Analyzing MILP Security Bound Modeling for Linear and Differential Cryptanalysis

Lars Huth

July 2023
Master Thesis

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

Advised by
Leo Wennmann

Hiermit erkläre ich, Lars Huth, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Ich bestätige außerdem, dass die vorliegende Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen meiner vorliegenden Arbeit mit der digital eingereichten elektronischen Version meiner Arbeit übereinstimmen.

Frankfurt am Main, den

Zusammenfassung

Die Verwendung von Mixed-Integer Linearer Programmierung (MILP) in Berechnungen der Sicherheit von Verschlüsselungsverfahren gegen lineare und differentielle Kryptanalyse hat seit der Einführung durch Mouha et al. [24] deutlich an Popularität gewonnen. Neue, sowohl mehr effektive als auch mehr effiziente, Modelierungsansätze sind hierbei über die letzten Jahre entwickelt worden. Ende 2022 fand Lohr [20] in einer Analyse von MILP Constraint Matrizen für AES und Enocoro-128v2 verschiedene Block Strukturen. Gegeben dieser Resulte erweitern wir das für die Analyse aufgestellte Framework, mittels der Implementierung von sechs neuen Modelierungsansätzen und zwei Verschlüsselungsverfahren, um ein vielfaches. Die meisten der hier implementierten Modelierungsansätze konzentrieren sich hierbei auf die Modellierung von S-boxen und den Übergängen derer Input Variablen zu Output Variablen, was zuvor nicht beachtet wurde. Mittels Nutzung des Frameworks analysieren wir die MILP Constraint Matrizen für alle von uns implementierten MOdellierungsansätze der differentiellen Kryptanalyse auf dem LBlock Verschlüsselungsverfahren. Hierbei fanden wir das alle Modelierungen des Verschlüsselungsverfahrens Block Strukturen beinhalten, nämlich n-fold und two-stage.
Unsere Aufteilung der MILPs in Blöcke erlaubt die Konstruktion von Constraint Matrizen für höhere Rundenzahlen des Verschlüsselungsverfahren ohne die Constraints einzeln Berechnen zu müssen. Zudem ist der Fakt, das man Sturkturen finden kann, nutzbar um den Prozess der Berechnung einer Lösung eines MILP effizienter zu gestalten.

Abstract

The usage of Mixed-Integer Linear Programming in the calculation of cipher's security bounds against differential and linear cryptanalysis has, since its introduction by Mouha et al. [24], gained a lot of popularity. New, both more effective and more efficient, techniques for modeling these security bounds have been introduced over the last years. At the end of last year, Lohr [20] analyzed the initial modeling's constraint matrix in a search for underlying block structures. For the ciphers AES and Enocoro-128v2, instances of 4-block and stair-fold were found. Inspired by those results, we extend the existing ASEL framework by adding six more approaches to modeling the cryptanalyses as well as the ciphers LBlock and Gift64. Most of the approaches we implemented, focus on the exclusion of specific input-output transitions, i.e. impossible transitions, for S-boxes in a bit-oriented setting. Using the framework, our analyses of different modelings for the differential cryptanalysis security bound for the LBlock cipher find more structures. For all newly implemented modelings, we find the underlying problem to be either an n-fold MILP or a two-stage MILP. Our separation of the constraint matrices into the general structure of blocks offers a less computationally demanding way of constructing constraint matrices for more rounds as well. Furthermore, given the fact that oftentimes the solving processes of constructed MILPs are treated as a black box, finding these structures would allow the usage of more algorithms likely more efficient than commercial solvers such as Gurobi and Cplex, which would reduce the time and resources required to calculate security bounds.

Acknowledgements

First of all, I would like to thank Leo Wennmann for introducing me to this interesting topic and for her patience with me as well as for providing me with all of the encouraging words when I needed them. My thanks also go to Holger Dell for having such a wide range of great projects in the group, allowing me to connect with others whom I would not have connected with otherwise, and learning more about the different intersections theory has to offer. From the people regularly strolling around the TCS group, a special thanks goes to Jonas Strauch who had great input available whenever I asked and whom I could trust to be there to join me after every Thursday lunch break when a coffee was more than necessary. For motivating me to work on the topic and providing a great base to work off, (Aura) Sofia (Elisabetta) Lohr has to be mentioned as well of course.

Lastly, I would like to mention and thank all of the people who either made these last 6 months enjoyable, provided me with moral support, or took a quick look at my thesis helped look for spelling mistakes, namely Ehud Cseresnyes, Emmanuela Georgoula, Marius Hagemann, Melvin Kallmeyer, Sebastian Link, Julian Ockelmann, Jannik Schäfer, Tolga Tel, and last but far from least Jacqueline Ulken.

Contents

1 Introduction

Say you are a brilliant, although most likely stressed, mathematician in the late 1930s working for an intelligence agency. The Germans take the messages used in their war communication and make them unreadable for anyone not in possession of both a certain password and a certain machine. As being brilliant entails working against the Nazis, you are sitting in your office in the Polish Cipher Bureau with an Enigma, one of the machines that are responsible for making all of the German communication unreadable. If you had the password, making the messages once again readable would be easy. However, while you do not have it, your spy colleagues provide you with messages, both in their readable and unreadable form, on a daily basis. With the help of the provided messages, which have been made unreadable using today's password, along with your knowledge of the machine's workings, your goal is to reverse the process and decipher the new German messages. This would allow you to become the first to crack the code. In cryptography, this is a so-called chosen-plaintext attack.

There are different techniques for conducting such chosen-plaintext attacks. One is to find encryption keys by comparing the inputs and outputs of encryption processes (ciphers) in order to find deviations from pure randomness. Whereas a perfect cipher would assign each input to any output with the same probability and no correlations would be due to any causality in the process, real ciphers often exhibit some causal patterns that can be exploited. These deviations give people, who would be interested in reversing an encryption, knowledge about which inputs are most likely to belong to which output and vice versa. Two of the most popular techniques exploiting those irregularities, and compared to those used in the 1930s and 1940s relatively new, are linear and differential cryptanalysis.

Biryukov and De Cannière [6] describe linear cryptanalysis as "probabilistic linear relations between parity bits of the plaintext, the ciphertext, and the secret key". Until Mouha, Wang, Gu, and Preneel [24] in 2011 modeled these probabilistic relations using mixed-integer linear programming (MILP), these often took rigorous calculations by the cryptanalysts themselves. By now, many additions have been made to the original technique with us only being able to consider 6 of them in this work.

1.1 Encryption, Decryption, and Ciphers

We begin by taking a look at what we understand as a cipher. Traditionally, a cipher's purpose is to take a readable string (plaintext) and make it unreadable (ciphertext), with the concept of readability applying to either humans or, in the more recent years of human history, machines. The process of transforming a plaintext into a ciphertext is referred to as encryption and the process of transforming a ciphertext back to its original plaintext is referred to as decryption. In accordance with Stinson [29], we will refer to either just an encryption function e or a pair (e, d) made up of an encryption function e and a decryption function d as a cryptosystem, depending on the process that is being looked at.

The field of cryptanalysis, as we deal with it, is largely motivated by the real-life processes of en- and decrypting data on machines, i.e. long strings of bits. This means, we represent the strings of bits, that make up the inputs and outputs of real ciphers, as vectors over the Galois field $\mathbb{F}_2$ of size 2, its elements usually represented as $\{0, 1\}$. On this field, we can perform all of the bit-wise operations that are done in real machines while staying true to ourselves as theoretical computer science enthusiasts. All ciphers we will consider in this thesis have two inputs, a long string of bits that is to be made unreadable, and a key k which is another string of bits used in how exactly the input is made unreadable.

► Definition 1.1 (Encryption function). *An encryption function $e(x, k) = y$ with $e : \mathbb{F}_2^{n_{in}}, \mathbb{F}_2^{n_{key}} \longrightarrow \mathbb{F}_2^{n_{out}}$ is a deterministic function taking a plaintext x made up of n_{in} bits and a key k made up of n_{key} bits while returning a ciphertext y made up of n_{out} bits.*

While there are processes requiring ciphers, for which inferring the plaintext x from a given ciphertext y is as difficult as possible such as cryptographic hash functions, many require some kind of function for decryption d to be available. This is the case, especially in communication, where in the best case the sender of a message wants the receiver to be able to read said message with little effort, while all others are unable to read the message.

► Definition 1.2 (Decryption function). *Given an encryption function e, a decryption function $d_e(y, k) = x$ with $d : \mathbb{F}_2^{n_{out}}, \mathbb{F}_2^{n_{key}} \longrightarrow \mathbb{F}_2^{n_{in}}$ is a deterministic function taking a ciphertext y made up of n_{out} bits and a key k made up of n_{key} bits while returning a plaintext x made up of n_{in} bits.*

With the need for our ciphers to be reversible, we opt for block ciphers, for which we take the definition from Knudsen and Robshaw [19].

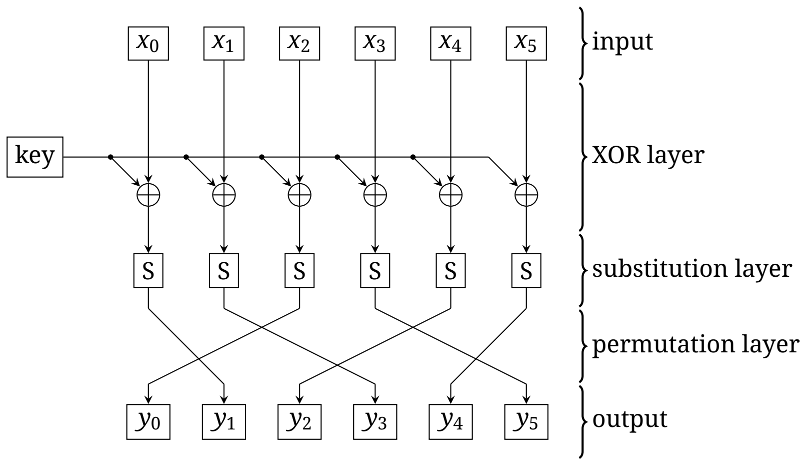
► Definition 1.3 (Block cipher). *A block cipher e_B is characterized by working on blocks of data and being reversible, wherein using a key k of size $|k|$ for an input block of n bits is mapped to an output block of n bits, $e_B : \{0, 1\}^n \times \{0, 1\}^{|k|} \longrightarrow \{0, 1\}^n$ and a decryption function d_B as an inversion of the original cipher exists $d_B = e_B^{-1} : \{0, 1\}^n \times \{0, 1\}^{|k|} \longrightarrow \{0, 1\}^n$.*

Cryptography is a field largely motivated by real life and the construction of most encryption processes follows at least some constraints set by hardware. Many ciphers, such as *LBlock* and the well-known *Rijndael (AES)*, only use operations done on sub-blocks rather than on the full block, with the two mentioned ones first introduced in Wu [35] and Daemen and Rijmen [11] respectively. The size of these sub-blocks is referred to as a cipher's "orientation", e.g. LBlock is a 64-bit block cipher with its sub-blocks being made up of 4 bits each and it, therefore, being referred to as a 4-bit oriented cipher. Some works also differentiate these orientations by classifying them as either bit-oriented, byte-oriented, or word-oriented, in which case LBlock would be referred to as a word-oriented cipher made up of 16 words with each word being made up of 4 bits.

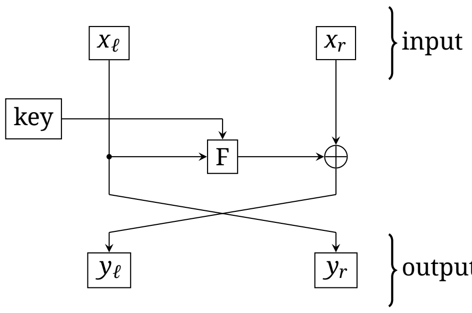
The most used operations on these plaintext sub-blocks are permutations, bit-wise XOR, linear transformation i.e. applying a linear function to the input values, and the usage of substitution-boxes (S-boxes).

► Definition 1.4 (S-box). *A $v \times w$ S-box implements a (possibly nonlinear) function $s : \mathbb{F}_2^v \longrightarrow \mathbb{F}_2^w$ by taking an input of v bits and, using a substitution table, maps said input to an output of w bits.*

S-boxes are oftentimes referenced by a string like `40132567E8A9CDBF`, which tells the reader how bits are mapped. In this case, the 4×4 S-box would map 0 (0000) to 4 (0100), 1 (0001) to 0 (0000), 2 (0010) to 1 (0001), and so on.
Both bijective as well as non-bijective S-boxes exist, however, the ciphers in this work only utilize bijective S-boxes.



■ Figure 1.1. Example of a generic substitution permutation network's round, inspired by the *Gift64* cipher.



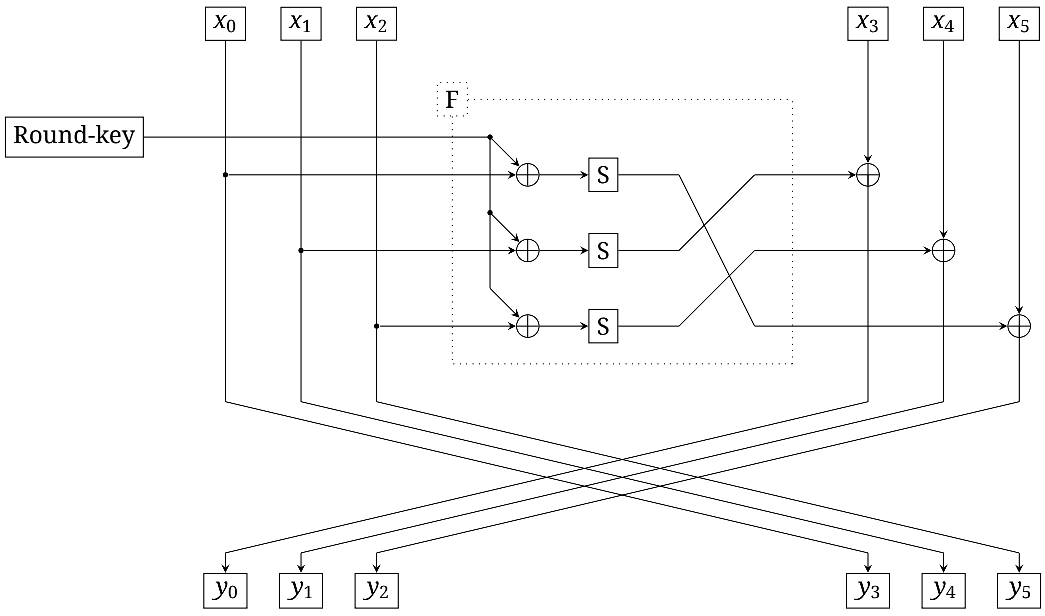
■ Figure 1.2. Structure of a Feistel network's round.

The ciphers we look at in this work are made up of *substitution permutation networks (SPNs)* and *Feistel networks*. In both types, a cipher is defined by a designated design for a round and the number of rounds it runs. For executing multiple rounds, the output y_i of the i-th round functions as the input x_{i+1} for round $i+1$.
Katz and Lindell [17] characterize SPNs as described in the following and illustrated in Figure 1.1.

▸ Definition 1.5 (Substitution Permutation Networks). *For round i of an SPN is made up of 3 distinct steps:*

1. *The round's input x_i is XORed with the round's (sub-)key k_i giving $x_i' := x_i \oplus k_i$,*
2. *x_i' is then funneled through s-boxes $x_i'' := s(x_i')$, and*
3. *with x_i'' then being permuted and resulting in the rounds output $y_i := p(x_i'')$.*

Note that these three steps do not have to happen in this order, e.g. AES has the key addition at the end of the round, with the exception of the first round where it happens both at the beginning and at the end. Some well known ciphers using SPNs are AES, *PRESENT*, and *GIFT*, with the later two first introduced in Bogdanov, Knudsen, Leander, Paar, Poschmann, Robshaw, Seurin, and Vikkelsoe [7] and Banik, Pandey, Peyrin, Sim, Todo, and Sasaki [3] respectively. On the other hand, with Feistel networks, we get invertible ciphers, despite them not necessarily using invertible S-boxes. A Feistel network's structure is most easily explained using Figure 1.2.



