Let L_i be the set of indices of s-box input variables for round i. Then,

$$L_1 = \{34, 33, 35, 36\}$$

$$\vdots$$

$$L_{96} = \{984, 976, 985, 986\}.$$

Using

$$J_N = \bigcup_{1 \leq i \leq N} L_i,$$

the objective function is defined as

$$minimize \sum_{j \in J_N} x_j.$$

Analogue to the MILP for differential cryptanalysis, the d-variables and the input variables of the first round are restricted to be binary, and the rest can take any value. The program is thus an MILP, but every variable automatically takes a binary value. To extract the required number of active s-boxes to be resistant against linear cryptanalysis, the resistance bound, Matsuis Piling-up Lemma [27] is needed:

▶ Lemma 3.1 (Piling-up Lemma, 1993). *Assuming there are n independent random binary variables $X_1, X_2, ..., X_n$ that take the value 0 with probability p_i, the probability that $X_1 \oplus X_2 \oplus ... \oplus X_n = 0$ equals*

$$Pr(X_1 \oplus X_2 \oplus ... \oplus X_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^{n} (p_i - 1/2)$$

and since the bias $\epsilon = p - 1/2$,

$$\epsilon = 2^{n-1} \prod_{i=1}^{n} \epsilon_i$$

and ϵ is the bias of the equation $X_1 \oplus X_2 \oplus ... \oplus X_n = 0$.

In our case, the random binary variables $X_1, X_2, ..., X_n$ will represent the linear approximation of the s-boxes. Thus, the linear expression for the cipher holds true with a bias of ϵ. For the s-box in

Enocoro-128v2 the MLP is 2^{-3}. The MLP was stated in Definition 2.1. Assuming that every s-box takes the worst approximation, the bias is $\epsilon_i = 2^{-3}$ for every s-box $i \in n$. This is because the MLP is already the deviation of the probability from $1/2$ so the MLP is already the bias. Consider a as the number of active s-boxes, then the overall bias ϵ is $2^{a-1} \cdot (2^{-3})^a$. This only works under the assumption that the approximations of the s-boxes are independent, which is not exactly true, but the calculations are sufficiently correct in practice. According to Matsui [27], to attack with linear cryptanalysis approximately ϵ^{-2} known plaintexts are needed. This is because the number of required plaintexts N_L is proportional to ϵ^{-2}, $N_L \approx \frac{1}{\epsilon^2}$. Consequently, if the attacker has N_L plaintexts, linear cryptanalysis can be applied. There exist 2^{64} different plaintexts for Enocoro-128v2, as the number of IVs is limited to 64 bits. Every bit can take the value 0 or 1, thus two values, and these possibilities exist 64 times, thus 2^{64}. If more than 2^{64} known plaintexts are needed, the attacker could also find the key by brute force, since they have every single plaintext. Using linear cryptanalysis would become obsolete. Thus, in order to be resistant against the attack, we want

$$\epsilon^{-2} > 2^{64},$$

with our bias

$$(2^{a-1} \cdot (2^{-3})^a)^{-2} > 2^{64},$$

and subsequently

$$a > 15.5\,.$$

It follows that the cipher is secure against linear cryptanalysis if there are at least 16 active s-boxes. Calculating the MILP with every number of rounds, Mouha et al. find that with 61 rounds, the minimum number of linearly active s-boxes is 18 (for 60 rounds it is 15). A detailed analysis of this linear cryptanalysis MILP follows in Chapter 4.

3.3 Linear and Differential Cryptanalysis Bound for AES

For the cipher AES the MILPs for linear and differential cryptanalysis are equivalent. This is because there is no three forked branch or XOR operation in the cipher that regards the state variables and not the key variables. The linear as well as the differential branch number for the *MixColumns()* operation is five, as stated by Daemen and Rijmen [15]. The *MixColumns()* operation is also called linear function.

In the beginning, the 128 bits are saved in a 4×4 array and every entry consists of one byte. For the MILP modeling, the array consists of 16 variables which describe the linear mask or difference of the byte. If the linear mask consists of at least one bit that has taken the value one or the differential has at least one bit that contains a difference, then the value of the x-variable is one.

The first step in a round is *AddRoundKey()*, but since this does not change the differential vector nor the linear mask vector, it is not relevant for the MILP. In differential cryptanalysis, two keys that are the same are added to the differential. The XOR-sum of the keys will be zero, since they cancel each other out. Thus, *AddRoundKey()* is not relevant. For linear cryptanalysis, the adding of the round key will not change which bits have to be ignored and which bits are active. Therefore, the linear mask will not differ after the *AddRoundKey()* operation.

In the *SubBytes()* step, each byte is put into the s-box, and if the value of the corresponding x-variable is one then the s-box is active and otherwise not. We want to minimize the number of active s-boxes, thus we want as many variables as possible to take the value $x_i = 0$. This is reflected in the objective function. Contrary to the Enocoro-128v2 cipher, every byte from the state array in every round will be going through the s-box, except for the new bytes in the last round. In the permutation layer, the variables are first shifted with *ShiftRows()*. The values of the variables stay the same, but

the state array is updated. Since they do not change value, the operation is not described in the constraint matrix. The state array changes and the following operations are executed with the new order of variables.

In the *MixColumns()* operation, every column is multiplied with a certain matrix. This is a linear function with four input and four output variables. The output variables are new variables that are the input for the next round. We know from the specifications that the branch number is five. Therefore, if one variable is active, at least four other variables are active too. More variables could be active, but in the minimum case five variables have to be active (except when no variable is active). Since the goal is to minimize the overall number of active variables, we take the branch number as the bound if at least one variable is active. The *MixColumns()* operation is described as

$$\begin{aligned} x_0 + x_5 + x_{10} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} &\geq 5d_0 \\ d_0 &\geq x_0 \\ d_0 &\geq x_5 \\ d_0 &\geq x_{10} \\ d_0 &\geq x_{15} \\ d_0 &\geq x_{16} \\ d_0 &\geq x_{17} \\ d_0 &\geq x_{18} \\ d_0 &\geq x_{19} \end{aligned}$$

where x_0, x_5, x_{10}, x_{15} are the input variables (thus the values of the column that will be multiplied) and $x_{16}, x_{17}, x_{18}, x_{19}$ are the new variables that result from the multiplication. The d-variable equals zero if every variable is also zero, otherwise it is one. For each round, 16 new x-variables and four new d-variables are created.

The objective function is the sum of the input variables of each s-box. Since in every round, every input variable in the matrix is put into an s-box, each x-variable (except the new variables of the last round) is in the objective function. For N rounds, it looks like this:

$$minimize \sum_{i=1}^{16N} x_i.$$

As with the MILP for Enocoro-128v2, a single equation is needed in order to exclude the trivial solution of no active s-boxes. This constraint has the following form:

$$\sum_{i=1}^{16N} x_i \geq 1.$$

In one round there are $9 \cdot 4$ constraints, thus for N rounds there are $36N + 1$ constraints. In order to get an MILP, the first input variables $(x_0, ..., x_{15})$ and all d-variables $(d_0, ..., d_{4r-1})$ are restricted to be binary, the rest is free. By doing this, we increase the efficiency, but all variables still take binary values.

Daemen and Rijmen [15] proved that the minimum number of s-boxes in a linear or differential cryptanalysis setting when doing four rounds is 25. Using the MILP created by Mouha et al., and calculating the minimum number of s-boxes, the result is the same. In Chapter 4 this MILP is analyzed and a new-found structure is presented in detail.

4 Analysis of Security Bound MILPs

After establishing how security bound MILPs are constructed in Chapter 3, we will describe how we analyzed them with a Python framework and present the newly revealed structures of the differential and linear cryptanalysis MILPs in this chapter.