■ Figure 1.3. Example of a generic Feistel network's round.

▸ Definition 1.6 (Feistel Network). *For bitstrings a and b, let $a||b$ be the concatenation of those strings and a function $f : \mathbb{F}_2^{n_{in}/2} \times \mathbb{F}_2^{|k|} \longrightarrow \mathbb{F}_2^{n_{in}/2}$.*

1. *The round's input x is split into two, namely x_ℓ and x_r such that $x_\ell||x_r = x$, each of length $n_{in}/2$.*
2. *x_ℓ is put out as y_r.*
3. *x_ℓ is permuted, XORed with a key k, and put through S-boxes in $f_k(x_\ell)$ and then XORed x_r with the results being put out as $y_\ell := f_k(x_\ell) \oplus x_r$.*
4. *This gives us output $y := y_\ell||y_r$.*

At times, the symmetric equivalent is done which results in $y_\ell = x_r$ and $y_r = x_\ell \oplus f_k(x_r)$. In either case, the function f includes XORing with a key, substitutions through S-boxes, and permutation of the bits.

1.2 Cryptanalyses

Now that we know what a cipher is, we should ask ourselves how we know that a cipher is safe, i.e. that not everyone is able to easily decrypt one of our secret messages. Informally speaking, cryptanalysis is the term used for processes that try to decrypt a ciphertext while missing some information such as the key. Be that someone trying to gain access to information or research trying to determine how difficult breaking in actually is. While we only consider differential and linear cryptanalysis, there are more types of cryptanalyses in existence.

As mentioned before, differential and linear cryptanalysis mainly looks into statistical deviations from complete randomness for some relations between inputs and outputs. In our work, these deviations are used to estimate how secure a cipher is. Security measures generally try to find the case in which a ciphertext is the easiest to decrypt. We generally call a cipher secure if, given that all parts work as intended, the probability of the best guess gaining the information we are trying to hold secure is negligible with the definition of negligibility translated from Schnitger [28].

▶ Definition 1.7 (Negligibility). *We call a function $f : \mathbb{N} \to [0,1]$ negligible iff for every $k \in \mathbb{N}$ there exists an $n \in \mathbb{N}$ such that for all $x \geq n$ the inequality $f(x) \leq \frac{1}{x^k}$ holds.*

▶ Definition 1.8 (Secure cipher). *We call a cipher (e, n) with encryption function e and n rounds secure iff the function describing probability of inverting a ciphertext back to $x \in \mathbb{F}_2^{n_{in}}$ or the key $k \in \mathbb{F}_2^{n_{key}}$ with any deterministic or randomized algorithm is negligible.*

This means a cipher is considered secure, under the assumption of $P \neq NP$, if the best educated guess for decrypting a ciphertext has the same chance of success as selecting the correct input by uniformly randomly drawing one from all possible inputs, i.e. no educated guess is better than brute-forcing the solution. We can see that finding the case in which a ciphertext is the easiest to decrypt is the same as finding an upper bound on the probability of decrypting a ciphertext successfully. This work, therefore, deals with us looking at how some of the models, which are created with the aim to find said case in which a ciphertext is the easiest to decrypt, are constructed. Yet, before doing so, we explain the decryption processes for which we are looking at the models.

We begin with taking a look into differential cryptanalysis, while a more comprehensive explanation, including how the key of a cipher can be extracted using the cryptanalysis, can be found in Lohr [20] which this thesis builds on. Biham and Shamir [5] first presented differential cryptanalysis to the scientific community in 1990. Meanwhile, Coppersmith [10] states that cryptanalysts at IBM already knew of the method since 1974 but never published it.

▶ Definition 1.9 (Differential Cryptanalysis, Heys [14]). *Differential cryptanalysis is a chosen plaintext attack, meaning that the attacker is able to select inputs and examine outputs in an attempt to derive the key. For differential cryptanalysis, the attacker will select pairs of inputs, x and y, to satisfy a particular $x \oplus y$, knowing that for that $x \oplus y$ value, a particular $e(x) \oplus e(y)$ value occurs with high probability.*

This means the general concept is to take all pairs of inputs $x_1 = (x_{1,0}, x_{1,1}, x_{1,2}, \ldots x_{1,(n_{in}-1)})$ and $x_2 = (x_{2,0}, x_{2,1}, x_{2,2}, \ldots x_{2,(n-1)})$ and encrypt them to $e(x_1) = y_1 = (y_{1,0}, y_{1,1}, y_{1,2}, \ldots y_{1,(n_{out}-1)})$ and $e(x_2) = y_2 = (y_{2,0}, y_{2,1}, y_{2,2}, \ldots y_{2,(n_{out}-1)})$. Using the inputs' and outputs' bitwise XOR $x = x_1 \oplus x_2 = (x_{1,0} \oplus x_{2,0}, x_{1,1} \oplus x_{2,1}, \ldots, x_{1,(n_{in}-1)} \oplus x_{2,(n_{in}-1)})$ and $y = y_1 \oplus y_2 = (y_{1,0} \oplus y_{2,0}, y_{1,1} \oplus y_{2,1}, \ldots, y_{1,(n_{out}-1)} \oplus y_{2,(n_{out}-1)})$, we can write the relation from input and output differences down in a table, usually referred to as a difference distribution table (DDT).

▶ Definition 1.10 (Difference Distribution Table (DDT)). *A table DDT_S for an S-box S whose rows represent an input difference x, columns represent an output difference y, and entries represent the number of input pairs $(x^{(a)}, x^{(b)})$ such that $x^{(a)} \oplus x^{(b)} = x$ and $S(x^{(a)}) \oplus S(x^{(b)}) = y$.*

As an example, we have the DDT of S-box used in the PRESENT cipher `C56B90AD3EF84712` in Table 1.1. We can define a simple measure of security for an S-box as the least possible amount of diffusion that can be achieved by a pair of input differences and output differences. This is analogous to the DDT's largest entry divided by the total number of possibly resulting output differences. This is known as the maximum differential probability (MDP).

▶ Definition 1.11 (Maximum differential probability, Ohkuma et al. [25]). *Let s be a $v \times w$ S-box. Then the MDP of s is the maximum number of pairs with the same input differences as well as the same output difference*

$$MDP_s = \max_{x,y \neq 0} \frac{|DDT_s(x,y)|}{2^v}.$$

On the other hand, if a specific transition (x, y) is taken in S-box s, then we can bound the security by the diffusion that is achieved by this transition, namely $\frac{|DDT_s(x,y)|}{2^v}$, as seen in Baksi [2].

x diff \ $S(x)$ diff	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
A	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
B	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
C	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
D	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
E	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
F	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

Table 1.1. DDT for $S_{PRESENT}$. Input difference on the y-axis, output difference on the x-axis

The linear counterpart was first introduced in Matsui [21]. A more comprehensive introduction than the one we offer here can again be found in Lohr [20]. While the concept of differential cryptanalysis builds on the difference between inputs being correlated, in linear cryptanalysis we try to find direct correlations between input and output.

► Definition 1.12 (Linear Cryptanalysis). *The general concept is to take all possible inputs $x = (x_0, x_1, x_2, \dots x_{(n_{in})-1})$ and encrypt them to $e(x) = y = (y_0, y_1, y_2, \dots y_{n_{out}-1})$. Now for all possible transitions (x, y), we apply what is called a linear mask $a = (a_0, ..., a_{n_{in}+n_{out}-1}) \in \mathbb{F}_2^{n_{in}+n_{out}}$ such that we can look at whether $a \times (x y)^T = 0$ which should be the case half the time as in a truly random cipher, the sum over a subset of output bits y_a is expected to come to $\frac{|y_a|}{2}$.*

As Heys [14] puts it, "The approach in linear cryptanalysis is to determine expressions of the form $[a \times (x y)^T = 0]$ which have a high or low probability of occurrence. No obvious linearity such as above should hold for all input and output values or the cipher would be trivially weak. If a cipher displays a tendency for a $[a \times (x y)^T = 0]$ to hold with high probability or not hold with high probability, this is evidence of the cipher's poor randomization abilities.". So when looking at an a applied to all $2^{n_{in}}$ possible inputs, we are expecting $a \times (x, e(x))^T$ to equal 0 in $\frac{2^{n_{in}}}{2}$ of those. We can write the relation down in a table, usually referred to as a linear approximation table (LAT), using the deviation from those $\frac{2^{n_{in}}}{2}$.

► Definition 1.13 (Linear Approximation Table (LAT)). *A table LAT_S for an S-box S whose rows represent a linear mask x, columns represent a linear mask y, and entries represent the number of inputs $x^{(a)}$ such that $x^{(a)} \oplus x = S(x^{(a)}) \oplus y$ minus the number of expected inputs fulfiling said constraint $2^{\max(n_{in}, n_{out})}/2$.*

Take, again, the S-box used in the PRESENT cipher `C56B90AD3EF84712` for which we created the LAT in Table 1.2. Analogously to differential cryptanalysis, we can define a simple measure of security of an S-box as the largest possible bias, that can be introduced. We can calculate that by dividing the total number of possibly resulting outputs by the LAT's largest entry. This is known as the maximum linear probability (MLP).

LM on $S(x)$ / LM on x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	4	0	4	0	0	0	0	0	4	0	4
2	0	0	2	2	2	2	0	0	2	2	0	4	0	4	2	2
3	0	0	2	2	2	2	4	0	2	2	4	0	0	0	2	2
4	0	0	2	2	2	2	0	4	2	2	0	4	0	0	2	2
5	0	0	2	2	2	2	0	0	2	2	4	0	4	0	2	2
6	0	0	0	4	0	0	4	0	0	4	0	0	4	0	0	0
7	0	0	0	4	4	0	0	0	0	4	0	0	0	0	4	0
8	0	0	2	2	0	0	2	2	2	2	0	0	2	2	4	4
9	0	4	2	2	0	0	2	2	2	2	4	0	2	2	0	0
A	0	0	4	0	2	2	2	2	0	0	0	4	2	2	2	2
B	0	4	0	0	2	2	2	2	4	0	0	0	2	2	2	2
C	0	0	0	0	2	2	2	2	4	0	0	4	2	2	2	2
D	0	4	4	0	2	2	2	2	0	0	0	0	2	2	2	2
E	0	0	2	2	4	4	2	2	2	2	0	0	2	2	0	0
F	0	4	2	2	0	0	2	2	2	2	4	0	2	2	0	0

Table 1.2. LAT for $S_{PRESENT}$. Input linear mask (LM) on the y-axis, output linear mask (LM) on the x-axis

▶ Definition 1.14 (Maximum linear probability, Ohkuma et al. [25]). *Let s be a $v \times w$ S-box. Then the MLP of s is the maximal bias $MLP_s = \max_{x,y\neq 0} \frac{|LAT_s(x,y)|}{2^v}$ that can be introduced to the S-box by any linear mask.*

On the other hand, if a specific transition (x, y) is taken in S-box s, then we can bound the security by the bias that is introduced by this transition, namely $\frac{|LAT_s(x,y)|}{2^v}$.

For a cipher to be valued as secure, the product of the MDPs, MLPs, or specific transition value, whichever has been chosen, must be negligible. From Section 1.1, we know that all of the ciphers considered in this work have a designated design for a round. Since someone looking to break the cipher knows all of the components, a cryptographer's choice for making a cipher more secure is executing enough rounds such that the negligibility is given. Generally, ciphers can be made more secure by increasing the number of rounds they run. That is, given they have not yet reached their maximal level of secureness. However, given that each additional round means a higher computational effort, and therefore costs time, electricity, and maybe even hardware. Therefore, reducing the number of rounds to the minimum required for a cipher to function securely and not being susceptible to known attacks. The research is therefore also driven by an interest to keep those costs low.

1.3 Mixed-Integer Linear Programming

There are different techniques and solvers for calculating a cipher's resistance against differential and linear cryptanalysis but as stated before, this thesis deals with those models based on Mixed-Integer Linear Programs (MILPs). Since the publication of Mouha et al. [24], the topic has received a surge of interest and different approaches to modeling the cryptanalyses, or rather a cipher's security against the cryptanalyses, have been published. In Chapter 2, we explain modelings that have been introduced in Sun, Hu, Wang, Qiao, Ma, and Song [32], Baksi [2], and Boura and Coggia [8]. But first, this section will explain what MILPs are and why they are so well-suited to the problem of bounding a cipher's security against our cryptanalyses. A more comprehensive introduction to the topic of MILPs, including more examples and figures, can be found in Hagemann [12]. This section's Definition 1.15 and Definition Definition 1.20 have been adapted from that same work.

▶ Definition 1.15 (Mixed-Integer Linear Programs). *A Mixed-Integer Linear Program (MILP) is an optimization with*

$$\begin{aligned} &\max c^T x + d^T y \\ &\begin{pmatrix} A & B \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \geq b \\ &\qquad\qquad x \in \mathbb{Z}_{\geq 0}^{n_1} \\ &\qquad\qquad y \in \mathbb{R}_{\geq 0}^{n_2} \end{aligned}$$

where $c \in \mathbb{Z}^{n_1}$ and $d \in \mathbb{Z}^{n_2}$ are the objective function vectors, $A \in \mathbb{Z}^{m\times n_1}$ and $B \in \mathbb{Z}^{m\times n_2}$ are the constraint matrices and $b \in \mathbb{Z}^{n_1+n_2}$ is the right-hand side.

This means we have an optimization problem with some variables being able to take integer values and some being able to take real values which makes Mixed-Integer Linear Programming a combination of *Integer Linear Programming (ILP)* where all variables have to take integer values and *Linear Programming (LP)* where all variables are allowed to take real values.

To understand why MILP has been used for the calculation of the security bound of ciphers, we take a look at how exactly a notion of security can be calculated for linear and differential cryptanalysis. For that, note that in the ciphers we consider in this work, the S-boxes are the only part with nonlinear properties. Coppersmith [10] implies that ciphers that can be seen as a linear relation are easy to decrypt when saying that without the nonlinearity introduced by S-boxes "[...] the entire algorithm would be linear and thus trivially broken [...].". Thus with Mouha et al. [24], when using MILPs, bounding a cipher's security against cryptanalysis was originally done by calculating a lower bound on the number of actively used S-boxes.

We call an S-box "active" in a cipher's cryptanalysis differential or linear characteristic if it actually has an influence on the resulting ciphertext. Now, if e.g. we find that in a specific cipher, most S-boxes are not active in some characteristics, i.e. they do not influence the ciphertext for some input or keys, we can judge the cipher to be less secure. This is the case since, as mentioned previously, the number of active S-boxes strongly influences how nonlinear a cipher is with nonlinearity being a strongly desirable characteristic for secure ciphers.

Mouha et al. [24] in 2011 bounded a cipher's security, i.e. the probability of inverting an encryption successfully, by calculating the minimum number of active S-boxes and looking at the diffusion or bias introduced by them.

▶ Definition 1.16 (Cipher's security bound against differential cryptanalysis, Mouha et al. [24]). *An n round cipher's secureness can be bounded against differential cryptanalysis by looking at all possible sets of active S-boxes $S_n^{(DC)}$ and taking the minimum amount of diffusion*

$$\min_{S_n^{(DC)}} \left\{ \prod_{s \in S_n^{(DC)}} (MDP_s) \right\}$$

that is introduced by it. A cipher is considered secure against differential cryptanalysis if the bound is less or equal to $\frac{2}{2^{n_{in}}}$. This means that the minimum possible product of active S-boxes' MDP is at least as small as the probability of guessing an input by drawing uniformly random from all possible inputs, holds.

If all the S-boxes in the cipher are of the same type s then the cipher is secure for n rounds if the minimum number of active S-boxes a_n is large enough for $(MDP_s)^{a_n} \leq \frac{2}{2^{n_{in}}}$ to hold.

► Definition 1.17 (Cipher's security bound against linear cryptanalysis, Mouha et al. [24]). *An n rounds cipher's secureness can be bounded against linear cryptanalysis by looking at all possible sets of active S-boxes $S_n^{(LC)}$ and taking the minimum amount of bias*

$$\min_{S^{(LC)n}} \left\{ \prod_{mlp \in \{MLP_s : s \in S^{(LC)}\}} \left(\left[2^{(|\{s \in S_n^{(LC)} : MLP_s = m\}| - 1)} \cdot mlp^{|\{s \in S_n^{(LC)} : MLP_s = m\}|} \right]^2 \right) \right\}$$

that is introduced by it. A cipher is considered secure against differential cryptanalysis if the bound is less or equal to $\frac{2}{2^{n_{in}}}$. This means that the minimum possible product of active S-boxes' MLP is at least as small as the probability of guessing an input by drawing uniformly random from all possible inputs, holds.

Mouha et al. [24] derives the amount of bias introduced from Matsui's Piling-Up Lemma [21]. The minimum amount of introduced bias is derived for a single value of MLP as follows: If all the S-boxes in the cipher have the same MILP m then the cipher is secure for n rounds if the minimum number of active S-boxes a_n is large enough for $(2^{a-1} \cdot (m)^{a_n})^2 \leq \frac{2}{2^{n_{in}}}$ to hold. In order to get the minimum amount of introduced bias for S-boxes with different MLPs, we calculate the bias as stated previously for each MILP $m \in \{MLP_s : s \in S_n^{(LC)}\}$ included in the active S-boxes and take the product of all those.

Baksi [2] and other later publications instead bound the probability by how difficult the substitutions performed by S-boxes make it to retrace what exactly happened.

► Definition 1.18 (Cipher's security bound against differential cryptanalysis, Baksi [2]). *Looking at all possible sets of S-boxes and their transitions $S_n^{(DC,t)}$, where for $(s, t_s) \in S_n^{(DC,t)}$, s is the S-box and t_s is the transition it takes, we bound the corresponding n rounds cipher's security against differential cryptanalysis by taking the minimum amount of diffusion*

$$\min_{S_n^{(DC,t)}} \left\{ \prod_{(s,t_s) \in S_n^{(DC)}} (MDP_s(t_s)) \right\}$$

that is introduced by it. A cipher is considered secure against differential cryptanalysis if the bound is less or equal to $\frac{2}{2^{n_{in}}}$. This means that the minimum possible product of active S-boxes' MDP is at least as small as the probability of guessing an input by drawing uniformly random from all possible inputs, holds.

► Definition 1.19 (Cipher's security bound against linear cryptanalysis, Baksi [2]). *Looking at all possible sets of S-boxes and their transitions $S_n^{(LC,t)}$, where for $(s, t_s) \in S_n^{(LC,t)}$, s is the S-box and t_s is the transition it takes, we bound the corresponding n rounds cipher's security against differential cryptanalysis by taking the minimum amount of bias*

$$\min_{S_n^{(LC,t)}} \left\{ \prod_{(s,t_s) \in S_n^{(LC)}} (MLP_s(t_s)) \right\}$$

that is introduced by it. A cipher is considered secure against linear cryptanalysis if the bound less or equal to $\frac{2}{2^{n_{in}}}$. This means that the minimum possible product of active S-boxes' MLP is at least as small as the probability of guessing an input by drawing uniformly random from all possible inputs, holds.

Although, as of now, this requires more variables, constraints, and computational effort. The difference between the values chosen is that of an MDP/MLP versus the actual diffusion achieved or bias introduced. Our goal is therefore to minimize the product over the factors of the diffusion or the bias introduced. As a MILP this means our target function is minimizing the sum over the logarithm of the factors which equals said product. Meanwhile, our constraints ensure that a variable representing an S-box is always active when the analyzed cipher's input warrants it. The difference between the two is shortly exemplified in Definition 2.4 after the introduction of the exact constraints required by the MILP.

The shortly mentioned ILPs are NP-hard while for LP there exist efficient algorithms for solving them. Despite the ILP staying NP-hard, the complexity of a MILP is generally considered as being somewhere in between the complexity of an ILP and an LP of the same size, as Lohr [20] states. For some kind of specifically structured constraints, there exist fixed-parameter tractable algorithms, which bring the problems' complexity closer to that of LPs. The structures we looked for in this work are *n-fold* and *extended two-stage*.

Let $\mathbb{X}^p = \mathbb{Z}^m \times \mathbb{R}^n$ for some $p = m + n \in \mathbb{Z}$, then a mixed-integer solution $x \in \mathbb{X}^p$ denotes a vector with m integral and n fractional values. In the following, we will use this notation without explicitly stating m and n.

► Definition 1.20 (n-fold MILP). *Let $A_1, \dots, A_n \in \mathbb{Z}^{r \times t_i}$ and $B_1, \dots, B_n \in \mathbb{Z}^{s_i \times t_i}$ be arbitrary matrices, $c \in \mathbb{Z}^{t_i n}$ the objective function and $b = (b_0, \dots, b_n) \in \mathbb{Z}^{r + s_i n}$ the right-hand side for some $r, s_i, t_i, n \in \mathbb{Z}_+$. For a mixed-integer solution $x = (x^{(1)}, \dots, x^{(n)}) \in \mathbb{X}^{t_i n}$, an n-fold MILP is defined as*

$$\max c^T x$$

$$\begin{pmatrix} A_1 & A_2 & \dots & A_n \\ B_1 & 0 & \dots & 0 \\ 0 & B_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & B_n \end{pmatrix} \cdot \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

► Definition 1.21 (Extended two-stage MILP). *Let $C \in \mathbb{Z}^{s_0 \times r}$, $A_1, \dots, A_n \in \mathbb{Z}^{s_i \times r}$ and $B_1, \dots, B_n \in \mathbb{Z}^{s_i \times t_i}$ be arbitrary matrices, $c \in \mathbb{Z}^{t_i n}$ the objective function and $b = (b_0, \dots, b_n) \in \mathbb{Z}^{s_0 + s_i n}$ the right-hand side for some $r, s_0, s_i, t_i, n \in \mathbb{Z}_+$. For a mixed-integer solution $x = (x^{(0)}, \dots, x^{(n)}) \in \mathbb{X}^{r + t_i n}$, an extended two-stage MILP is defined by:*

$$\max c^T x$$

$$\begin{pmatrix} C & 0 & \dots & & 0 \\ A_1 & B_1 & 0 & \dots & 0 \\ A_2 & 0 & B_2 & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & 0 \\ A_n & 0 & \dots & 0 & B_n \end{pmatrix} \cdot \begin{pmatrix} x^{(0)} \\ x^{(1)} \\ \vdots \\ x^{(n)} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

With Definition 1.21 having been adapted from the two-stage definition for ILPs in Klein [18]. Note that an extended two-stage MILP includes a two-stage structure. This is given by the fact that a two-stage MILP is a extended two-stage MILP where all entries of the C block are set to 0.
Now, if we find a way to permute a matrix's rows and columns to show one of those structures, we can show that the underlying MILP has a characteristic allowing it to be solved more easily than others. Using this knowledge, we can gauge some measure of complexity for the security-bound modeling approaches and see how they changed over time.

1.4 Overview

As mentioned previously, this work is based on and an extension to Lohr [20], which, to our knowledge, was the first work to look at block structures in the modeling of linear and differential cryptanalysis security bounds using MILPs.

In said work, constraint matrix generation for the first approach to modeling the security of word-oriented ciphers as a MILP was implemented. The implementation gave matrices for both the linear and differential cryptanalysis of the AES cipher and the Enocoro128-v2 cipher. Lohr found the first real-world occurrence of a 4-block structure, which had previously been written about but never observed, within the constraint matrix for AES and introduced a new structure called stair-fold which can be found in Enocoro128-v2's constraint matrix. The success of that work in large part motivated this thesis in trying to look for further structures and see how different modelings differ in their underlying structures and characteristics.

A large part of the process of looking for further block structure included the extension of the original framework to include the six new modeling approaches as well as two new ciphers. This thesis first introduces all of these implemented modeling approaches in Chapter 2 and after goes into technical details regarding the extension of the framework in Chapter 3. Finally, the results of our analyses of modeling approaches for the MILP representation of differential cryptanalysis on LBlock are presented and this thesis is concluded by reviewing our findings and the potential of the analysis of modeling approaches going forward in Chapter 4.

2 MILP Modeling Approaches

Many different approaches and extensions to the modeling have been made, since the introduction of the first method for modeling cryptanalysis as MILPs by Mouha et al. [24] which Lohr [20] implemented. In this chapter, we explain which of these approaches and extensions we implemented and analyzed. The publications, whose approaches we implemented, were Sun, Hu, Song, Xie, Wang [30], followed by Baksi [2], and some approaches from Boura and Coggia [8].

Given that the modelings introduced from Section 2.3 to Section 2.7 all focus on S-boxes, they, therefore, work under the assumption that bit-oriented modeling is done as well and the constraints from both are being used.

In the following, when we speak about the value of bits or words, it either refers to the XORwise difference between bits in differential cryptanalysis or the linear mask times the bit value of a normal cipher run in linear cryptanalysis depending on the context of which type of cryptanalysis is currently handled.

2.1 Word-Oriented Modeling

The difference between word-oriented and bit-oriented modeling of ciphers is analogous to the difference between the terms bit- and word-oriented ciphers, as explained shortly after Definition 1.3.

Following Mouha et el. [24], when analyzing a word-oriented cipher in a word-oriented fashion, we can represent each word, i.e. a group of input bits that is always modified as one, as a single Boolean variable for a word-oriented cipher. This variable is 1 if and only if any of the bits making it up is nonzero and 0 else. For example, AES works on bytes, i.e. a word size of 8 bits, and the variable x_0 representing the first byte at the beginning of the first round is 0 if and only if all of the 8 bits which make up $x_0 = x_{0,0}, x_{0,1}, \ldots, x_{0,7}$ are zero.

The operations, that play a role in changing the value of said byte are XOR and linear transformation in differential cryptanalysis and three-forked branch and linear transformations in linear cryptanalysis. Input and output variables for all these operations can be constrained using linear inequalities. For a more in-depth explanation of where the inequalities are derived from, we refer to Mouha et al. [24] and Lohr [20]. A XOR operation of 2 words $x_i \oplus x_j = x_{i\oplus j}$ is modeled using a binary dummy variable $d_\oplus$ and the following constraints:

$$\begin{aligned} x_i &\leq d_\oplus \\ x_j &\leq d_\oplus \\ x_{i\oplus j} &\leq d_\oplus \\ x_i + x_j + x_{i\oplus j} &\geq 2 \cdot d_\oplus \end{aligned}$$

A three-forked branch of a word x_{ij} splitting up into two words x_i, x_j can be modeled using a binary dummy variable d_{tfb} and the following constraints:

$$\begin{aligned} x_{ij} &\leq d_{tfb} \\ x_j &\leq d_{tfb} \\ x_j &\leq d_{tfb} \\ x_i + x_j + x_{ij} &\geq 2 \cdot d_{tfb} \end{aligned}$$

The linear transformation operation performing a linear function $lt : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ on n input variables $x_{in_0}, \ldots, x_{in_{n-1}}$ and m output variables $x_{out_0}, \ldots, x_{out_{m-1}}$ can be modeled using the branch number $\mathcal{B}_{lt}$ of the underlying fucntion.

▶ Definition 2.1 (Branch number, Sun et al. [30]). *For a function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$, the branch number $\mathcal{B}_f$ is defined as the minimal hamming weight an input a and output b can have such that $b = f(a)$ and it is not zero*

$$\mathcal{B}_f = \min_{a \in \mathbb{F}_2^n, b \in \mathbb{F}_2^m} (wt(a||b) : lt(a) = b \wedge wt(a||b) > 0)$$

with the hamming weight $wt(x)$ being the number of bits set to 1 in word x.

Given said branch number, we can model the linear transformation lt using a binary dummy variable d_{lt} and the following constraints:

$$\begin{aligned} \sum_{i \in \{0,\ldots,n-1\}} x_{in_i} + \sum_{j \in \{0,\ldots,m-1\}} x_{out_j} &\geq \mathcal{B}_{lt} \cdot d_{lt} \\ x_{in_i} &\leq d_{lt}, & i \in \{0, \ldots, n-1\} \\ x_{out_j} &\leq d_{lt}, & j \in \{0, \ldots, n-1\} \end{aligned}$$

There are no constraints for S-boxes in modeling word-oriented cryptanalysis.

While we have not seen it mentioned in literature, note that the constraints listed previously do not suffice for perfectly accurate results. Taking the inequalities for an XOR operation as an example, an assignment of variables leading to $1 \oplus 1 = 1$ fulfills the constraints. Analogously the same is true for the three-forked branch, despite not being necessarily correct and allowing for variable assignments which cannot happen in a real system. Meanwhile, in the linear transformation, the constraints do not ensure that a transformation from $a \in \mathbb{F}_2^n$ to $b \in \mathbb{F}_2^m$ fulfills $lt(a) = b$. All of this leads to an optimizer, trying to minimize the number of active S-boxes, possibly choosing a, b in a way that $lt(a) \neq b$.

As more options are given to the optimization process which could reduce the number of active S-boxes, this only decreases the security bound. With a lower security bound comes the assumption that the model is less secure than it actually is, which usually means that more rounds of a cipher are used and the process ends up with a better worst-case diffusion than estimated but a larger computational effort than necessary.

2.2 Bit-Oriented Modeling

In contrast to word-oriented modeling, bit-oriented modeling extends the possibility of modeling to ciphers like GIFT64 [3] while also ensuring that the security bound is less underestimated due to e.g. the generalization of multiple different words like 0001, 1111, 1010, etc. all into $x_i = 1$. For the modeling, the largest difference between word- and bit-oriented is that while XOR, three-forked branches, and linear transformations work the same, S-boxes now need to be modeled with the modelings making up a large part of the constraints.

With modeling S-boxes, feasible and impossible transitions are concepts we are going to deal with quite a bit. The modelings presented in Section 2.3 and after all deal with ways to make sure that for our ciphers' S-boxes, feasible transitions are included while impossible transitions are excluded.

► Definition 2.2 (Feasible S-box Transitions). *Let $(X, Y) = ((x_0, \dots x_{v-1}), (y_0, \dots y_{w-1}))$ be a feasible transition of an S-box s, if in differential cryptanalysis at least one pair of inputs, which have an input difference equaling X, give an output difference equaling Y or in linear cryptanalysis if the bias of the linear approximation resulting from using X as the input linear mask and Y as the output linear mask is nonzero.*

Which have a counterpart of so-called impossible transitions.

► Definition 2.3 (Impossible S-box Transitions). *Let F_s be the set of all feasible transitions for a $v \times w$ S-box. The set of impossible transitions $P_s = \mathbb{F}_2^{v+w} \setminus F_s$ is the set of all transitions whose DDT or LAT equals 0, depending on the respective modeling.*

This means when talking about feasible transitions, one can read which transitions are feasible and which are impossible by looking through the corresponding DDT or LAT. Feasible transitions are referred to as "difference propagations" in Zhou et al. [36] and as "possible differentials of an S-box" in Sun et al. [32]. In the second of which, the term of "differentials" is used analogously with should not be confused with the one used for a cryptanalysis' differentials as defined in Definition 2.4 or the term of "differential properties" from **??**.