4.1 Python Framework for Blockstructured Analysis

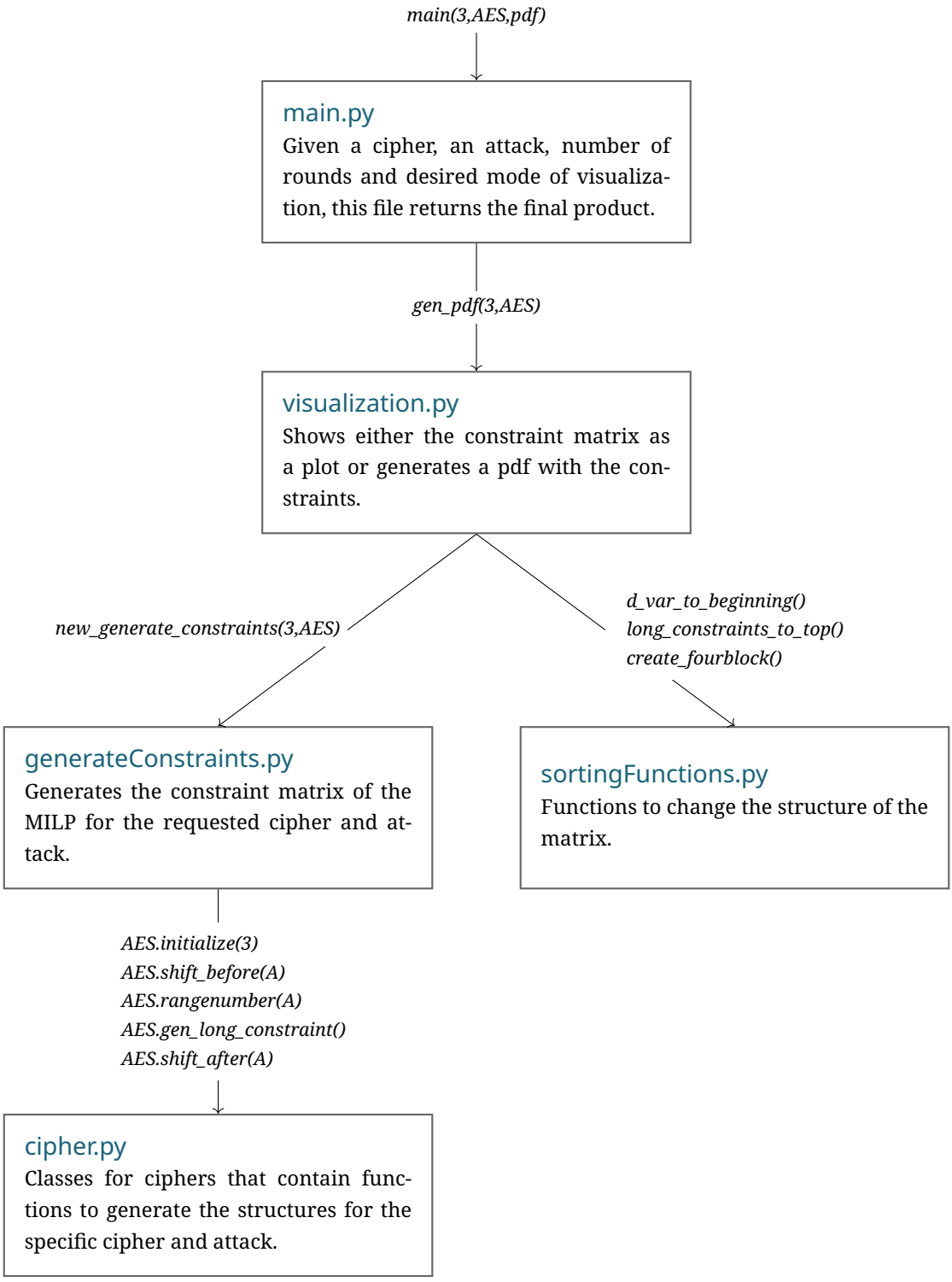
The framework computes the constraints and visualizes the constraint matrix of the MILP. It can generate the constraints for the block cipher AES and the stream cipher Enocoro-128v2 in the differential and linear cryptanalysis setting. But the most important part is the modular structure. This way, the framework can not only generate the constraint matrix for two ciphers, but for any other cipher. One just has to add a class, and the other parts of the code do the rest. Furthermore, there are functions to alter the structure of the constraint matrix. Finally, the visualizations are portrayed as a plot or in a PDF file. The modular structure can be seen in Figure 4.1. This structure allows to easily add or modify functionalities. Below, a description for each file can be found.

cipher.py

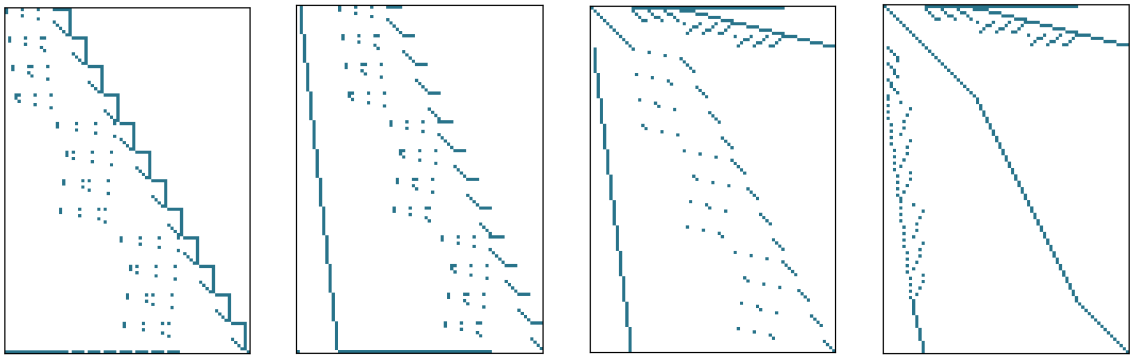
The file `cipher.py` contains a class for each cipher and mode of attack. Two classes for differential and linear cryptanalysis in Enocoro-128v2, and a class for AES, which is used for both cryptanalysis types. Every class has the same functions that differ in their way to generate the matrix. In *initialize()* an empty matrix with the right size dependent of the number of rounds is initialized and the initial variables are set. *input_sbox()* returns a list with all variables that are the input to an s-box. This is needed for the constraint that ensures that at least one s-box is active. There are functions that are used if the values in the cipher shift before or after a round (*shift_before, shift_after*). For every cipher operation (e.g. XOR, three forked branch or linear function) there is one long constraint that describes it and then for each variable in it a constraint with just the d-variable. *gen_long_constraint()* generates this first long constraint. The small constraints with the d-variables are constructed in the same way for every cipher, thus they are generated in another file (since it is not cipher- or attack-specific). If one wants to use the code for another cipher or attack, they would have to add a new class in this file. The visualization and sorting of the matrix will work immediately.

generateConstraints.py

Using the functions of `cipher.py`, the file `generateConstraints.py` creates the memory representation for constraint matrix of the MILP. Given a cipher and mode of attack, *new_generate_constraints()* returns the matrix and the corresponding vector. After initializing the empty matrix, the code goes through every round to fill in the entries. If necessary, it shifts the position of the values before or after a round. In between, it generates the long constraint for every operation. The function *generate_smallconstraints()* constructs the small constraints that consist of just the d-variable and a variable from the long constraint. The constraints are constructed in a slightly different manner than in



■ Figure 4.1. Python framework for generating and analyzing constraint matrices of MILPs that find the minimum number of active s-boxes in a cipher given a certain attack. Along the arrows an example call for AES with three rounds and PDF visualization can be found.



a. native Matrix b. d_var_to_beginning() c. long_constraints_to_top() d. creating _diagonal_in4block()

■ Figure 4.2. Example for the sorting functions. The subfigures show the structure of the matrix when the corresponding function is applied to the matrix in the subfigure before. Every non-white entry describes a nonzero entry.

the previous chapter. The difference is that the right-hand side of the equation is changed to be an all-zero vector. Thus, from $x_1 + x_2 + x_3 \geq 2d_0$ follows the constraint $-2d_0 + x_1 + x_2 + x_3 \geq 0$ and a short constraint $d_0 \geq x_1$ transforms into $d_0 - x_1 \geq 0$. Additionally, for the constraint that ensures that at least one s-box is active, we introduce a new constant l in the solution vector. Mouha et al. set this constraint to e.g. $x_2 + x_4 + x_6 \geq 1$ when x_2, x_4, x_6 are the input variables to all the s-boxes in the cipher. In order to get a zero on the right-hand side, we would now transform the constraint into $-1 + x_2 + x_4 + x_6 \geq 0$. To portray this integer on the left side when having the constraints displayed in a constraint matrix, we introduce the constant $l = 1$. Then, the constraint looks like this $-1 \cdot 1 + x_2 + x_4 + x_6 \geq 0$, and we have an additional entry in the solution vector.

sortingFunctions.py

Now that the representation of the constraint matrix in memory is created, it is still unstructured and convoluted, since it was generated on a round-basis. The functions in the `sortingFunctions.py` file are able to reveal the structures of the matrices. The functions *permutate_rows(), permutate_columns()* permutate the rows or columns of the matrix in a specific given order. Since the d-variables are newly generated in each round, they are scattered troughout the columns. With *d_var_to_beginning()* the columns corresponding to the d-variables are at the beginning of the matrix which then has a clearer structure. This function is used to get the stair-fold structure. Then, *long_constraints_to_top()* permutates the long constraints to the top rows. As a result there is a block on the top rows and the rest of the matrix is sparse. The function *creating_diagonal_in4block()* then tries to find a diagonal that goes through the matrix by permutating the rows. This helps in getting a 4-block structure, if there is one. In Figure 4.2 these functions are applied subsequently to an example and one can see how in the end the 4-block structure is built. The function *create_fourblock()* combines the three functions. Given a freshly generated matrix, the function permutates the columns for the d-variables to the left, the rows for the long constraints to the top and creates the diagonal.

visualization.py

In the file `visualization.py` the results are visualized. One can either generate a PDF or a detailed version of the matrices. The detailed version generates four representations of the same matrix. It shows the nonzero entries in order to depict the structure more clearly. They are generated with Matplot. The PDF contains the Matplot plots and then all constraints that describe the matrix. The

function *constraints()* takes the matrix and the vector and translates it into LaTeXcode.