► Definition 2.4 (Differentials and differential characteristics, Mouha and Preneel [23]). *A differential is a difference propagation from an input difference to an output difference. A differential characteristic defines not only the input and output differences but also the internal differences after every round of the iterated cipher. The probability of a differential is equal to the sum of the probabilities of all differential characteristics that correspond to this differential. It is commonly assumed that the probability of the best differential can accurately be estimated by the probability of the best differential characteristic.*

Linear characteristics can be defined, and are used, analogously.

The first modeling of S-boxes or rather the bit-oriented modeling of ciphers has been done by Sun et al. [32] for differential cryptanalysis. For a $v \times w$ S-box, let $(x_0, \dots, x_{v-1})$ and $(y_0, \dots, y_{w-1})$ be the variables which respectively denote the input and output differences or linear mask, A_t the variable which indicates whether the S-box is active, and d_t another dummy variable described later on.

While all of the ciphers we have implemented in this work use bijective S-boxes, we will start off by recalling which different types exist and how their properties affect the inequalities modeling them. For bijective S-boxes, i.e. the same amount of input bits and output bits as well as an invertible mapping, we need to ensure that there is a nonzero difference in the output if and only if there is a nonzero difference in the input. We do this by ensuring that if at least one input bit is 1, then at least one output bit has to be 1 as well, and the other way around.

$$w \cdot (x_0 + \cdots + x_{v-1}) \geq (y_0 + \cdots + y_{w-1}) \tag{2.1}$$

$$v \cdot (y_0 + \cdots + y_{w-1}) \geq (x_0 + \cdots + x_{v-1}) \tag{2.2}$$

To make sure that these inequalities work as intended, we take a look at the following possible cases:

1. There is no input difference, i.e. $\forall i \in \{0, \ldots, v-1\} : x_i = 0$ and
   (a) No output difference, i.e. $\forall i \in \{0, \ldots, w-1\} : y_i = 0$.
   This should hold as for a bijective function S as per definition $X_1 = X_2$ implies $S(X_1) = S(X_2)$. As both the difference of all input bits as well as that of all output bits is zero, our inequalities come in as $0 \geq 0$.
   (b) An output difference, i.e. $\exists i \in \{0, \ldots, w-1\} : y_i = 1$.
   This should not hold for a bijective function S as $X_1 = X_2$ but $S(X_1) \neq S(X_2)$ would mean that S can act nondeterministic on the same input. As the difference of all input bits is zero but that of all output bits is not, the constraint in Inequality 2.1 would result in $0 \geq \ell$ with $\ell \in \{1, \ldots, v-1\}$, which does not hold.
2. There is an input difference, i.e. $\exists i \in \{0, \ldots, v-1\} : x_i = 1$ and
   (a) No output difference, i.e. $\forall i \in \{0, \ldots, v-1\} : x_i = 0$.
   This should not hold as $X_1 \neq X_2$, but $S(X_1) = S(X_2)$ would mean that for a function S is not surjective and therefore out function (or rather S-box S) would not be bijective. As the difference of all input bits is nonzero but that of all output bits is not, the constraint in Inequality 2.2 would result in $0 \geq \ell$ with $\ell \in \{1, \ldots, w-1\}$, which does not hold.
   (b) An output difference, i.e. $\exists i \in \{0, \ldots, v-1\} : x_i = 1$. This should hold for a bijective function S as per definition $X_1 \neq X_2$ implies $S(X_1) \neq S(X_2)$. As both the difference of all input bits as well as that of all output bits are nonzero, our inequalities hold, since they have at least v or w on their left side which is greater than any value the respective right side can take.

For surjective S-boxes, the condition that a nonzero input difference yields a nonzero output difference does not have to hold. In Sun et al. [32], constraints from both Inequality 2.2 and Inequality 2.1 are removed if a nonbijective S-Box is modeled. However, since all S-boxes are functions, our framework would continue on using the constraint from Inequality 2.1, which ensured that a nonzero output difference implies a nonzero input difference, even with nonbijective S-Boxes.

Additionally, for an S-box s whose branch number is larger than 2 or which is not invertible, e.g. 6×4 S-boxes used in the *Data Encryption Standard (DES)* cipher, another dummy variable is introduced. This dummy works analogous to those for XORs and three-forked branches.

$$\sum_{i \in \{0,\ldots,v-1\}} x_i + \sum_{j \in \{0,\ldots,w-1\}} y_j \geq \mathcal{B}_s \cdot d_s \tag{2.3}$$

$$d_S \geq x_i, \qquad i \in \{0, \ldots, v-1\}$$

$$d_S \geq y_j, \qquad j \in \{0, \ldots, w-1\}$$

For S-boxes that do not fulfill the aforementioned criteria, i.e. their branch number $B_s = 2$ and they are invertible, the inequalities using the d_s dummy variable are redundant. This is the case because if S-box s's branch number is 2, the constraints from Inequality 2.3 is already handled by the constraint from Inequality 2.2 and the constraint from Inequality 2.1 due to there being at least one x var and one y var being set to 1 if active.

Identically to the count of active S-boxes in Mouha et al. [24], we ensure that we mark an S-box as active, i.e. $a_t = 1$, if and only if at least one of the S-box's input bits is active, i.e. there is a difference in the compared bits. We ensure this by letting the dummy variable only be 0 if all input bits x_i are 0 and requiring there to be at least one input bit set to 1 if the dummy is 0.

$$a_t \geq x_i, \qquad i \in \{0, \ldots, v-1\} \tag{2.4}$$

$$\sum_{i \in \{0,\ldots,v-1\}} x_i \geq a_t \tag{2.5}$$

With all of this, we have a base for modeling differential cryptanalysis of ciphers as bit-oriented. When modeling S-boxes in linear cryptanalysis, as described in
Sun, Hu, Wang, Wang, Qiao, Ma, Shi, and Song [31], the modelings work analogously. For a $v \times w$ S-box, let $(x_0, \ldots, x_{v-1})$ and $(y_0, \ldots, y_{w-1})$ be the variables which denote the respective linear masks of input and output, and A_t the variable which indicates whether the S-box is active. To check for whether an S-box is active, this time we just compare the output linear mask with our indicator/dummy variable A_t

$$y_i \leq A_t, \forall i \in \{0, \ldots, w-1\}$$

$$\sum_{i=0}^{w-1} y_i \geq A_t$$

which ensures that $A_t = 1$ iff the output linear mask is nonzero. For bijective S-boxes, we again follow that a nonzero input linear mask must result in nonzero output linear mask and vice versa which can be achieved by the same constraints as listed in Inequality 2.2.

While, as mentioned at the end of Section 2.1, the modelings of constraints we have taken from said section also only provide an estimate to the security bound according to the respective cryptanalysis and not an accurate number, the same applies to the modeling of S-boxes so far.

For example, given the DDT of the PRESENT S-box in Table 1.1, none of the possible inputs with input difference $(0,0,0,1)$ can yield an output difference of $(0,0,0,1)$. Yet, those bit values do fulfill all of the previously introduced inequalities. This, again, yields a less accurate amount of active S-boxes since e.g. for given input differences a choice of output differences could be made which is not feasible given the specific S-box that is being analyzed, yet minimizes the number of active future S-boxes. We are now going to look at approaches to fix this mistake when it comes to S-boxes.

2.3 Logical Condition Modeling

The first approach introduced by Sun et al. [32] to tighten the region of feasible variable assignments to be more in line with the region of feasible transitions is their so-called *logical condition modeling*. In their *Theorem 1* it is stated and proven that m binary variables $(x_0, \ldots, x_{m-1}) = (\delta_0, \ldots, \delta_{m-1}) \in \mathbb{F}_2^m$ implying one other binary variable $y = \delta \in \mathbb{F}_2$ can be described using the inequality

$$\left(\sum_{i=0}^{m-1} (-1)^{\delta_i} x_i \right) + (-1)^{\delta+1} y - \delta + \left(\sum_{i=0}^{m-1} \delta_i \right) \geq 0.$$

When creating constraints for differential cryptanalysis, this leads to us looking at an S-box's so-called differential properties. These are a generalization of the information found in the DDT. For example, when looking at the DDT for the PRESENT cipher Table 1.1, we can see the input difference of 9_{10} or rather 1001_2 leading to the output differences of

$$2_{16}, 4_{16}, 6_{16}, 8_{16}, C_{16}, E_{16} = 001\mathbf{0}_2, 010\mathbf{0}_2, 100\mathbf{0}_2, 110\mathbf{0}_2, 111\mathbf{0}_2$$

All of these output differences correspond to an even number in the output difference, i.e. an output difference with a 0 in the least significant bit. We therefore get the implication that if $(x_0, \ldots, x_3) = (1,0,0,1)$ then $y_3 = 0$ which is also modeled using the constraint

$$-x_0 + x_1 + x_2 - x_3 - y_3 + 2 \geq 0.$$

The other differential properties of the PRESENT S-box are

1. if $x = (0,0,0,1)$ then $y_3 = 1$
2. if $x = (1,0,0,0)$ then $y_3 = 1$
3. if $y = (0,0,0,1)$ then $x_3 = 1$
4. if $y = (0,1,0,0)$ then $x_3 = 1$
5. if $y = (0,1,0,1)$ then $x_3 = 0$

which allows us to exclude 64 impossible transitions with just 6 constraints.

2.4 Convex Hull Modeling

Sun et al. [32] described the next addition in the same publication as the logical condition modeling in Section 2.3. In order to restrict the bits' value assignment to mostly valid configurations or rather bit assignments, which match transition pairs, the concept of using a convex hull was introduced. Following de Berg, Cheong, van Kreveld, Overmars [4], we define a convex hull.

▶ Definition 2.5 (Convex Hull). *A convex set C is a set of points in a plane, which, for every pair of points $p, q \in C$, also includes all points laying on the line in between these two points. For a given set of points X, a convex hull of that set $\mathcal{CH}(X)$ is a minimum convex set that includes all of the points from X.*

Conveniently for us, a convex hull has a representation done fully through linear inequalities. This so-called half-space representation makes the concept of convex hulls easily usable in the setting of representing a cipher as a MILP.

The convex hull, used to restrict the transitions, is constructed over a space including the points representing feasible transitions. For this, we take all feasible transitions pairs $(X, Y) = ((x_0, \dots x_{v-1}), (y_0, \dots, y_{w-1}))$ and representing them as vectors $(x_0, \dots, x_{v-1}, y_0, \dots, y_{w-1})^T$. The construction of the convex hull is performed using the `sagemath` programming language. Given the vectors, we have it compute a polyhedron for which we can get the convex hull in the form of linear inequalities. These are formulated as

$$\lambda_{0,0}x_0 + \lambda_{0,1}x_1 + \cdots + \lambda_{0,v-1}x_{v-1} + \lambda_{0,v}y_0 + \lambda_{0,(v+1)}y_1 + \cdots + \lambda_{0,(v+w-2)}y_{w-1} \geq c_0$$

with $\lambda_{i,j} \in \mathbb{Z}$ with $i, j \in \mathbb{N}$ representing the factor of the j-th variable in the i-th inequality and c_i represents the constant used in the i-th inequality.

Since many of the constraints of the convex hulls are typically redundant for us, their number can be reduced by using a greedy Algorithm 1 which exhaustively includes the constraint excluding the most impossible transitions until none of the left over constraints exclude any more.

As Baksi [2] was the first to explicitly mention, one of the drawbacks is that some invalid S-box transitions might still be accepted due to the convex nature of a convex hull.

This was exemplified using the trivial S-box `0123456789ABCDEF` which simply substitutes every input with itself and changes nothing in the end. For these transition pairs or rather the vectors of said transition pairs, i.e. $(0,0,0,0,0,0,0,0)$ for $(0,0)$, $(0,0,0,1,0,0,0,1)$ for $(1,1)$, $(0,0,1,0,0,0,1,0)$ for $(2,2)$, etc. until $(1,1,1,1,1,1,1,1)$ for $(15,15)$ or (F,F) the convex hull can be described by the constraints in Table 2.1.

These do not constrain anything given that none of the vectors $(x_0, x_1, \dots, x_7) \in \mathbb{F}_2^8$ falsify any of the constraints. It is suggested that using equality constraints in combination with inequality constraints leads to a complete and correct exclusion of impossible transitions, though this has not been proven.

```
Input   : $C$ - Set of $k$ constraints describing an S-boxes convex hull;
          $X$ - Set of the S-box's $\ell$ impossible transitions;
Output: $best\_constraints$ - A minimum set of constraints excluding the maximum number of
          impossible transitions;
Function Greedy_choice($C$,$X$):
    current_best_constraint := None;
    $constraints\_left \leftarrow C$;
    $transitions\_to\_be\_excluded \leftarrow X$;
    $best\_constraints \leftarrow \emptyset$;
    while $transitions\_to\_be\_excluded \neq \emptyset$ do
        current_best_constraint ← one of the inequalities from $constraints\_left$ which
          excludes the most impossible transitions from $transitions\_to\_be\_excluded$;
        $newly\_excluded \leftarrow$ { transitions excluded by current_best_constraint } A subroutine
        of finding the constraint which currently excludes the most not yet excluded constraints
        is not given in the publication. Given the small problem size, we simply use the brute
        force approach.
        if $|newly\_excluded| == 0$ then
            break;
        end
        $constraints\_left \leftarrow constraints\_left -$ {current_best_constraint};
        $transitions\_to\_be\_excluded \leftarrow transitions\_to\_be\_excluded - newly\_excluded$;
        $best\_constraints \leftarrow O \cup$ { current_best_constraint };
    end
    return $best\_constraints$;
end
```

Algorithm 1: Sun et al. [32]'s greedy algorithm for choosing constraints

$-x_4$				$+1$	≥ 0
	$-x_5$			$+1$	≥ 0
		$-x_6$		$+1$	≥ 0
			$-x_7$	$+1$	≥ 0
x_4					≥ 0
	x_5				≥ 0
		x_6			≥ 0
			x_7		≥ 0

Table 2.1. Convex hull constraints for the trivial S-box `0123456789ABCDEF`

While the first publication only describes this approach for differential cryptanalysis, Sun et al. [31] mention the application to the transitions in linear cryptanalysis analogously by substituting the transitions from the DDT with those from the LAT which show a bias, i.e. a statistical deviation marked in the table by a nonzero entry.

2.5 Exact Transition Modeling

Baksi [2] was not only the first to explicitly point out that many convex hull modelings allow for impossible transitions to fulfill the constraints but also included another approach to modeling S-boxes in a way that their constraints only allow for the feasible transitions. The modeling applies to both linear and differential cryptanalysis.

For the j-th round's i-th S-box, variables are created from which the exact transition that was taken can be understood. Let $Q_{i,j}$ be this model's equivalent to the A_t used previously, i.e. a dummy variable that indicates that a specific S-box is active. Then $Q^p_{i,j}$ indicates that the transition taken is one of the ones whose LAT/DDT value equals p with another dummy variable $Q^p_{i,j,\ell}$ with ℓ indicating which of the q_p possible different transitions of value p is taken. Lastly, $x_{i,j} = (x_{i,j,0}, \dots, x_{i,j,v-1}) \in \mathbb{F}_2^v$ is the vector including the variables representing the input difference or the input linear mask while $y_{i,j} = (y_{i,j,0}, \dots, y_{i,j,w-1}) \in \mathbb{F}_2^w$ is the vector including the variables representing the output difference or the output linear mask.

Given these variables, we can say that the S-box being active $Q_{i,j} = 1$ biimplicates one of the feasible transitions being taken $Q^p_{i,j,\ell} = 1$ and with

$$Q_{i,j} = \sum_p Q^p_{i,j}$$

$$Q^p_{i,j} = \sum_\ell Q^p_{i,j,\ell}$$

we end up fulfilling this biimplication. So now if we manage to set it up that $Q^p_{i,j,\ell} = 1$ if for the j-th round's i-th S-box, the input $x_{i,j}$ and output $y_{i,j}$ correspond to the DDT/LAT's ℓ-th transition with value p, then we have it set up correctly. For that, let $\lambda_{in} = (\lambda_{in,0}, \dots, \lambda_{in,v-1})$ and $\lambda_{out} = (\lambda_{out,0}, \dots, \lambda_{out,w-1})$ be the transition that a $Q^p_{i,j,\ell}$ represents. $Q^p_{i,j,\ell} = 1$ iff $x_{i,j} = \lambda_{in}$ and $y_{i,j} = \lambda_{out}$ which is constrainted by

$$\sum_{pos=0}^{v-1} (1-x_{i,j,pos}) \cdot \lambda_{in,pos} + x_{i,j,pos} \cdot (1-\lambda_{in,pos}) + \sum_{pos=0}^{w-1} (1-y_{i,j,pos}) \cdot \lambda_{out,pos} + y_{i,j,pos} \cdot (1-\lambda_{out,pos}) \leq (v+w) \cdot Q^p_{i,j,\ell}.$$

Take the LAT of the PRESENT S-Box as seen in Table 1.2 for example. Let $Q_{PRESENT}$ be our $Q_{i,j}$. In $Q^p_{PRESENT}$, the p can take a value from $\{2, 4, 8\}$. The numbers q_p possible paths are $q_2 = 96$, $q_4 = 36$, and $q_8 = 1$. So for example $(1, 7)$ is a transition with LAT entry of 4 and, depending on how we read the table, the 2nd one. Therefore $Q^4_{PRESENT,2}$ is the dummy variable indicating that the transition was taken.

While $x_{i,j} = \lambda_{in}$ and $y_{i,j} = \lambda_{out} \rightarrow Q^p_{i,j,\ell} = 1$, the other direction is not explicitly constrained by the inequality. However, implicitly the process of minimizing the level of diffusion achieved by a cipher, leads to $Q^p_{i,j,\ell} = 1$ only if it has to.

Note that the second to last equation is different from the one presented in Baksi [2], After correspondence with this approach's author, we came to the agreement that there is some faulty notation which has been corrected here.

2.6 Exclusion of Impossible Transitions Modeling

From Boura and Coggia [8], we take yet another modeling approach for S-boxes.

Firstly, what had been implicitly used in previous approaches but never stated explicitly is the fact that an impossible transition $(\lambda_{in}, \lambda_{out})$ with $\lambda_{in} = (\lambda_{in,0}, \ldots, \lambda_{in,v-1})$ and $\lambda_{out} = (\lambda_{out,0}, \ldots, \lambda_{out,w-1})$ can be excluded or rather be made impossible for input x and output y using

$$\sum_{pos=0}^{v-1} (1 - \lambda_{in,pos}) \cdot x_{pos} + \lambda_{in,pos}(1 - x_{pos}) + \sum_{pos=0}^{w-1} (1 - \lambda_{out,pos}) \cdot y_{pos} + \lambda_{out,pos}(1 - y_{pos}) \geq 1.$$

This inequality only constrains (x, y) to not equal the transition $(\lambda_{in}, \lambda_{out})$ since only if equality of the two holds, the sum comes to 0. In all other cases, (x, y) has a 0 where $(\lambda_{in}, \lambda_{out})$ has a 1 or vice versa. If that happens, w.l.o.g. looking at getting a 0 while there is 1 in the impossible transition, the sum for that position which is

$$\begin{aligned}(1 - \lambda_{in,pos}) \cdot x_{pos} + \lambda_{in,pos}(1 - x_{pos}) &= (1 - 1) \cdot x_{pos} + 1(1 - x_{pos}) \\ &= 0 \cdot x_{pos} + (1 - x_{pos}) \\ &= (1 - x_{pos}) \\ &= (1 - 0) = 1\end{aligned}$$

our sum does not add up to 0 but more.

For example, we are going to exclude the impossible transition $(2, 14)$ from the PRESENT S-box. This gives us the $(\lambda_{in}, \lambda_{out}) = ((0, 0, 1, 0), (1, 1, 1, 0))$ with the transition being therefore excluded by

$$x_0 + x_1 + (1 - x_2) + x_3 + (1 - y_0) + (1 - y_1) + (1 - y_2) + y_3 \geq 1.$$

The usage of this approach is not pursued further in the publication. However, the approach can be utilized to e.g. exclude the impossible transitions which are still included by the convex hull modeling to fix the inaccuracies pointed out in Baksi [2], is something worth taking a look at, which we have not found being mentioned in the literature so far.

2.7 Efficient Logical Condition Techniques

The previously explained approaches of Sun et al. [32] and Baksi [2] were motivated by no modeling existing yet and the existing modeling being incomplete respectively. Meanwhile, another approach by Boura and Coggia [8] was motivated by the existing approaches being inefficient for larger S-boxes like the 8×8 S-box used in AES.

Here, given a vector $u = (u_0, \ldots, u_{m-1}) \in \mathbb{F}_2^m$, let the set of vectors which u succeeds be $Prec(u) = \{x \in \mathbb{F}_2^m | x \preceq u\}$ with $x \preceq u$ meaning that $\forall i \in 0, \ldots, m-1 : x_i \leq u_i$. We also define with $supp(u) = i \in \mathbb{N} : u_i = 1$ the set of indices at which a vector u's value is 1.

The authors propose that given $a, u \in \mathbb{F}_2^m$ such that none of the a_i, u_i are both 1, i.e. $|\{i \in \mathbb{N} : a_i = u_i = 1\}| = 0$ or $supp(a) \cap supp(u) = \emptyset$, and I being the set of indices for positions at which a as well as u have 0 $I = \{0, \ldots, m-1\} \setminus (supp(a) \cup supp(u))$, it can be followed that

$$\forall x \in \mathbb{F}_2^m : \left(- \sum_{i \in supp(a)} x_i + \sum_{i \in I} x_i \geq 1 - wt(a) \right) \Leftrightarrow x \notin a \oplus Prec(u).$$

For example, from the LAT of the PRESENT S-box Table 1.1, we can take $u = (1, 1) = 00010001_2$ and $a = (6, A) = 01101010_2$. With $Prec(00010001_2) = \{000\mathbf{0}000\mathbf{0}_2, 000\mathbf{0}000\mathbf{1}_2, 000\mathbf{1}000\mathbf{0}_2, 000\mathbf{1}000\mathbf{1}_2\}$, we get $a \oplus Prec_u = \{01101010_2, 01101011_2, 01111010_2, 01111011_2\} = (6, A), (6, B), (7, A), (7, B)$, all of which are impossible transitions. If the constraint

$$x_0 - x_1 - x_2 - x_4 + x_5 - x_6 \geq 1 - 4$$

holds, we can be assured that all 4 impossible transitions are excluded.

Using this proposition, if we are lucky we can exclude a lot of impossible transitions using very few constraints. Boura and Coggia [8] continue on by giving an algorithm that searches for spaces of the form $a \oplus Prec(u)$, which we have included in Algorithm 2.

```
Input   : $P$ - Set of S-box's impossible transitions;
Output: $S_{out}$ - The set of all pairs $(a, u)$ such that $a \oplus Prec(u) \subseteq P$;
Function AffinePrec($P$):
    $S_{out} \leftarrow \emptyset$;
    for $a \in P$ do
        $S_{interesting} \leftarrow \emptyset$;
        for $i \in \{0, \dots, v + w\}$ do
            $S_i \leftarrow \emptyset$;
            $U_i \leftarrow \emptyset$;
        end
        for $p \in P$ do
            $u \leftarrow (a \oplus p)$;
            if $supp(a) \cap supp(u) == \emptyset$ then
                $U_{wt(u)} \leftarrow U_{wt(u)} \cup \{u\}$;
            end
        end
        if $U_1 == \emptyset$ then
            $S_{interesting} \leftarrow \{(a, u) : u \in U_0\}$;
        else
            $S_{interesting} \leftarrow \{(a, u) : u \in U_1\}$;
        end
        $S_0 \leftarrow \{a \oplus Prec(u) : u \in U_0\}$;
        $S_1 \leftarrow \{a \oplus Prec(u) : u \in U_0\}$;
        for $k \in \{2, \dots, v + w\}$ do
            for $u \in U_k$ do
                if $\forall pred \prec u : wt(pred) = k - 1$ and $a \oplus Prec(pred) \subset a \oplus Prec(u)$ then
                    $S_k \leftarrow S_k \cup \{(a, u)\}$;
                    for $pred \prec u$ with $wt(pred) = k - 1$ do
                        $S_{interesting} \leftarrow S_{interesting} - \{(a, pred)\}$;
                    end
                end
            end
            $S_{interesting} \leftarrow S_{interesting} \cup S_k$;
        end
        $S_{out} \leftarrow S_{out} \cup S_{interesting}$;
    end
    return $S_{out}$;
end
```

Algorithm 2: Boura and Coggia's [8] algorithm for finding all subsets of the impossible transitions of form $a \oplus Prec(u)$ for a $v \times w$ S-box s.

2.8 Other Works Considered

With our focus shifting ever so slightly to the modeling of S-boxes, the approaches presented in Zhou et al. [36] as well as the remaining approaches in Boura and Coggia [8] were out of scope for this work.

In Zhou et al. [36], approaches were presented that dealt with the inclusion of a few more constraints as to make the MILP easier to calculate especially in a setting of parallel computation. For the same goal of more efficient computation, heuristics for the order of value assignment to variables were introduced which, although not rooted in theory, seemed to lead to better results and would be worth a thorough analysis. Boura and Coggia [8] included another approach, extending the one explained in *Efficient Logical Condition Techniques* in Section 2.7, which made use of the concept of a ball and its sphere in metric space, and a new approach to modeling a cipher's linear layers using matrix multiplication to reduce the number of constraints.

2.9 tl:dr

The first two sections of this chapter provide a short recap regarding word-oriented cryptanalysis, which has previously been implemented by Lohr [20], followed by what changes with modeling bit-oriented cryptanalysis as MILPs. This is followed by six sections explaining some of the different approaches and extensions for modeling how feasible and impossible S-box transitions are respectively allowed and excluded. They can be summarized as follows:

2.3 How to: Exclude transitions that do not follow the implication of "if these k variables $x_1, \dots, x_k$ take specific values then this variable y is only allowed to have this value".

2.4 How to: Apply the concept of convex hulls from computational geometry in order to exclude a lot of impossible transitions while allowing all feasible transitions.

2.5 How to: Exclude all impossible transitions while not only including all feasible transitions but also knowing the exact transition that was taken.

2.6 How to: Exclude a specific assignment of variables. This can be applied to excluding all impossible transitions of an S-box.

2.7 How to: Exclude multiple specific assignments of variables at once using one constraint that makes use of specific characteristics of two points in space.
How to: Find pairs of points such that they have the desired characteristics and thereby exclude nothing but unwanted assignments.

3 ASEL Framework Extension

This chapter describes the ASEL (abbr. *Aura Sofia Elisabetta Lohr*) framework, originally by Lohr 2022 [20], which we further developed, and subsequently used to generate the constraint matrices analyzed in Chapter 4. On our github repository, which was last referenced at commit E1E03B6, the code can be found for easier accessibility. The exact contributions to the code, which have been made by us, can be clearly differentiated by going through the repository's commit history, given that we worked independently from the original author and the *git blame* file is available online as well as on the codebase provided with printed versions of this thesis.

It should be noted that this section differentiates between the terms of *module* and *project*. When we refer to the module or the codebase we are just talking about the code responsible for generating constraint matrices, whereas project refers to all files which also includes the README and unit tests.

3.1 Project Structure

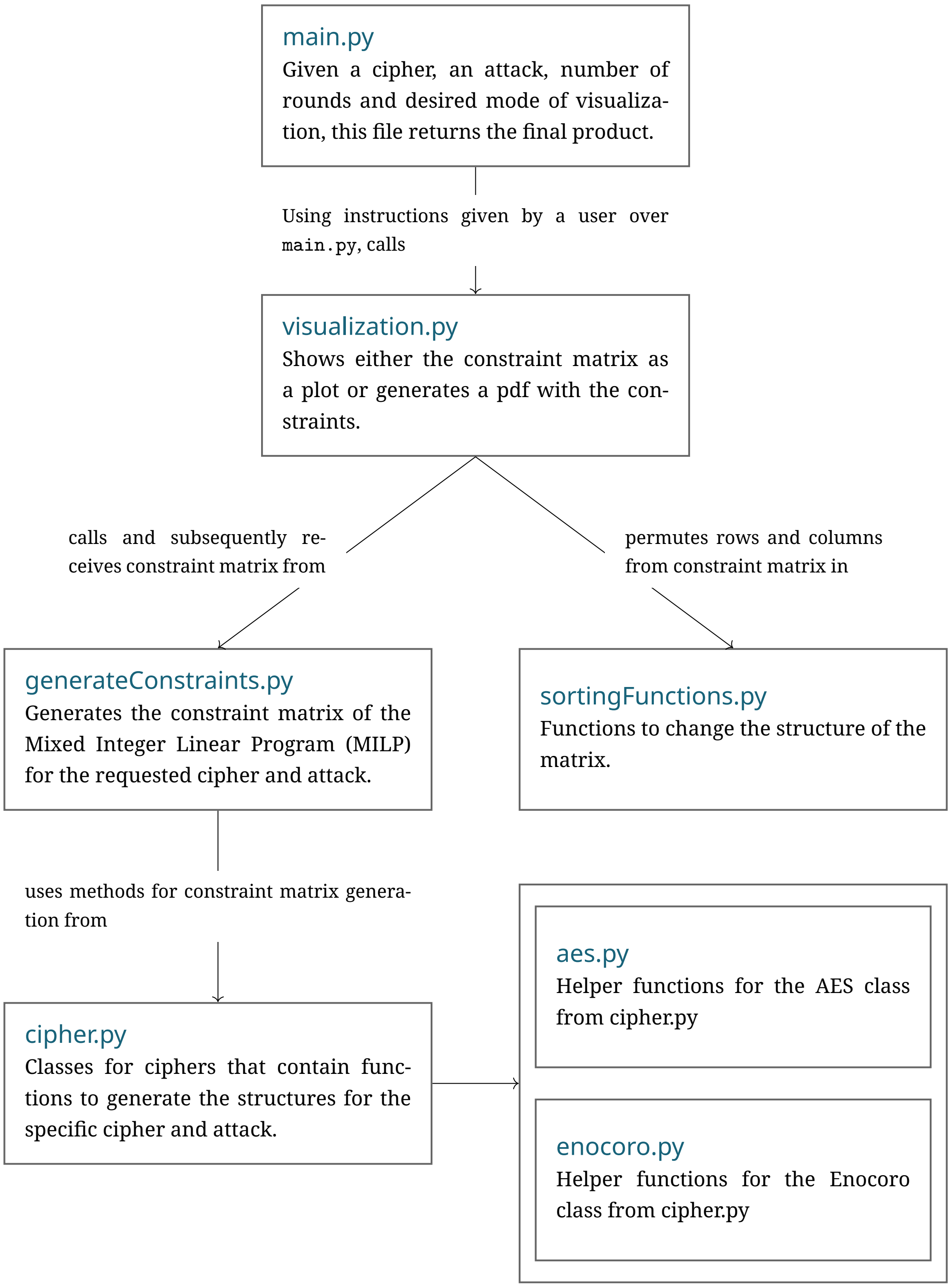
We will start off with a short overview of how the framework started out. The repository at commit 48314A1 on December 4, 2022, used to be mostly scripts and functions, as described more thoroughly in Lohr 2022 [20], with the module and the project being the same. Although `cipher.py` is originally described as having classes for each of the implemented ciphers, *AES* and *Enocoro*, it made no use of most of the functionalities object-oriented programming has to offer. The module's files were all stored in the same folder without any further structure, which was fine for a project of its size. How the files of the source code related at that time can be seen in Figure 3.1. However, with the pending increase in the size of the project, a restructuring was necessary.