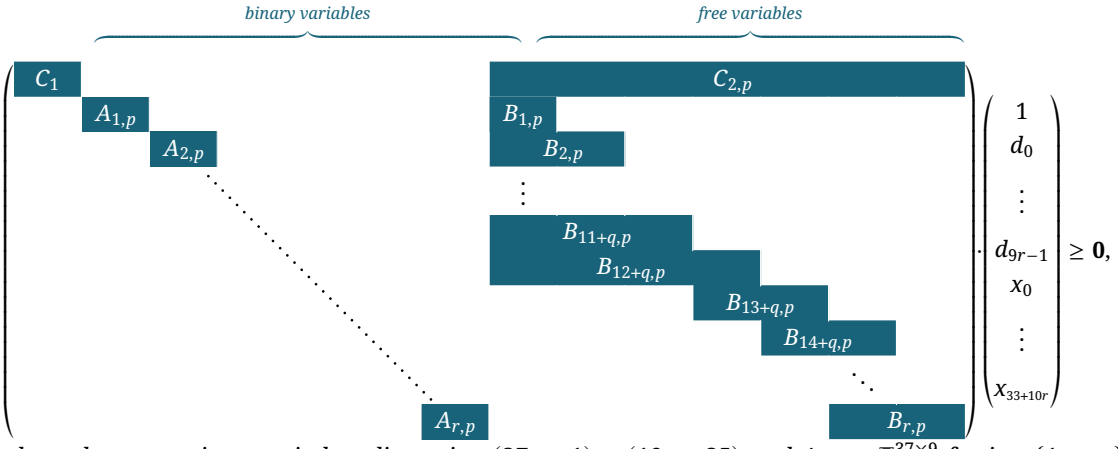
main.py

Finally, the `main.py` file provides a way to easily generate the constraint matrix for the wanted cipher and attack. An example call for the cipher AES with 3 rounds visualized in a PDF can be found in Figure 4.1. After giving as input the number of rounds, cipher and mode of visualization, the function *main()* calls the function *gen_pdf()* in the `visualization.py` file. In this function, the constraints are first generated with *new_generate_constraints()* in `generateConstraints.py` which calls the functions of the class *AES* in the `cipher.py` file. Then, the sorting functions from `sortingFunctions.py` are applied to the matrix. The resulting matrices are then put in the PDF file which is the output of the main function.

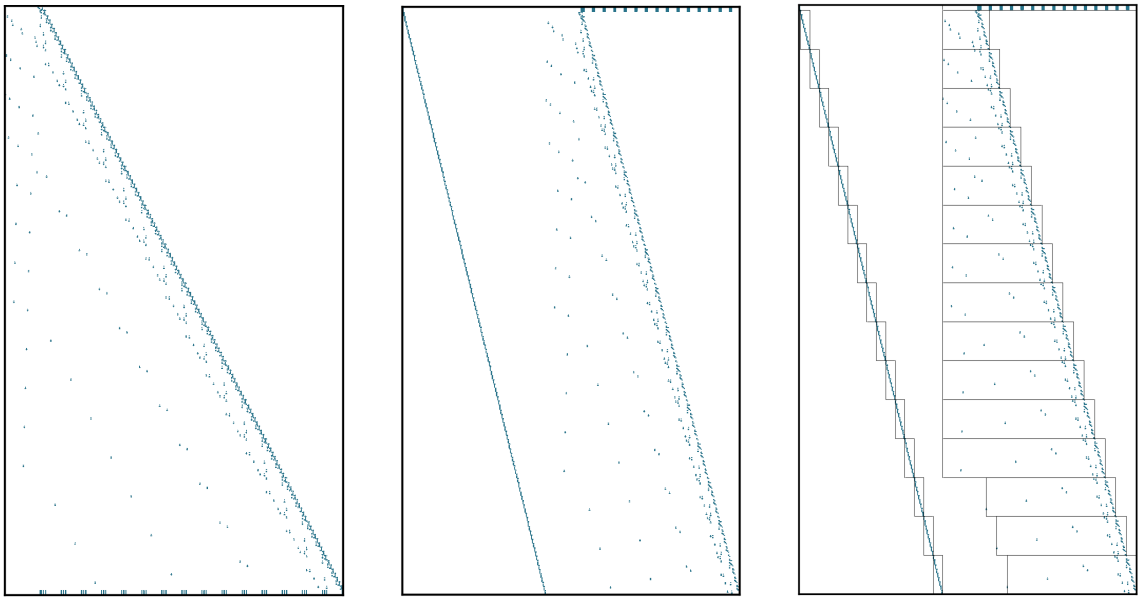
4.2 Stair-Fold MILP for Enocoro-128v2

Since Enocoro-128v2 is a very secure cipher [18, 11], it has a high diffusion. For example, the variables used in one round are not only the ones from the last round like in AES, but also from the rounds that came before. Therefore it is difficult to find a matrix structure like 4-block where the blocks are independent from each other. Though, the MILP can take the stair-fold structure defined in 1.3. The stair-fold structure allows the blocks to be dependent of each other. This section will give a description for the matrix without having to go through every operation in the cipher again. This is an advantage because it reduces the intricacy and allows one to focus on the MILP instead of the cryptanalysis. To see how the stair-fold structure was achieved, consider Figure 4.3, where the constraint matrix structure for the linear cryptanalysis MILP of Enocoro-128v2 cipher with fifteen rounds is shown. The native matrix is depicted in Figure 4.3a. Then, with the help of our sorting functions, the columns corresponding to the d-variables are permutated to the beginning of the matrix and the last constraint was rearranged into the first row. This stair-fold structure becomes clear in Figure 4.3c, where the outline of the blocks is added. The matrix is still round-based, so the blocks correspond to a round. There are three types of blocks in the matrix, depicted in Theorem 4.1.

▶ Theorem 4.1 (Stair-fold MILP for Enocoro-128v2). *Let r be the number of rounds. The MILP for Enocoro-128v2 has a stair-fold structure, i.e. we have*



where the constraint matrix has dimension $(37r+1) \times (19r+35)$ and $A_{i,p} \in \mathbb{Z}^{37\times 9}$ for $i = \{1, ..., r\}$, $C_1 \in \mathbb{Z}^{1\times 1}$, $C_{2,p} \in \mathbb{Z}^{1\times 34+10r}$ and $B_{i,p} \in \mathbb{Z}^{37\times 34+10i}$ for $i = \{1, ...12+q\}$ and $B_{i,p} \in \mathbb{Z}^{37\times 123+\frac{19}{3}q}$ for $i = \{13+q, ..., r\}$. Further, we have $d_k, x_j \in \{0,1\}$ for $k = \{0, ..., 9r-1\}$ and $j = \{0, ..., 33\}$ and $x_j \in \mathbb{R}$ for $j = \{34, ..., 33+10r\}$. The variables p, q depend on the attack.



a. native Matrix b. sorted matrix c. stair-fold structure

■ Figure 4.3. Constraint matrix for the linear cryptanalysis MILP for 15 rounds in the cipher Enocoro-128v2. Every non-white entry is a nonzero entry in the matrix. The C-block was enhanced in order to make it visible.

The A-blocks describe the d-variables, the B-blocks the x-variables and the C-blocks correspond to the constant $l = 1$ and the x-variables. The d-variables and the variables that are the input to the first round have to take binary values. The rest of the variables have no restrictions. In the matrix, there are r A- and B-blocks and the blocks $A_{i,p}, B_{i,p}$ describe the round i. The blocks C_1 and $C_{2,p}$ construct the first row which describes the constraint that ensures that at least one s-box is active. A description for every type of block follows.