So, with our ambition to go from word-oriented analysis to bit-oriented analysis and include more different approaches, we decided to not only make the code more modular but also change up the project's structure. In particular, our focus went to the interactions between the `generateConstraints.py`, `cipher.py`, and `aes.py` as well as `enocoro.py` modules.

With the number of files required for a more modular approach, we are first taking a look at the initial project structure.

```
├── main.py
├── generateConstraints.py
├── visualizations.py
├── sortingFunctions.py
├── cipher.py
├── aes.py
└── enocoro.py
```

The files of `main.py`, `generateConstraints.py`, `visualizations.py`, and `sortingFunctions.py` kept their respective task, but changes were made to fit the new data structures and reduce the overall complexity while increasing robustness. This lead to `generateConstraints.py` assigning no more


main.py
Given a cipher, an attack, number of rounds and desired mode of visualization, this file returns the final product.
Using instructions given by a user over main.py, calls
visualization.py
Shows either the constraint matrix as a plot or generates a pdf with the constraints.
calls and subsequently receives constraint matrix from
permutes rows and columns from constraint matrix in
generateConstraints.py
Generates the constraint matrix of the Mixed Integer Linear Program (MILP) for the requested cipher and attack.
sortingFunctions.py
Functions to change the structure of the matrix.
uses methods for constraint matrix generation from
cipher.py
Classes for ciphers that contain functions to generate the structures for the specific cipher and attack.
aes.py
Helper functions for the AES class from cipher.py
enocoro.py
Helper functions for the Enocoro class from cipher.py

■ Figure 3.1. Diagram depicting relations in the framework of the ASEL framework at commit 48314A1. Arrows signify function calls. Figure has been taken from Lohr 2022 [20] and modified.

variables explicitly and simply calling a class method for an instance of the cipher.

Since the code made no use of inheritance, instances, and attributes, we decided to change that and try to reduce the amount of input and output required, especially in the attempt not to task the system in mistaken or redundant call-by-value and call-by-reference. We decided to have `cipher.py` be the file for the base class `Cipher` instead of containing multiple classes which we put into its own folder.

```
├─ main.py
├─ generateConstraints.py
├─ visualizations.py
├─ sortingFunctions.py
└─ cipher
   ├─ cipher.py
   ├─ aes.py
   └─ enocoro.py
```

While following the previous structure, the files `aes.py`, `lblock.py`, and `gift64.py` would have only included helper function for the ciphers, they now include one class each. The classes `AES`, `LBlock`, and `Gift64`, are the ones that generate the constraint matrix for the respective cipher and already include their own helper functions. The ciphers, these classes are based on, are introduced later in Section 3.3. Whether the constraints generated are those for linear or differential cryptanalysis depends on whether e.g. the `AES` from the `aes.py` file is the one found under the path of `cipher/linear` or `cipher/differential`, given that we have each of our ciphers represented in both types of cryptanalysis. The differentiation between word- and bit-oriented modelings was small enough to be included in the classes themselves.

```
├─ main.py
├─ ...
└─ cipher
   ├─ cipher.py
   ├─ folder with linear cryptanalysis cipher subclasses "linear"
   │  ├─ aes.py
   │  ├─ lblock.py
   │  └─ gift64.py
   └─ folder with differential cryptanalysis cipher subclasses "differential"
      ├─ aes.py
      ├─ lblock.py
      └─ gift64.py
```

We included a class `SBox` for S-boxes in `sbox.py`, given that in bit-oriented every of our ciphers needs more information on them and that the construction of instances is more flexible which fits how different ciphers require different numbers of S-boxes.

```
├─ main.py
├─ ...
└─ cipher
   ├─ cipher.py
   ├─ linear
   │  └─ ...
   ├─ differential
   │  └─ ...
   └─ sbox.py
```

For the `Sbox` to be able to generate all variables we were interested in, we included the file `convexHull.py` which makes use of some of the functionalities of the mathematics software system *SageMath* [33] by importing its source code as a library.

```
├─main.py
├─...
├─convexHull.py
└─cipher
  ├─cipher.py
  ├─linear
  │ └─...
  ├─differential
  │ └─...
  └─sbox.py
```

Finally, the decision was made to include another base class `Action` in `action.py`, from which the subclasses were able to easily generate the constraints for the standard operations our ciphers are made up of. The final structure of the module came to be

```
├─main.py
├─...
├─convexHull.py
└─cipher
  ├─cipher.py
  ├─linear
  │ └─...
  ├─differential
  │ └─...
  ├─sbox.py
  ├─action.py
  └─folder with action subclasses "actions"
    ├─lineartransformationaction.py
    ├─overwriteaction.py
    ├─permutationaction.py
    ├─sboxaction.py
    ├─threeforkedbranchaction.py
    └─xoraction.py
```

What follows is a short introduction of each of the files in the order they were mentioned previously. It mainly contains their primary use, where they are called from or which classes they inherit from, and their most important methods or functions. An outline of how these files all interact is found in Section 3.2.

main.py, visualization.py, sortingFunctions.py. As mentioned previously, these files continue on with the same functionality as described in Lohr 2022 [20]. `main.py` allows users to interactively, using terminal prompts, generate the matrices they are interested in. Users are able to choose which type of cryptanalysis, linear or differential, which of the ciphers, AES, LBlock, and Gift64, it is to be applied to, whether the analysis is done word- or bit-oriented, and which of the models they want to have used.

Visualizations of generated constraint matrices are still performed in `visualizations.py`, although due to the increase in the number of constraints with bit-oriented ciphers, the visualization in a PDF is not fully supported anymore.

generateConstraints.py. While previously `generateConstraints.py` called the methods for generating the constraint matrices, now it only creates an instance of the chosen cipher and initiates the generation which happens fully within the specific cipher's class. If the constraints generated include those for S-boxes, then after the generation is done a matrix containing the constraints from Section 2.2 and one containing the constraints from one of the S-box modelings are put together. The removal of zero columns and rows happens in this file as well.

cipher/cipher.py. Includes the `cipher` base class used by all of the classes structuring a cipher's cryptanalysis constraint modeling. Includes a constructor method used by all subclasses and some more methods as not to write redundant code. No instances initiated or methods called by any class or file outside of the subclasses.

cipher/differential/gift.64, cipher/differential/lblock.py, and cipher/differential/aes.py. Includes the subclasses to the aforementioned `cipher` base class, which generate a cipher's cryptanalysis constraint modeling, named `Gift64`, `LBlock`, and `AES` respectively. Instances of a chosen one are created in `cipher` while `cipher`, as well as technically in `cipher`, and `cipher` due to inheritance. The `run_round()` method is the only method called from outside the class, which generates the constraints for a cipher's next round using instances of the respective `Action` subclasses.

cipher/linear/gift64.py, cipher/linear/lblock.py, cipher/linear/aes.py. These classes, again named `Gift64`, `LBlock`, and `AES` respectively, do the same as their counterparts in `cipher/differential/` but for linear cryptanalysis. For simplicity's sake, the classes for linear cryptanalysis inherit from those for differential cryptanalysis. The `run_round()` method, with the same functionality as aforementioned, is again the only method called from outside the class.

cipher/sbox.py. The `cipher/sbox.py` file includes the `Sbox` class which acts as a pseudo data structure where a specific S-box is an instance constructed using a substitution table and then offering us methods and attributes for all of the information we might require of said S-boxes such as its DDT, LAT, etc. For all of the S-boxes required by a cipher, an instance of the `Sbox` class is created. Functions from the `convexHull.py` file are called but it does not make use of any of the other classes in the project. Since this class has no use other than providing information, for ciphers such as AES which use the same kind of S-box in parallel, it sufficed to just reference the same instance multiple times.

convexHull.py. As aforementioned, the `convexHull.py` file makes use of some of the functionalities of the mathematics software system *SageMath* [33] by importing its source code as a library. Functions from this file are called in the `Sbox` class.

cipher/action.py. This file includes the `Action` base class which is used by the classes in the files in `cipher/actions/*`. Its methods are mostly generalizations allowing the subclasses to write to matrices more easily and edit multiple entries at once with less commands.

cipher/actions/*. The classes `LinTransformationAction`, `OverwriteAction`, `PermutationAction`, `SboxAction`, `ThreeForkedBranchAction`, and `XorAction`, found in the respective files, serve as ways to easily map a cipher's operation to the generation of the corresponding constraints. In the ciphers' classes, instances of them are created and their `run_action()` methods are called.

Variable type	naming convention	prefix (if applicable)
bits' values	prefix + numbering	x
XOR dummy variables	prefix + numbering	dx
three-forked branch dummy variables	prefix + numbering	dt
linear transformation dummy variables	prefix + numbering	dl
S-box active dummy variables	prefix + numbering	a
S-box Inequality 2.3 dummy variables	prefix + numbering	ds
S-box Exact Transition indicator variables	see Section 2.5	

Table 3.1. Types of variables that can be found in a constraint matrix

3.2 Scheme of Computing a Constraint Matrix

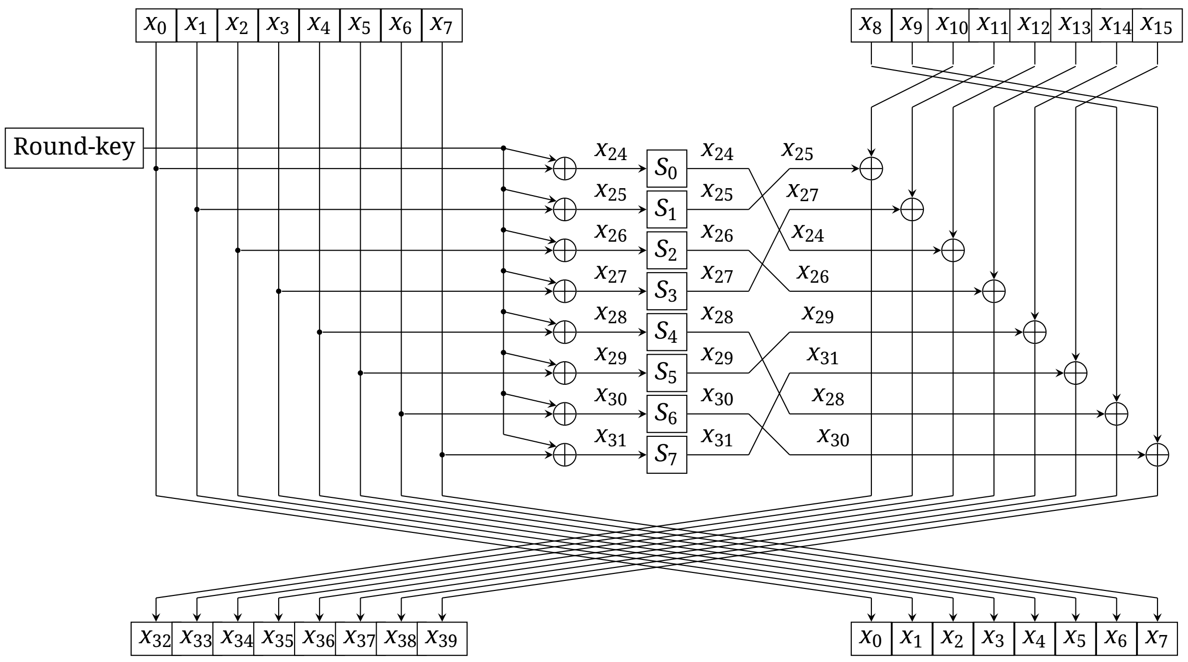
When computing the constraint matrix for a combination of modeling and cipher, the process is as follows: A user has the choice to use the command line questionnaire from `main.py`'s `safe_call()` function or call the process of interest directly. Either way, an instance of the cipher is created by constructing an instance of the corresponding class. In this class, firstly the required resources are determined. These are the number of variables, the different types of which can be found in Table 3.1, that are required for the constraints of the combination of cipher, type of modeling, and number of rounds, as well as the number of constraints required by the word- or bit-oriented modeling part of the MILP. For these k variables, including the right-hand side of the inequality, and n constraints a $n \times k$ matrix is created. During this process, an instance of the `SBox` class is constructed for each type of S-box that can be found in the respective cipher. This means for efficiency's sake, in both AES and Gift64 one instance is constructed while in LBlock eight instances are constructed.

With each round, constraints representing XORs, three-forked branches, and linear transformations for the respective word- or bit-oriented modeling are written into the matrix. Meanwhile, the chosen model's S-box constraints are added to a different matrix which gets appended to the initial one. This is slower than it could be but given that we do not know how many constraints most of the modelings yield until executing them, this saves us some computation and calculation at the initial construction of the cipher instance.

Once all the constraints have been saved into the matrix, i.e. once we have generated them for the number of rounds this instance was designated to run, we use `visualization.py` and functionalities from `sorting_functions.py` to visualize our constraint matrix. We have opted for simply creating a picture with the same dimensions as the resulting constraint matrix and setting a pixel to black iff its entry in the matrix is nonzero. While the details of the exact factor assigned to each variable in a specific constraint are lost, this is not only a computationally feasible approach to creating graphics with at times more than 30.000.000 pixels with limited memory but also easier to analyze when looking for block structures.

With the increase in computational effort in ciphers with larger inputs, S-boxes, and in later modelings, the framework saves files for the convex hull of S-boxes, the resulting matrix, the resulting mapping of variables to matrix columns, and the resulting image in the location from which it is called.

Especially for the purpose of reproducibility of the results, we are going to give a short summary of the technical details of the project. All the implementation and testing were done on an `Ubuntu 23.04` [9] computer using `Python 3.11.2` [26]. For the constraint matrix generation, the libraries `numpy 1.24.2` [13], `Scipy 1.10.1` [34], `itertools 3.11.4`, and `re` are included, as well as `SageMath 9.5` [33] which is its own programming language but for which could import the source code into Python and use it that way. Once the matrices are generated, the libraries `matplotlib`



x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
Round-key
x_{24} x_{25} x_{26} x_{27} x_{28} x_{29} x_{30} x_{31}
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7
x_{25} x_{27} x_{24} x_{26} x_{29} x_{31} x_{28} x_{30}
x_{32} x_{33} x_{34} x_{35} x_{36} x_{37} x_{38} x_{39}
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7

■ Figure 3.2. Visualization of how the variables change in one round of bit-oriented differential cryptanalysis for LBlock. The output of the right-hand XOR operation is not included directly at the operation but down at the output after the permutation as not to make the figure convoluted. Note that the output after the S-box is not a different variable in word-oriented cryptanalysis, we include this change in generalizing bit-oriented cryptanalysis.