▶ Corollary 4.2 (A-blocks). *The $A_{i,p}$-blocks have the following form*

$$R = \begin{pmatrix} -2 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad F = \begin{pmatrix} -3 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad A_{i,p} = \begin{pmatrix} R & & & \\ & \ddots & & \\ & & R & \\ & & & F \\ & & & & R & \\ & & & & & \ddots \\ & & & & & & R \end{pmatrix} \Big\} \, p \text{ times}$$

where R describes a XOR operation or three forked branch, F describes the linear function. For an entry y of matrix $A_{i,p}$ holds $y \in \{-3, 1\}$.

▶ Corollary 4.3 (B-blocks). *The $B_{i,p}$-blocks are constructed in the following way*

$$B_{i,p} = \begin{pmatrix} T_{1,i,p} \\ \vdots \\ T_{p,i,p} \\ L_{i,p} \\ T_{p+1,i,p} \\ \vdots \\ T_{8,i,p} \end{pmatrix} \Big\} \, p \text{ times} \qquad T_{k,i,p} = \begin{pmatrix} m_{k,i,p} & n_{k,i,p} & o_{k,i,p} \\ 1 & 1 & 1 \\ -1 & & \\ & -1 & \\ & & -1 \end{pmatrix}$$

with the submatrices $T_{k,i,p}$ that describe the three forked branch or XOR operation. The submatrices $L_{i,p}$ have the form

$$L_{i,p} = \begin{pmatrix} \overbrace{}^{10i+42-6p} & \overbrace{}^{-3.5(p-3)} & & & \overbrace{}^{8-p} \\ 1 & 1 & 1 & 1 & \\ -1 & & & & \\ & -1 & & & \\ & & -1 & & \\ & & & -1 & \end{pmatrix} \quad L_{1,1} = \begin{pmatrix} \overbrace{}^{33} & & & & \overbrace{}^{7} \\ 1 & 1 & 1 & 1 & \\ -1 & & & & \\ & -1 & & & \\ & & -1 & & \\ & & & -1 & \end{pmatrix}$$

where the submatrix with $p = 1$ and for the first round $L_{1,1}$ has a different form. For an entry y of matrix $B_{i,p}$ holds $y \in \{-1, 1\}$.

▶ Corollary 4.4 (*C*-blocks). *The two C-blocks C_1 and $C_{2,p}$ have the following form*

$$C_1 = \begin{pmatrix} -1 \end{pmatrix} \qquad C_{2,p} = \begin{pmatrix} d_{1,p} & e_{1,p} & f_{1,p} & g_{1,p} & \cdots & d_{r,p} & e_{r,p} & f_{r,p} & g_{r,p} \\ 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 \end{pmatrix}$$

where every entry in the $C_{2,p}$-block describes a variable that is the input to an s-box. For an entry y of matrix $C_{2,p}$ holds $y \in \{0, 1\}$.

4.2.1 Differential Cryptanalysis Stair-Fold MILP for Enocoro-128v2

The stair-fold MILP for differential cryptanalysis can be generated using the Theorem 4.1. The variables p, q have to be set to $p = q = 3$. Each block can then be constructed with the following details:

Block A

The blocks $A_{i,3}$ depicted in Corollary 4.2 describe the part of the constraints where the d-variable is used. There are r blocks, and $A_{i,3}$ then denotes the block for round i in differential cryptanalysis. All variables corresponding to the $A_{i,3}$-blocks have to take a binary value. As explained earlier, a XOR operation needs four constraints and a linear function needs five. Since there are eight XOR operations and one linear function in a round, the blocks consist of 37 rows. The dimension of an $A_{i,3}$-block is 37×9. The constraints of an operation consist of the long constraint where the sum of the x-variables has to be greater or equal to the d-variable times the branch number and the short constraints where an x-variable has to be greater or equal to the d-variable. Therefore the entries of the block are the branch number (-2 or -3) for the long constraint and one for the short constraints. To simplify the visualization, we introduce submatrices for the blocks: R, which describes the entries for a XOR operation and F which describes the entries for a linear function.

Block B

The B-blocks define the values corresponding to the x-variables in the matrix. There are r B-blocks, one for each round. In Corollary 4.3 the $B_{i,3}$-blocks are depicted for differential cryptanalysis. The blocks up until the 15th round consist of $10i + 34$ columns and 37 rows. From the 16th round, the blocks have the dimension 142×37. The variables belonging to the first 34 columns in the $B_{1,3}$-block have to take binary values. The rest of the variables is free. Each constraint and therefore row belongs to a certain operation in the cipher. Just like in the A-blocks, we have four rows belonging to a XOR operation and five rows belonging to a linear function. The first row for an operation describes the long constraint, thus it has three entries for a XOR operation and four entries for the linear function. The rows following this first row describe the short constraints and thus have one entry. Because

$$m_{1,i,3} = \begin{cases} 33 - i, & \text{if } i <= 15 \\ 43 + 10 \cdot (i - 16) - z_i, & \text{otherwise} \end{cases}$$

$$n_{1,i,3} = \begin{cases} 33, & \text{if } i <= 1 \\ 20 + 10i - z_i, & \text{otherwise} \end{cases}$$

$$o_{1,i,3} = 25 + 10i - z_i$$

a. first XOR operation

$$m_{2,i,3} = \begin{cases} 4 - i, & \text{if } i <= 3 \\ 44 + 10 \cdot (i - 4) - z_i, & \text{otherwise} \end{cases}$$

$$n_{2,i,3} = \begin{cases} 33, & \text{if } i <= 1 \\ 20 + 10i - z_i, & \text{otherwise} \end{cases}$$

$$o_{2,i,3} = 26 + 10i - z_i$$

b. second XOR operation

$$m_{3,i,3} = \begin{cases} 9 - i, & \text{if } i <= 5 \\ 42 + 10 \cdot (i - 6) - z_i, & \text{otherwise} \end{cases}$$

$$n_{3,i,3} = \begin{cases} 34, & \text{if } i <= 1 \\ 21 + 10i - z_i, & \text{otherwise} \end{cases}$$

$$o_{3,i,3} = 27 + 10i - z_i$$

c. third XOR operation

$$m_{4,i,3} = \begin{cases} 18 - i, & \text{if } i <= 9 \\ 43 + 10 \cdot (i - 10) - z_i, & \text{otherwise} \end{cases}$$

$$n_{4,i,3} = 28 + 10i - z_i$$

$$o_{4,i,3} = 30 + 10i - z_i$$

d. fourth XOR operation

$$m_{5,i,3} = \begin{cases} 31 - i, & \text{if } i <= 13 \\ 44 + 10 \cdot (i - 14) - z_i, & \text{otherwise} \end{cases}$$

$$n_{5,i,3} = 29 + 10i - z_i$$

$$o_{5,i,3} = 31 + 10i - z_i$$

e. fifth XOR operation

$$m_{6,i,3} = \begin{cases} 4 - i, & \text{if } i <= 3 \\ 35 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$n_{6,i,3} = \begin{cases} 8 - i, & \text{if } i <= 4 \\ 42 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$o_{6,i,3} = 32 + 10i - z_i$$

f. sixth XOR operation

$$m_{7,i,3} = \begin{cases} 9 - i, & \text{if } i <= 5 \\ 42 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$n_{7,i,3} = \begin{cases} 17 - i, & \text{if } i <= 8 \\ 43 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$o_{7,i,3} = 33 + 10i - z_i$$

g. seventh XOR operation

$$m_{8,i,3} = \begin{cases} 18 - i, & \text{if } i <= 9 \\ 43 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$n_{8,i,3} = \begin{cases} 30 - i, & \text{if } i <= 12 \\ 44 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$o_{8,i,3} = 34 + 10i - z_i$$

h. eighth XOR operation

■ Figure 4.4. Positions of the entries for each round and XOR operations for $T_{k,i,3}$, a subblock of $B_{i,3}$

$$d_{i,3} = \begin{cases} 2 - i, & \text{if } i <= 2 \\ 34 + 10 \cdot (i - 3), & \text{otherwise} \end{cases} \qquad e_{i,3} = \begin{cases} 7 - i, & \text{if } i <= 4 \\ 41 + 10 \cdot (i - 5), & \text{otherwise} \end{cases}$$

$$f_{i,3} = \begin{cases} 16 - i, & \text{if } i <= 8 \\ 42 + 10 \cdot (i - 9), & \text{otherwise} \end{cases} \qquad g_{i,3} = \begin{cases} 29 - i, & \text{if } i <= 12 \\ 43 + 10 \cdot (i - 13), & \text{otherwise} \end{cases}$$