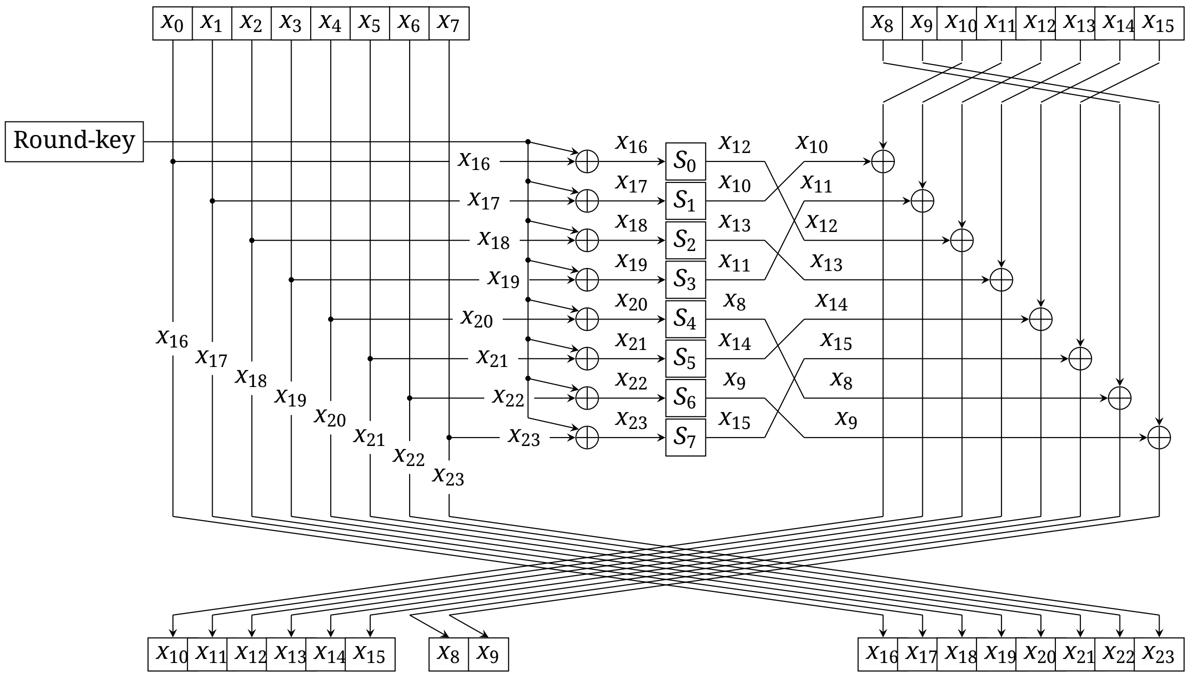
`3.5.2` [15] and `pandas 2.0.2` [22] are used for their visualization. For some further efficiency, such as not having to calculate the same constraints, e.g. the h-representation of the convex hulls over an S-box's feasible transitions, some exporting and importing was done using the `pickle` library. Finally, we used the `unittest` library for the testing of our module.
Versions are not listed for libraries that are part of the Python standard library.

3.3 Implemented Ciphers

The ciphers, for which we implemented the generation of the constraints explained in Chapter 2 are LBlock, Gift64, and AES. LBlock is a Feistel cipher while Gift64 and AES are SPNs.

LBlock. LBlock is a lightweight block cipher encrypting a 64-bit plaintext x to a 64-bit ciphertext $y = LBlock(x)$ using an 80-bit key in a Feistel structure. LBlock can be seen as a 4-bit word-oriented cipher. Aiding the text explaining the cipher's functionality are the figures showing which variables are being assigned to which operations in differential cryptanalysis Figure 3.2 and linear cryptanalysis Figure 3.3 respectively. For simpler visualization, we treat each x_i in said figures as a nibble instead of including all variables.

As a Feistel cipher, LBlock divides the 64-bit plaintext x into two parts x_ℓ, x_r at the beginning, such that their concatenation makes up the original plaintext $x = x_\ell || x_r$. In the i-th round, LBlock first derives a round-key $k^{(i)}$ from the key k. In a round, LBlock's $f(x_\ell)$ function first XORs the input with the round-key $x_\ell \oplus k^{(i)}$. It then funnels the output of said XORs through the cipher's S-boxes, all of which perform different substitutions. The substitutions can be found in Table 3.2. Lastly, the output of the S-boxes is then permuted. Describing the XORing through function x_k, the substitution through function s and the permutation through function p would give us $f(x_\ell) = p(s(x_k(x_\ell)))$. Following


Round-key
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
S_0 S_1 S_2 S_3 S_4 S_5 S_6 S_7
x_{16} x_{17} x_{18} x_{19} x_{20} x_{21} x_{22} x_{23}
x_{10} x_{11} x_{12} x_{13} x_{14} x_{15} x_8 x_9
x_{16} x_{17} x_{18} x_{19} x_{20} x_{21} x_{22} x_{23}

■ Figure 3.3. Visualization of how the variables change in one round of linear cryptanalysis for LBlock. Note that the output after the S-box is not a different variable in word-oriented cryptanalysis, we include this change in generalizing bit-oriented cryptanalysis.

S-box	Substitutions
S_0	E9F0D4AB128376C5
S_1	4BE9FD0A7C562813
S_2	1E7CFD06B593248A
S_3	768B0F3E9ACD5241

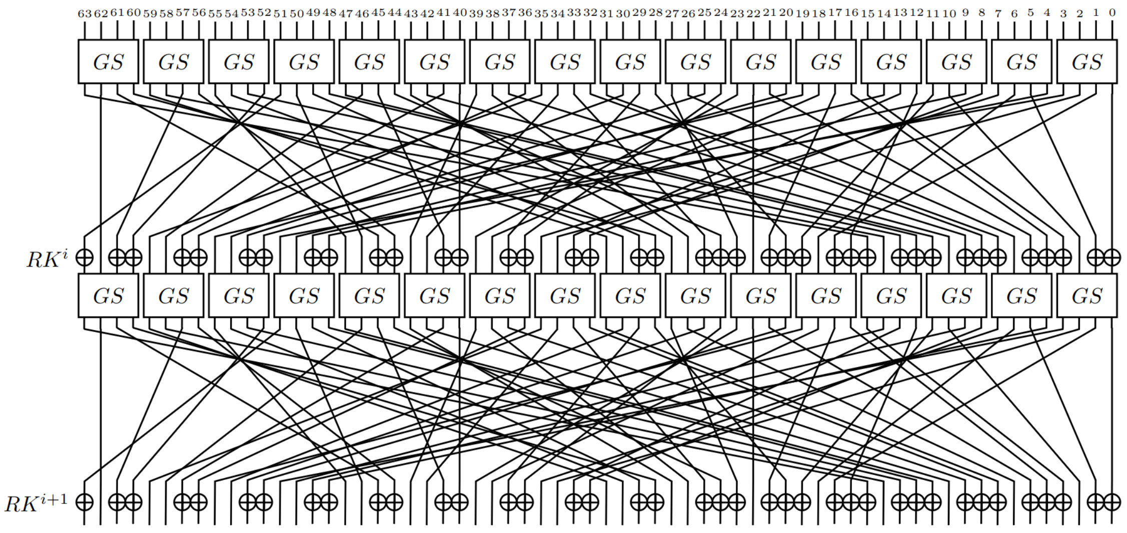
S-box	Substitutions
S_4	E5F072CD1849BA63
S_5	2DBCFE097A631845
S_6	B94E0FAD6C573812
S_7	DAF0E49B218375C6

Table 3.2. Substitutions performed by LBlock's S-boxes.

the Feistel networks structure, we receive an output y_ℓ and y_r that can be defined as $y_r = x_\ell$ and $y_\ell = x_r \oplus f(x_\ell) = x_r \oplus p(s(x_k(x_\ell)))$.

Gift64. Gift64 paragraph the 64-bit plaintext variation of the Gift cipher, introduced in Banik et al. [3]. Gift64 has an SPN structure with first substitution, then permutation and lastly permutations and linear transformations applied to some specific bits, which can also be seen in Figure 3.4. The cipher can only be represented and analyzed in a bit-oriented fashion.
For each round, the following steps happen:

1. The input bits are divided into words of length 4.
2. Each word is then put through an S-box performing 1A4C6F392DB7508E.
3. After the substitution, the outputs are permuted such that the bit that was previously at position i is now at position
$$s\lfloor\frac{i}{16}\rfloor + 16\left(\left(s\lfloor\frac{i \mod 16}{4}\rfloor + (i \mod 4)\right) \mod 4\right) + (i \mod 4).$$
4. Some of the bits are then XORed with bits from the key. The bits which get XORed are $b_{4i+1}, b_{4i} \forall i \in \{0, \dots, 15\}$.



■ Figure 3.4. Visualization of two rounds of Gift64 from Banik et al. [3].

5. Some of the bits that are not XORed with the key then get flipped, i.e. XORed with 1. The bits being XORed are those at the positions $n-1, 23, 19, 15, 11, 7$, and 3.

AES. As a third cipher, we adapted the implementation of Rijndael or the *Advanced Encryption Standard (AES)* from Lohr [20] to fit our new project structure. AES is a 128-bit plaintext to 128-bit ciphertext SPN. As a byte-oriented cipher, it is our cipher with the largest orientation and the only one to use an 8-bit S-box. We just give a short introduction to the cipher, a more detailed description can be found in Daemen [11] or Lohr [20].

Before the start of the first round, the ciphertext is XORed with a sub-key generated from a given 128-bit key. After that, in each round, the following operations are performed:

1. `ByteSub` - where the S-box is applied to each of the 16 byte in AES.

2. `ShiftRows` - which permutes the order of bytes to 1, 6, 11, 16, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12.

3. `MixColumns` - which performs a linear transformation of taking vectors of 4 bytes each, i.e. $(1,6,11,16)^T, (5,10,15,4)^T, (9,14,3,8)^T$, and $(13,2,7,12)^T$, with which $M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$ is multiplied, such that the old entries of the bytes corresponding to the entries of a vector x are now $M \cdot x$. Since bytes are only able to take values between 0 and $2^8 - 1$, this multiplication is performed with each entry in $\mathbb{F}_{2^8}$.

4. `AddRoundKey` - where everything is XORed with a sub-key, different from all others previously generated.

Only the last round does not perform `MixColumns`.

4 Analysis of MILP Modeling Approaches for Differential Cryptanalysis of LBLock

Lastly, we analyze the MILP modeling approaches for differential cryptanalysis of LBLock and come to a conclusion on the results found. For that, this chapter starts off by reviewing how our understanding of the problem, i.e. modeling the security bounds for linear and differential cryptanalysis using MILPs, changed over time and why there is not one clear direction it is developing in. Then, we take a look at the results we got from analyzing the constraint matrices returned by the framework. Finally, concluding this thesis, we go over some open problems that we encountered and that were out of scope for this work.

We find that the underlying structure of the modelings, with the exception of one, does not change over time. Finding the structures in the MILPs is a great result since it means that there is a good chance for a more optimal algorithm to be used in solving rather than e.g. CPLEX [16] used by Mouha et al. [24] which as of now does not mention block structures like n-fold in its documentation. However, at the beginning of the work period for this thesis, we expected to look at the different MILP modelings that have been developed over the last years and see their underlying structures change in a way making them algorithmically easier to solve. Over time, we realized that there are different measures of quality for a MILP, namely:

1. The complexity of generating and then solving the MILP.
2. The quality of the solution provided by the solved MILP

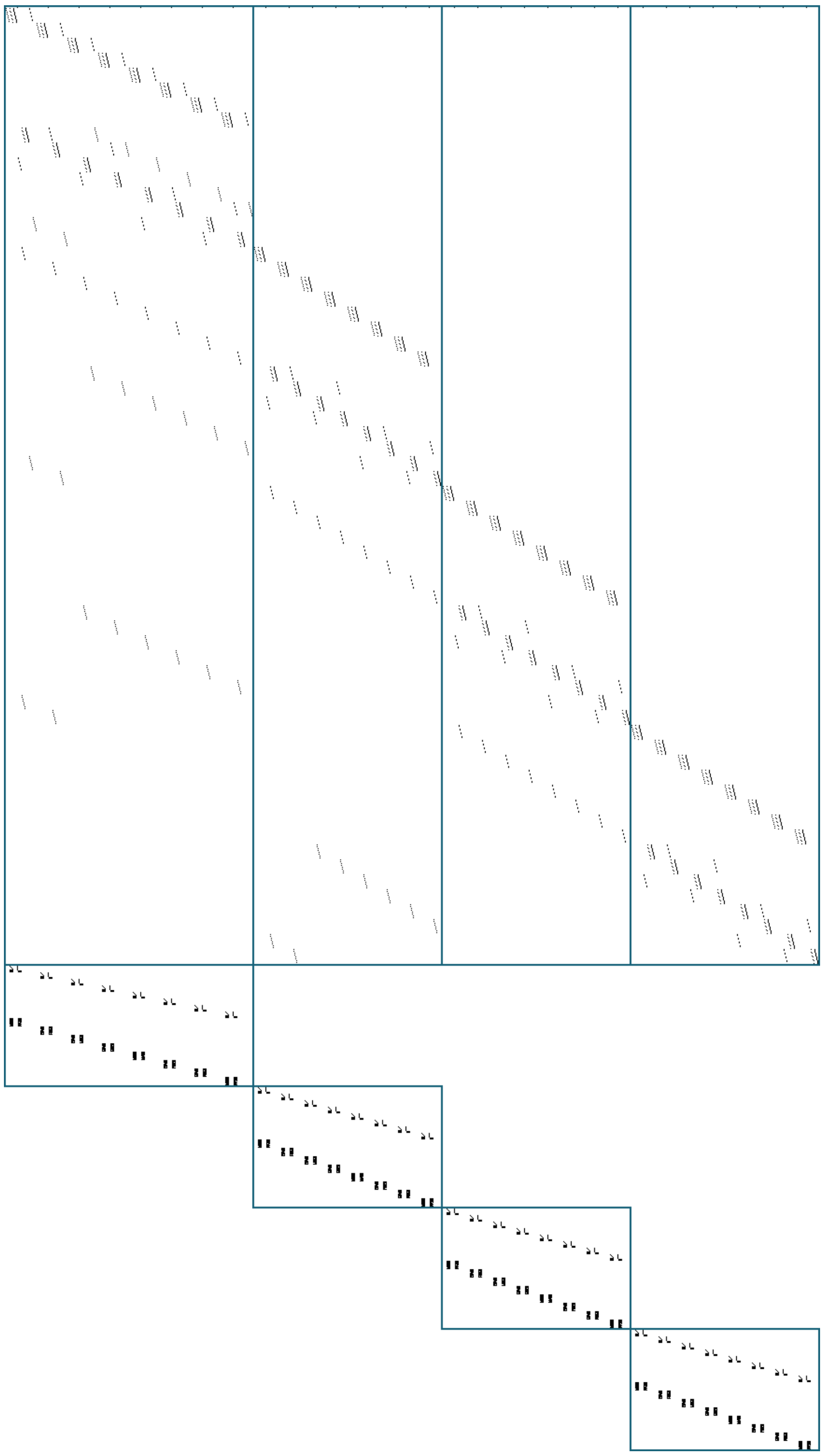
For example, the *exact transition modeling* from Section 2.5 is expected to lead to the most accurate bound while, due to the large number of introduced constraints and variables, also being the most difficult to compute. In comparison, using *logical condition modeling* is expected to compute faster but not necessarily exclude all impossible transitions.

In the following analysis results, we are going to use zero matrices for the sake of readability.

▶ Definition 4.1 (Zeroes matrix $\mathbf{0}^{n\times m}$). *The zeroes matrix $\mathbf{0}^{n\times m}$ is a $n \times m$ matrix from $\mathbb{Z}^{n\times m}$ which has a* 0 *for all of its entries.*

4.1 n-fold MILP for the Convex Hull Approach

Initially, the ASEL framework sorts the variables by their type. The types of variables that can be found in our modelings, as well as their respective naming convention, can be found in Table 3.1. For bit-oriented differential cryptanalysis' LBlock, depending on the modeling used, we found both n-fold or 2-stage structures when diverging from the grouping of variables by type.