■ Figure 4.5. Positions of entries in the $C_{2,3}$-block for differential cryptanalysis

every operation uses different variables in the state, we introduce submatrices for each operation. The subblock $L_{i,3}$ corresponds to the linear function. For the subblock $T_{k,i,3}$ that describes a XOR operation the position of the entries are displayed in Figure 4.4. There are eight different subblocks for the eight different XOR operations. For the k-th XOR operation of the i-th round, $m_{k,i,3}$ and $n_{k,i,3}$ describe the positions of the entries for the two input variables and $o_{k,i,3}$ the position for the output variable entry. Until the 15th round, there is not a specific structure that repeats itself. This is because only after 16 rounds every variable from the state in the first round will be replaced. Thus, the width is $10i + 34$ for the first 15 rounds. Beginning from the 16th round, one can see a structure. The blocks then have a fixed size of 142 columns and they form a diagonal with blocks that overlap, since they are not independent from each other. Then, for every round with $i \geq 16$ there is an empty matrix $\mathbf{0}^{37\times 43+10(i-16)}$ before the block $B_{i,3}$. Because of this, the positions of the entries are also dependent of the variable z_i which has the value $z_i = 0$ for $i \leq 15$ and the value $z_i = 43 + 10(i - 16)$ for $i \geq 16$. For every round, right after a $B_{i,3}$-block follows an empty matrix $\mathbf{0}^{37\times 10(r-i)}$.

Block C

The C-block consists of one row. This row describes the constraint that ensures that at least one s-box is active. It is split into two blocks, C_1 and $C_{2,3}$ and is described in Corollary 4.4. The block C_1 consists of one entry, and has the value -1. This entry corresponds to the column that describes the constant $l = 1$ so that the value in the equation is $-1 \cdot 1$. The sum of all input variables of the s-boxes has to be therefore greater than or equal to one. The $C_{2,3}$-block consists of all variables that are the input to an s-box and has dimensions $1 \times 34 + 10r$. In Figure 4.5, the positions of the entries for the $C_{2,3}$-block are noted.

4.2.2 Linear Cryptanalysis Stair-Fold MILP for Enocoro-128v2

As determined in Chapter 3.2, the linear cryptanalysis MILP is similar to the MILP for differential cryptanalysis. The difference lies in the XOR operations that do not have to be analyzed separately. Instead, the three forked branches have to be considered, but they behave like XOR operations. Again, the matrix will be described in a way so that it can be generated without having to go through every operation. The matrix is depicted in Theorem 4.1 with the variables q, p set to $q = 0$ and $p = 1$.

Block A

The A-blocks describe the part of the d-variables for the corresponding constraint. They all have to take binary values. The description of an $A_{i,1}$-block can be found in Corollary 4.2. Since nine operations are considered in one round, we can split the blocks into nine submatrices. The submatrices consist of the d-variable part of the long constraint and corresponding short constraints. The matrix F describes the d-variables for the linear function and R the d-variables for the three forked branch. An A-block for linear cryptanalysis $A_{i,1}$ has the dimensions 37×9.

Block B

The B-blocks also have a height of 37, but differ in their width. The variables corresponding to the first 34 columns of the $B_{1,1}$-block have to take binary values, and the rest of the variables corresponding to a $B_{i,1}$-block are free. After a $B_{i,1}$-block follows an empty matrix $\mathbf{0}^{37\times 10(r-i)}$. Because the variables from the first round are used until the 12th round, the blocks do not have the same structure until round 13. There are subblocks for the nine operations in one round, depicted in Corollary 4.3. The subblock for the linear function $L_{i,1}$ has a similar structure as in differential cryptanalysis, but the structure for the first and the rest of the rounds differ. The first two non-empty columns of $L_{i,1}$ describe the input

$$m_{1,i,1} = \begin{cases} 33 - i, & \text{if } i <= 2 \\ 44 + 10 \cdot (i - 3) - z_i, & \text{otherwise} \end{cases}$$

$$n_{1,i,1} = \begin{cases} 33, & \text{if } i <= 1 \\ 16 + 10i - z_i, & \text{otherwise} \end{cases}$$

$$o_{1,i,1} = 25 + 10i - z_i$$

a. first three forked branch

$$m_{2,i,1} = \begin{cases} 4 - i, & \text{if } i <= 3 \\ 36 - i, & \text{if } i = 4, 5 \\ 44 + 10 \cdot (i - 6) - z_i, & \text{otherwise} \end{cases}$$

$$n_{2,i,1} = 25 + 10i - z_i$$

$$o_{2,i,1} = 28 + 10i - z_i$$

b. second three forked branch

$$m_{3,i,1} = \begin{cases} 8 - i, & \text{if } i <= 4 \\ 38 + 10 \cdot (i - 5) - z_i, & \text{otherwise} \end{cases}$$

$$n_{3,i,1} = 28 + 10i - z_i$$

$$o_{3,i,1} = 29 + 10i - z_i$$

c. third three forked branch

$$m_{4,i,1} = \begin{cases} 8, & \text{if } i = 1 \\ 37 + 10 \cdot (i - 2) - z_i, & \text{otherwise} \end{cases}$$

$$n_{4,i,1} = \begin{cases} 34, & \text{if } i = 1 \\ 39 + 10 \cdot (i - 2) - z_i, & \text{otherwise} \end{cases}$$

$$o_{4,i,1} = 30 + 10i - z_i$$

d. fourth three forked branch

$$m_{5,i,1} = \begin{cases} 17 - i, & \text{if } i <= 8 \\ 40 + 10 \cdot (i - 9) - z_i, & \text{otherwise} \end{cases}$$

$$n_{5,i,1} = 30 + 10i - z_i$$

$$o_{5,i,1} = 31 + 10i - z_i$$

e. fifth three forked branch

$$m_{6,i,1} = \begin{cases} 17, & \text{if } i = 1 \\ 41 + 10 \cdot (i - 2) - z_i, & \text{otherwise} \end{cases}$$

$$n_{6,i,1} = 36 + 10 \cdot (i - 1) - z_i$$

$$o_{6,i,1} = 32 + 10i - z_i$$

f. sixth three forked branch

$$m_{7,i,1} = \begin{cases} 30 - i, & \text{if } i <= 12 \\ 42 + 10 \cdot (i - 13) - z_i, & \text{otherwise} \end{cases}$$

$$n_{7,i,1} = 42 + 10 \cdot (i - 1) - z_i$$

$$o_{7,i,1} = 33 + 10i - z_i$$

g. seventh three forked branch

$$m_{8,i,1} = \begin{cases} 30, & \text{if } i = 1 \\ 43 + 10 \cdot (i - 1) - z_i, & \text{otherwise} \end{cases}$$

$$n_{8,i,1} = 37 + 10 \cdot (i - 1) - z_i$$

$$o_{8,i,1} = 34 + 10i - z_i$$

h. eighth three forked branch

■ Figure 4.6. Positions of entries for each three forked branch in round i for $T_{k,i,1}$, subblock of $B_{i,1}$

$$d_{i,1} = 35 + 10 \cdot (i - 1) \qquad e_{i,1} = \begin{cases} 34, & \text{if } i = 1 \\ 37 + 10 \cdot (i - 2), & \text{otherwise} \end{cases}$$

$$f_{i,1} = 36 + 10 \cdot (i - 1) \qquad g_{i,1} = 37 + 10 \cdot (i - 1)$$

■ Figure 4.7. Positions of entries in the $C_{2,1}$-block for linear cryptanalysis

for the linear function and the last two describe the output variables. The subblocks for the three forked branches are again described by the position of their entries in Figure 4.6. Since the three forked branch has one in- and two outputs, the first column in $T_{k,i,1}$ describes the input variable at position $m_{k,i,1}$ and the last two columns describe the outputs in position $n_{k,i,1}$ and $o_{k,i,1}$. Up until the 12th round, the blocks have a width of $10i + 34$. From the 13th round, they have a fixed width of 123 and an empty matrix $\mathbf{0}^{37\times 41+10(i-13)}$ before them. Because of this, the positions of the entries are dependent of the variable z_i which has the value $z_i = 0$ for $i \leq 12$ and the value $z_i = 41 + 10(i - 13)$ for $i \geq 13$.

Block C

The C-block describes the constraint that ensures that at least one s-box is active. Therefore, it consists of one row. As one can see in Corollary 4.4, it is split into two blocks. The C_1-block consists of one entry that corresponds the constant $l = 1$ and has the value -1. The $C_{2,1}$-block has an entry for every variable that is at some point the input to an s-box in the cipher. It has the dimensions $1\times 34+10r$. The Figure 4.7 describes the position of the four variables that are the input to an s-box in each round.

This description of the linear and differential cryptanalysis MILPs makes it easier to generate them, as it does not require a deep understanding of the cipher. The MILPs take the novel stair-fold structure, introduced in Chapter 1. If an efficient algorithm for solving stair-fold MILPs exists and is found, then the result could be computed much faster than with an MILP solver.

4.3 Linear and Differential Cryptanalysis 4-Block MILP for AES

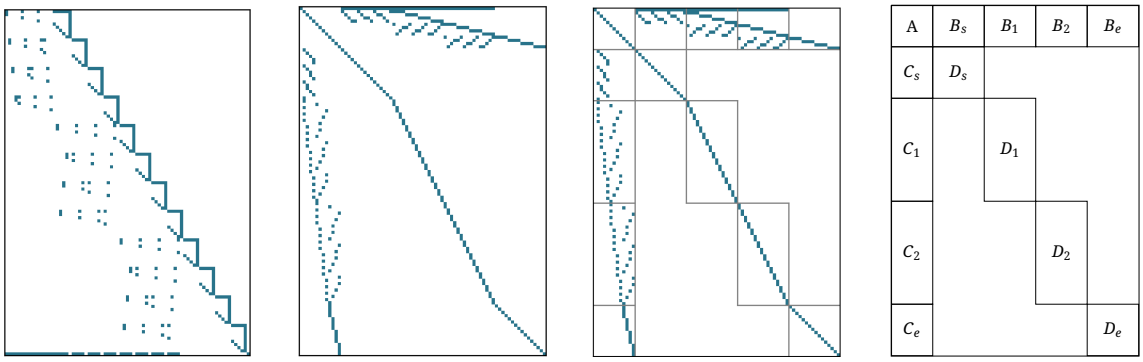
While analyzing the AES MILP, the framework found a 4-block structure. Based on this structure, the following description can generate and visualize the matrix without knowing the underlying principles of the cipher. To understand how the matrix takes the 4-block structure, consider Figure 4.8. This example depicts the constraint matrix for three rounds. The freshly generated matrix with no sorting function applied is shown in Figure 4.8a. Using the sorting functions from the framework, the matrix takes the structure in Figure 4.8b. To see how this corresponds to our 4-block, the corresponding blocks are added in Figure 4.8c. Furthermore, we can see in Figure 4.8d the individual blocks that construct the structure. The 4-block MILP for r rounds is depicted in Theorem 4.5. The matrix is a $36r + 1 \times 16 + 20r + 1$ matrix. The 4-block structure contains the block types A, B, C, D.

▶ Theorem 4.5 (4-block MILP for AES). *Let r be the number of rounds. The linear and differential cryptanalysis MILP for AES has 4-block structure, i. e. we have*

$$\overbrace{}^{\text{binary variables}}\overbrace{\phantom{B_1\cdots B_{r-1}B_e}}^{\text{free variables}}$$

$$\begin{pmatrix} A & B_s & B_1 & \cdots & B_{r-1} & B_e \\ C_s & D_s & & & & \\ C_1 & & D_1 & & & \\ \vdots & & & \ddots & & \\ C_{r-1} & & & & D_{r-1} & \\ C_e & & & & & D_e \end{pmatrix} \cdot \begin{pmatrix} 1 \\ d_0 \\ \vdots \\ d_{4r-1} \\ x_0 \\ \vdots \\ x_{16(r+1)-1} \end{pmatrix} \geq \mathbf{0},$$

where the constraint matrix has dimension $36r + 1 \times 20r + 17$ and $A \in \mathbb{Z}^{4r+1\times 4r+1}$, $B_s, B_e, B_i \in \mathbb{Z}^{4r+1\times 16}$ for $i = \{1, \ldots, r\}$, $C_s, C_e \in \mathbb{Z}^{16\times 4r+1}$, $D_s, D_e \in \mathbb{Z}^{16\times 16}$ and $C_i \in \mathbb{Z}^{32\times 4r+1}$, $D_i \in \mathbb{Z}^{32\times 16}$ for $i = \{1, \ldots, r-1\}$. Further, we have $d_k \in \{0, 1\}$ for $k = \{0, \ldots, 4r-1\}, x_j \in \{0, 1\}$ for $j = \{0, \ldots, 15\}$ and $x_i \in \mathbb{R}$ for $i = \{16, \ldots, 16(r+1)-1\}$.



a. native Matrix b. sorted Matrix c. 4-block structure d. formal 4-block

■ Figure 4.8. Constraint matrix for the linear and differential cryptanalysis MILP for three rounds in the cipher AES. Every non-white entry is a nonzero entry in the matrix.

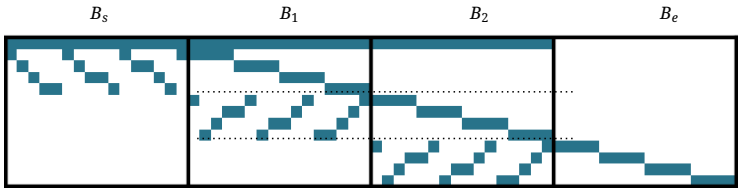
Block A

The block A has dimensions $4r + 1 \times 4r + 1$. It contains a simple diagonal. The value in the first row is -1. It stands for the constraint that ensures that at least one s-box is active: The sum of the input of the s-boxes minus one has to be greater equal 0. The column corresponds to the constant $l = 1$ in the solution vector. The remaining values in the diagonal are -5, because they describe the branch number. Each one of the rows in the A-block except for the first describe the linear function, the *MixColumns()* operation in the cipher. Since there are four linear functions executed in every round, there are $4r$ rows +1 row for the other constraint. These columns correspond to the d-variables. Since there are four new d-variables in every round, the A-block has $4r + 1$ columns. The d-variables have to take binary values. For an entry y of A holds $y \in \{-5, -1\}$. The A-block has the following form

$$A = \begin{pmatrix} -1 & & & \\ & -5 & & \\ & & \ddots & \\ & & & -5 \end{pmatrix}.$$

Block B

The B-blocks all have the same dimensions and their height is the same as the A-block, thus $4r + 1$. The width is not dependent from the number of rounds, as it is always 16. There are three different types of these $4r + 1 \times 16$ B-blocks: $B_s, B_i \, \forall i \in \{1, ..., r - 1\}$ and B_e. For an entry y of B_s, B_e, B_i holds $y \in \{0, 1\}$ for $i \in \{1, ..., r - 1\}$. The columns from the block B_s describe the first variables that are in the state array from the start. All of them have to take a binary value. The first row contains the value 1 for every column. This is because every variable is an input to the s-box, and the first constraint contains every input of the s-boxes. The second to the fifth row describe the input to the linear function. The last $4(r - 1)$ rows are empty. This is due to the constraints which are built in the other rounds, where the variables from the first state array are already discarded and therefore not being used again. The block B_e is the last B-block. It describes the variables that are created in the last round, which can take any value. These variables are not the input to another round, thus they only occur in one constraint. Since they are also not the input to an s-box, they are not contained in the constraint in the first row. Because of this, there are only four non-empty rows in this block.
The Blocks B_s and B_e look like the following



■ Figure 4.9. Visualization of the B-block for three rounds. In the dotted lines, one can see the four long constraints belonging to a round. In Block B_1 are the 'old' variables from the last round and in B_2 we can see the 'new' constraints that are generated in this round.

$$B_s = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & & & & 1 & & & & 1 & & & & 1 & & & \\ & 1 & & & & 1 & & & & 1 & 1 & & & 1 & & \\ & & 1 & & & & 1 & 1 & & & & 1 & & & 1 & \\ & & & 1 & 1 & & & & 1 & & & & 1 & & & \\ & & & & & & & & & & & & & & & \\ & & & & & & & \mathbf{0}^{4(r-1)\times 16} & & & & & & & & & \end{pmatrix},$$

$$B_e = \begin{pmatrix} & & & & & & & \mathbf{0}^{4(r-1)+1\times 16} & & & & & & & & \\ 1 & 1 & 1 & 1 & & & & & & & & & & & & \\ & & & & 1 & 1 & 1 & 1 & & & & & & & & \\ & & & & & & & & 1 & 1 & 1 & 1 & & & & \\ & & & & & & & & & & & & 1 & 1 & 1 & 1 \end{pmatrix}.$$

The blocks B_i also have a continuous first row of value 1, because the columns of these blocks describe the variables that are created in round 1 to round $r-1$. This means that every variable is also the input to an s-box. Thus, they are contained in the constraint that ensures that at least one s-box is active. Additionally, the variables can take any value, they are not restricted. All remaining rows except for eight rows are empty. The position of these eight non-empty rows depends on the i. Each variable of the B_i-blocks is contained in two long constraints. The first constraint a variable is contained, is where the variable is newly generated along with other three new variables. The second constraint where the variables are contained is when they are the input to the next round, specifically the *MixColumns()* operation. When looking at the block B_{i-1} next to a B_i-block, on the same height as the lower four non-empty constraints the new variables that result from the input of the old variables in B_i can be found. This can be seen in Figure 4.9. The B_i-blocks have the following form

$$B_i = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ & & & & & & & \mathbf{0}^{4(i-1)\times 16} & & & & & & & & \\ 1 & 1 & 1 & 1 & & & & & & & & & & & & \\ & & & & 1 & 1 & 1 & 1 & & & & & & & & \\ & & & & & & & & 1 & 1 & 1 & 1 & & & & \\ & & & & & & & & & & & & 1 & 1 & 1 & 1 \\ 1 & & & & 1 & & & & 1 & & & & 1 & & & \\ & 1 & & & & 1 & & & & 1 & & & & 1 & & \\ & & 1 & & & & 1 & & & & 1 & & & & 1 & \\ & & & 1 & & & & 1 & & & & 1 & 1 & & & \\ & & & & & & & & & & & & & & & \\ & & & & & & & \mathbf{0}^{4(r-i-1)\times 16} & & & & & & & & \end{pmatrix}.$$

Block C

The lower part of the matrix is made out of C- and D-blocks. Each row describes a short constraint where the d-variable has to be greater than or equal to the x-variable. Since each long constraint has eight corresponding short constraints, the lower part has eight times more rows than the upper part minus the first constraint. The entries describing the d-variables are in the C-blocks, and the entries describing the x-variables are in the D-blocks. The C-blocks all have the same width $4r + 1$ and for an entry y of $C_s, C_e, C_1, ..., C_{r-1}$ holds $y \in \{0, 1\}$. Just like in the A-block, the first column describes the constant l and the rest of the columns describe the d-variables. The number of rows is fixed. The first and last block have 16 rows, the remaining blocks have 32. The C_s and the C_e-block with the size

$16 \times 4r + 1$ have this form

$$C_s = \left(\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \mathbf{0}^{16\times 4(r-1)} \right), \qquad C_e = \left(\mathbf{0}^{16\times 4(r-1)+1} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \right).$$

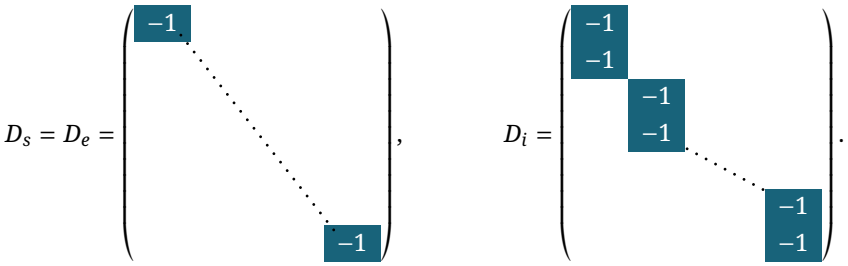
The rest of the C-blocks have a height of 32 rows. The reason for the different heights of the C-blocks is that the first and last 16 x-variables are contained in only one *MixColumns()* operation. Thus, each one of these variables is contained in only one short constraint. The rest of the x-variables are in two long constraints, one for when being newly generated and one belonging to the input of the new round. Thus, the C_i-blocks consist of 32 rows instead of one. The C_i-blocks have an empty column at the beginning. Then $4 \cdot (i-1)$ empty columns follow. The next eight columns contain the values for the d-variables. The last $4 \cdot (r-i-1)$ columns are also empty. Their block size is always $32 \times 4r + 1$ and they look like

$$C_i = \left(\mathbf{0}^{32\times 4(i-1)+1} \begin{array}{c} \\ \end{array} \mathbf{0}^{32\times 4(r-i-1)} \right).$$

Block D

The D-blocks form a diagonal in the matrix. They describe again a short constraint, but the part with the x-variable. The nonzero values are all -1, because the d-variable has to be greater than or equal to the x-variable. Thus, $d \geq x$, which then was transformed to $d - x \geq 0$. For an entry y of $D_s, D_e, D_1, ..., D_{r-1}$ holds $y \in \{-1, 0\}$. Their height is either 16 (the first and last block) or 32. Each block has a width of 16. This means, the block size is not dependent on the number of rounds. There are $r + 1$ D-blocks. The D_s and D_e-blocks are the exact same. This is again because the x-variables corresponding to the columns are contained in one *MixColumns()* operation, not two. The size of D_s and D_e is 16×16. The D_i-blocks have double the rows of the D_s and D_e-blocks, since the corresponding x-variables are contained in two long constraints. The size of a D_i-block is 32×16. The D-blocks have

the following form



The D-blocks are all independent from each other. Thus, the matrix has a 4-block structure where the C- and D-blocks have two different sizes. Alternatively one could fill the rows of the C_s, C_e, D_s, D_e-blocks with 16 empty rows. Then every C and D-block would have the same height of 32 rows.

With this description, knowledge about the underlying principles of the cipher is not needed, the matrix and its structure can be generated nevertheless. The analysis revealed a 4-block structure, which is the first real-world example of this theoretical construct. In case theoretical research results in an efficient algorithm for 4-block, whose existence is yet to determine, then computing the minimum number of active s-boxes for AES might be much faster than using a general MILP solver.

4.4 Open Problems

By using the descriptions outlined in this chapter to create MILPs, it is possible to construct the matrix more efficiently and quickly. Researchers who analyze the constraint matrix would not need to go through the cipher and every single operation, but can just use these specifications. Through our analysis the linear and differential cryptanalysis security bound MILP formulation for Enocoro-128v2 takes the novel stair-fold structure. This structure may be applicable to other MILPs for different ciphers. Then, the stair-fold structure itself could be of interest in theoretical research, and an efficient algorithm could be developed for it in the future. For the MILP that calculates the security bound for the cipher AES for linear and differential cryptanalysis, we found a 4-block structure. Even though an algorithm to solve a 4-block has yet to be found, the block-structure in general is widely researched and this is the first native occurrence of it. Since Mouha et al. used MILPs for searching bounds for the first time, the method became widely used. Some examples are: Baksi [1] calculates bounds on the GIFT-128 cipher using the convex hull method to derive the constraints. Boura et al. [9] present new ways to show the behavior of the s-box with fewer inequalities in the constraint matrix. Zhou et al. [35] analyze the problem using a divide-and-conquer approach. Using the framework, one could generate the corresponding MILPs and analyze them. Perhaps even more new real-life structures for MILPs can be found such that it becomes more efficient to use specific MILP algorithms instead of using arbitrary MILP solver such as Gurobi.

Bibliography

[1] Anubhab Baksi. “New Insights on Differential and Linear Bounds Using Mixed Integer Linear Programming”. In: *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. Singapore: Springer Singapore, 2022, pp. 109–140. ISBN: 978-981-16-6522-6. DOI: 10.1007/978-981-16-6522-6_5. URL: https://doi.org/10.1007/978-981-16-6522-6_5.

[2] Eli Biham. “Differential Cryptanalysis”. In: *Encyclopedia of Cryptography and Security, 2nd Ed.* Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Springer, 2011, pp. 332–336. DOI: 10.1007/978-1-4419-5906-5_572. URL: https://doi.org/10.1007/978-1-4419-5906-5%5C_572.

[3] Eli Biham. “On Matsui’s Linear Cryptanalysis”. In: *Advances in Cryptology - EUROCRYPT ’94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*. Ed. by Alfredo De Santis. Vol. 950. Lecture Notes in Computer Science. Springer, 1994, pp. 341–355. DOI: 10.1007/BFb0053449. URL: https://doi.org/10.1007/BFb0053449.

[4] Eli Biham and Adi Shamir. “Differential Cryptanalysis of DES-like Cryptosystems”. In: *Advances in Cryptology - CRYPTO ’90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by Alfred Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1. URL: https://doi.org/10.1007/3-540-38424-3%5C_1.

[5] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993. ISBN: 978-1-4613-9316-0. DOI: 10.1007/978-1-4613-9314-6. URL: https://doi.org/10.1007/978-1-4613-9314-6.

[6] Andrey Bogdanov. “Analysis and design of block cipher constructions”. PhD thesis. Ruhr University Bochum, 2010. ISBN: 978-3-89966-354-9. URL: https://d-nb.info/1003338747.

[7] Andrey Bogdanov. “On unbalanced Feistel networks with contracting MDS diffusion”. In: *Des. Codes Cryptogr.* 59.1-3 (2011), pp. 35–58. DOI: 10.1007/s10623-010-9462-0. URL: https://doi.org/10.1007/s10623-010-9462-0.

[8] Charles Bouillaguet, Pierre-Alain Fouque, and Gaëtan Leurent. “Security Analysis of SIMD”. In: *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. Lecture Notes in Computer Science. Springer, 2010, pp. 351–368. DOI: 10.1007/978-3-642-19574-7_24. URL: https://doi.org/10.1007/978-3-642-19574-7%5C_24.

[9] Daniel Coggia and Christina Boura. “Efficient MILP Modelings for Sboxes and Linear Layers of SPN ciphers”. In: *IACR Transactions on Symmetric Cryptology* 2020, Issue 3 (2020), pp. 327–361. DOI: 10.13154/tosc.v2020.i3.327-361. URL: https://tosc.iacr.org/index.php/ToSC/article/view/8705.

[10] Don Coppersmith. “The Data Encryption Standard (DES) and its strength against attacks”. In: *IBM J. Res. Dev.* 38.3 (1994), pp. 243–250. DOI: 10.1147/rd.383.0243. URL: https://doi.org/10.1147/rd.383.0243.

[11] CRYPTREC. *Cryptographic Technology Guideline (Lightweight Cryptography)*. https://www.cryptrec.go.jp/report/cryptrec-gl-2003-2016en.pdf. Accessed: 2022-12-07.

[12] Jana Cslovjecsek, Friedrich Eisenbrand, Michal Pilipczuk, Moritz Venzin, and Robert Weismantel. “Efficient Sequential and Parallel Algorithms for Multistage Stochastic Integer Programming Using Proximity”. In: *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*. Ed. by Petra Mutzel, Rasmus Pagh, and Grzegorz Herman. Vol. 204. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 33:1–33:14. DOI: 10.4230/LIPIcs.ESA.2021.33. URL: https://doi.org/10.4230/LIPIcs.ESA.2021.33.

[13] Jana Cslovjecsek, Friedrich Eisenbrand, and Robert Weismantel. “N-fold integer programming via LP rounding”. In: *CoRR* abs/2002.07745 (2020). arXiv: 2002.07745. URL: https://arxiv.org/abs/2002.07745.

[14] Joan Daemen and Vincent Rijmen. “Rijndael”. In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 1046–1049. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_611. URL: https://doi.org/10.1007/978-1-4419-5906-5_611.

[15] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4. URL: https://doi.org/10.1007/978-3-662-04722-4.

[16] Joan Daemen and Vincent Rijmen. “The Wide Trail Design Strategy”. In: *Proceedings of the 8th IMA International Conference on Cryptography and Coding*. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 222–238. ISBN: 3540430261.

[17] Universität Graz. *Wortschätze*. https://wortschaetze.uni-graz.at/de/wortschaetze/schrift/belegdatenbank/z/ziffer/. Accessed: 2022-11-14.

[18] Martin Hell and Thomas Johansson. *Security Evaluation of Stream Cipher Enocoro-128v2*. English. CRYPTREC Technical Report, 2010.

[19] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. “n-Fold integer programming in cubic time”. In: *Math. Program.* 137.1-2 (2013), pp. 325–341. DOI: 10.1007/s10107-011-0490-y. URL: https://doi.org/10.1007/s10107-011-0490-y.

[20] Howard M. Heys. “A Tutorial on Linear and Differential Cryptanalysis”. In: *Cryptologia* 26.3 (2002), pp. 189–221. DOI: 10.1080/0161-110291890885. URL: https://doi.org/10.1080/0161-110291890885.

[21] Hitachi. *Enocoro-128v2 specifications*. https://www.hitachi.com/rd/yrl/crypto/enocoro/index.html. Accessed: 2022-12-20.

[22] Klaus Jansen, Kim-Manuel Klein, and Alexandra Lassota. “The Double Exponential Runtime is Tight for 2-Stage Stochastic ILPs”. In: *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*. Ed. by Mohit Singh and David P. Williamson. Vol. 12707. Lecture Notes in Computer Science. Springer, 2021, pp. 297–310. DOI: 10.1007/978-3-030-73879-2_21. URL: https://doi.org/10.1007/978-3-030-73879-2%5C_21.

[23] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. “Near-Linear Time Algorithm for n-Fold ILPs via Color Coding”. In: *SIAM J. Discret. Math.* 34.4 (2020), pp. 2282–2299. DOI: 10.1137/19M1303873. URL: https://doi.org/10.1137/19M1303873.

[24] Masayuki Kanda. “Practical Security Evaluation against Differential and Linear Cryptanalyses for Feistel Ciphers with SPN Round Function”. In: *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings*. Ed. by Douglas R. Stinson and Stafford E. Tavares. Vol. 2012. Lecture Notes in Computer Science. Springer, 2000, pp. 324–338. DOI: 10.1007/3-540-44983-3_24. URL: https://doi.org/10.1007/3-540-44983-3%5C_24.

[25] Kim-Manuel Klein. “About the Complexity of Two-Stage Stochastic IPs”. In: *Integer Programming and Combinatorial Optimization - 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*. Ed. by Daniel Bienstock and Giacomo Zambelli. Vol. 12125. Lecture Notes in Computer Science. Springer, 2020, pp. 252–265. DOI: 10.1007/978-3-030-45771-6_20. URL: https://doi.org/10.1007/978-3-030-45771-6%5C_20.

[26] Kim-Manuel Klein and Janina Reuter. “Collapsing the Tower - On the Complexity of Multistage Stochastic IPs”. In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*. Ed. by Joseph (Seffi) Naor and Niv Buchbinder. SIAM, 2022, pp. 348–358. DOI: 10.1137/1.9781611977073.17. URL: https://doi.org/10.1137/1.9781611977073.17.

[27] Mitsuru Matsui. “Linear Cryptanalysis Method for DES Cipher”. In: *Advances in Cryptology - EUROCRYPT ’93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, 1993, pp. 386–397. DOI: 10.1007/3-540-48285-7_33. URL: https://doi.org/10.1007/3-540-48285-7%5C_33.

[28] Mitsuru Matsui and Atsuhiro Yamagishi. “A New Method for Known Plaintext Attack of FEAL Cipher”. In: *Advances in Cryptology - EUROCRYPT ’92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*. Ed. by Rainer A. Rueppel. Vol. 658. Lecture Notes in Computer Science. Springer, 1992, pp. 81–91. DOI: 10.1007/3-540-47555-9_7. URL: https://doi.org/10.1007/3-540-47555-9%5C_7.

[29] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. “Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming”. In: *Information Security and Cryptology*. Ed. by Chuan-Kun Wu, Moti Yung, and Dongdai Lin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 57–76. ISBN: 978-3-642-34704-7.

[30] NIST. *Announcing Development of a Federal Information Processing Standard for Advanced Encryption Standard*. https://csrc.nist.gov/news/1997/announcing-development-of-fips-for-advanced-encryp. Accessed: 2022-11-14.

[31] NIST. *Commerce Department Announces Winner of Global Information Security Competition*. https://www.nist.gov/news-events/news/2000/10/commerce-department-announces-winner-global-information-security. Accessed: 2022-11-14.

[32] Naoki Shibayama and Yasutaka Igarashi. “A New Higher Order Differential of Enocoro-128v2”. In: *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*. 2021, pp. 379–384. DOI: 10.1109/CANDARW53999.2021.00070.

[33] Kyoji Shibutani. “On the Diffusion of Generalized Feistel Structures Regarding Differential and Linear Cryptanalysis”. In: *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. Lecture Notes in Computer Science. Springer, 2010, pp. 211–228. DOI: 10.1007/978-3-642-19574-7_15. URL: https://doi.org/10.1007/978-3-642-19574-7%5C_15.

[34] National Security Agency USA. *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*. https://web.archive.org/web/20101106122007/http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf. Accessed: 2022-11-14.

[35] Chunning Zhou, Wentao Zhang, Tianyou Ding, and Zejun Xiang. *Improving the MILP-based Security Evaluation Algorithm against Differential/Linear Cryptanalysis Using A Divide-and-Conquer Approach*. Cryptology ePrint Archive, Paper 2019/019. https://eprint.iacr.org/2019/019. 2019. URL: https://eprint.iacr.org/2019/019.