■ Figure 4.1. Visualization of the n-fold structure in the constraint matrix of the differential cryptanalysis for 4 rounds of LBlock with greedily chosen convex hull constraints. A pixel is black iff the corresponding matrix entry is non-zero. The blocks are marked using the blue lines.

► Theorem 4.2 (n-fold MILP for the convex hull modeling). *Let $n \geq 4$ be the number of rounds. The MILP for LBlock has an n-fold structure. Our constraint matrix is of the form*

$$\begin{pmatrix} A^{(1)} & A^{(2)} & \cdots & A^{(n-1)} & A^{(n)} \\ B & & & & \\ & B & & & \\ & & \ddots & & \\ & & & B & \\ & & & & B \end{pmatrix}$$

where the matrix is from $\mathbb{Z}^{(376)n+1\times 200n+64}$, with $A^{(1)} \in \mathbb{Z}^{256n+1\times 264}$ and $A^{(2)}, \dots, A^{(n)} \in \mathbb{Z}^{256n\times 200}$, $B \in \mathbb{Z}^{120\times 264}$ for the one vertically aligning with $A^{(1)}$ and $B \in \mathbb{Z}^{120\times 200}$ for the rest.

The overall n-fold structure has been visually emphasized in Figure 4.1 which represents a constraint matrix for the differential cryptanalysis of 4 rounds of LBlock with greedily chosen convex hull constraints. The figure also shows how each of the A blocks is made up of a combination of four different types of sub-matrices. Each A block can also be divided into one of four different types of A blocks. These types are the $A^{(1)}$ block, the $A^{(i)}$ blocks for $i \in \{2, \dots, n-2\}$, the $A^{(n-1)}$ block, and lastly the the $A^{(n)}$ block. In Figure 4.2 it can be seen that this structure is followed for more than 4 rounds of differential cryptanalysis for LBlock as well.

We begin by giving a schematic definition of the A and B blocks which, as seen in Figure 4.1, are made up of smaller building-block matrices.

► Definition 4.3 (A and B blocks). *The four different types of A Blocks are*

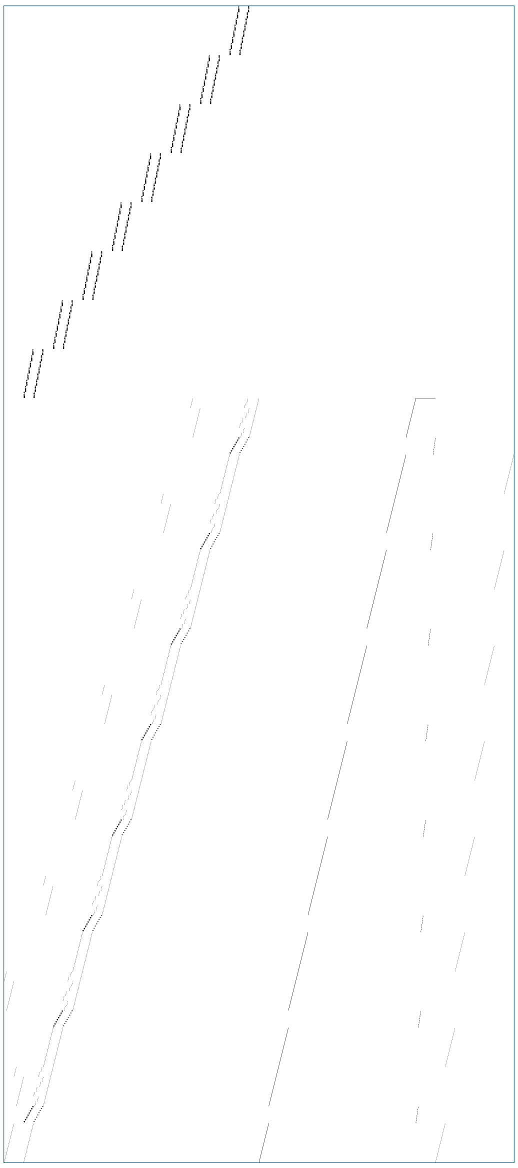
$$A^{(1)} = \begin{pmatrix} M^{(extended)} \\ N \\ P \\ \mathbf{0}^{128\times 264} \\ P^{(extra)} \\ \mathbf{0}^{256(n-3)\times 264} \end{pmatrix}, \quad A^{(i)} = \begin{pmatrix} \mathbf{0}^{256(i-1)\times 200} \\ M \\ N \\ \mathbf{0}^{256\times 200} \\ P \\ \mathbf{0}^{256(n-4)\times 200} \end{pmatrix} \textit{for } i \in \{2, \dots, n-2\}$$

$$A^{(n-1)} = \begin{pmatrix} \mathbf{0}^{256(n-2)\times 200} \\ M \\ N \\ \mathbf{0}^{128\times 200} \end{pmatrix}, \quad A^{(n)} = \begin{pmatrix} \mathbf{0}^{256(n-1)\times 200} \\ M \end{pmatrix}$$

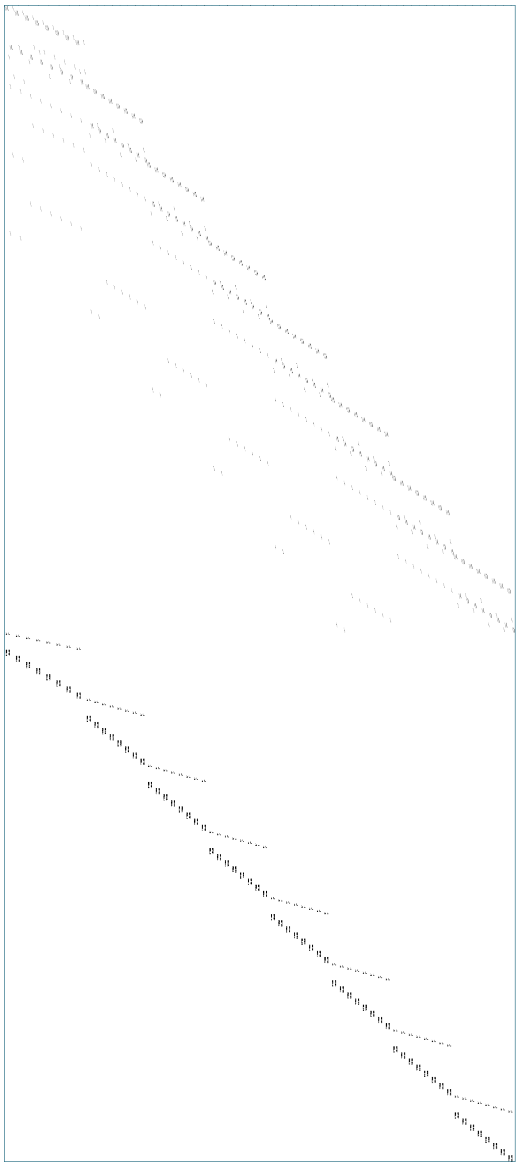
and the B blocks are

$$B = \begin{pmatrix} B^{(bit-oriented)} \\ B^{(modeling)} \end{pmatrix}$$

Each vertically aligned pair of $(A^{(i)}, B)$ blocks, for all $i \in \{1, ..., n\}$, includes the variables introduced in round i. The M and $M^{(extended)}$ blocks are at the height of the round's XOR constraints



■ Figure 4.2. Constraint matrix for differential cryptanalysis of 8 rounds of LBlock with convex hull constraints prior to being rearranged.

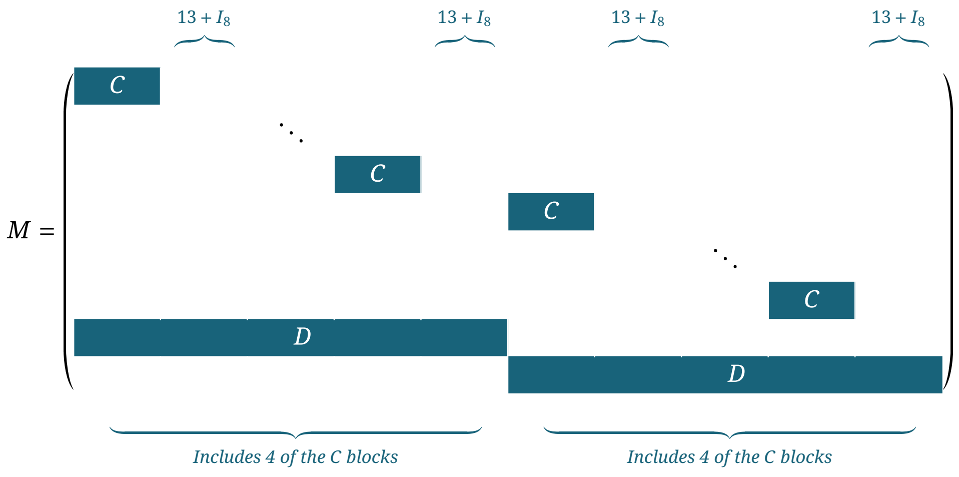


■ Figure 4.3. Constraint matrix for differential cryptanalysis of 8 rounds of LBlock with convex hull constraints after rearrangement to visualize the n-fold structure.

though, from round 2 onwards, they are missing one of the input variables. The missing input variables are found in the N and P blocks of previous rounds, since the output of round i's right-hand XOR becomes an input to round $i+1$'s left-hand XOR as well as an input to round $i+1$'s right-hand XOR. This is the same reason why round $n-1$ and round n do not have both the N and the P blocks, as there are no more XORs to come that require output from these rounds as input. In the same vein, the $A^{(1)}$ block has a second P block since there is no round 0 form which the inputs to the round 2 right-hand XOR could come from and they are instead treated as variables of round 1. To account for these extra variables in the first pair, we include the I_8 notation which equals 8 in $A^{(1)}$ and its corresponding B block and 0 for all others.

The M blocks are divisible further into matrices, which N, P and $M^{(extended)}$ can be constructed out of as well.

▶ Definition 4.4 (M blocks). *Let $M \in \mathbb{Z}^{128\times 200+8I_8}$ be a matrix such that*



The division of M into the C blocks and the D blocks again makes sense given the cipher's structure. The C matrices are each representative of one of the cipher's words from x_ℓ being XORed with a part of the key. The D blocks on the other hand include the constraints of the rounds $f(x_\ell)$ output XORed with x_r.

► Definition 4.5 (C and D blocks). *Let* $C \in \mathbb{Z}^{16\times 12}$, $D \in \mathbb{Z}^{64\times 100+4I_8}$, $D^{(1)},\dots,D^{(4)} \in \mathbb{Z}^{64\times 25+I_8}$, *and* $M^{(\alpha)}, M^{(\delta)}, M^{(\beta)}, M^{(\gamma)} \in \mathbb{Z}^{16\times 4}$ *with*

$$C = \left(\boxed{M^{(\alpha)} \quad M^{(\beta)} \quad M^{(\gamma)}}\right)$$

$$D = \left(\boxed{D^{(1)}}\boxed{D^{(2)}}\boxed{D^{(3)}}\boxed{D^{(4)}}\right)$$

$$D^{(1)} = \left(\begin{array}{ccc} \overbrace{\qquad}^{13} & & \overbrace{\qquad}^{I_8} \\ & \boxed{M^{(\beta)} \quad M^{(\gamma)}} & \\ & \mathbf{0}^{16\times 25} & \\ & \boxed{M^{(\delta)}} & \\ & \mathbf{0}^{16\times 25} & \end{array}\right), \quad D^{(2)} = \left(\begin{array}{ccc} \overbrace{\qquad}^{13} & & \overbrace{\qquad}^{I_8} \\ & \boxed{M^{(\delta)}} & \\ & \boxed{M^{(\beta)}}\boxed{M^{(\gamma)}} & \\ & \mathbf{0}^{32\times 25} & \end{array}\right)$$

$$D^{(3)} = \left(\begin{array}{ccc} \overbrace{\qquad}^{13} & & \overbrace{\qquad}^{I_8} \\ & \mathbf{0}^{32\times 25} & \\ & \boxed{M^{(\beta)}}\boxed{M^{(\gamma)}} & \\ & \boxed{M^{(\delta)}} & \end{array}\right), \quad D^{(4)} = \left(\begin{array}{ccc} \overbrace{\qquad}^{13} & & \overbrace{\qquad}^{I_8} \\ & \mathbf{0}^{16\times 25} & \\ & \boxed{M^{(\delta)}} & \\ & \mathbf{0}^{16\times 25} & \\ & \boxed{M^{(\beta)}}\boxed{M^{(\gamma)}} & \end{array}\right)$$

$$M^{(\alpha)} = \begin{pmatrix} \alpha & & & \\ & \alpha & & \\ & & \alpha & \\ & & & \alpha \end{pmatrix} \qquad \text{with } \alpha = \begin{pmatrix} -1 \\ 0 \\ -1 \\ 0 \end{pmatrix},$$

$$M^{(\beta)} = \begin{pmatrix} \beta & & & \\ & \beta & & \\ & & \beta & \\ & & & \beta \end{pmatrix} \qquad \text{with } \beta = \begin{pmatrix} -1 \\ 0 \\ 0 \\ -1 \end{pmatrix},$$

$$M^{(\gamma)} = \begin{pmatrix} \gamma & & & \\ & \gamma & & \\ & & \gamma & \\ & & & \gamma \end{pmatrix} \qquad \text{with } \gamma = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix},$$

$$M^{(\delta)} = \begin{pmatrix} \delta & & & \\ & \delta & & \\ & & \delta & \\ & & & \delta \end{pmatrix}, \qquad \text{with } \delta = \begin{pmatrix} -1 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

The permutation performed in f is the reason why the structure of the D blocks appears to be more jumbled than the structure of the C blocks. For the XOR constraints in our matrices, $M^{(\alpha)}$ and $M^{(\delta)}$ represent input variables, the $M^{(\beta)}$ represent the output variables, and the $M^{(\gamma)}$ represent the dummy variables.

▸ Definition 4.6 (N, P, and $M^{extended}$ blocks). *Let $N, P, P^{(extra)} \in \mathbb{Z}^{64\times 200+4I_8}$ and $M^{(extended)} \in \mathbb{Z}^{256\times 264}$ with*

$$N = \left(\begin{array}{ccccccc} \overbrace{}^{17} & & \overbrace{}^{21+I_8} & & & & \overbrace{}^{4+I_8} \\ & \boxed{M^{(\delta)}} & & & & & \\ & & & \boxed{M^{(\delta)}} & & & \\ & & & & \ddots & & \\ & & & & & \boxed{M^{(\delta)}} & \\ & & & & \underbrace{}_{5 \text{ more } M^{(\delta)} \text{ parted by } \mathbf{0}^{256\times 21+I_8}} & & \end{array}\right)$$

$$P = \left(\begin{array}{ccccccccc} \overbrace{}^{17+I_8} & & \overbrace{}^{21+I_8} & & \overbrace{}^{21+I_8} & & & & \overbrace{}^{4} \\ & & & & & \boxed{M^{(\alpha)}} & & & \\ & & & & & & \ddots & & \\ & & & & & & & \boxed{M^{(\alpha)}} & \\ & \boxed{M^{(\alpha)}} & & & & & & & \\ & & & \boxed{M^{(\alpha)}} & & & & & \\ & & & & & & \underbrace{}_{4 \text{ more } M^{(\alpha)} \text{ parted by } \mathbf{0}^{256\times 17+I_8}} & & \end{array}\right)$$

$$P^{(extra)} = \left(\begin{array}{ccccccccc} \overbrace{}^{17} & & \overbrace{}^{21+I_8} & & \overbrace{}^{21+I_8} & & & & \overbrace{}^{4+I_8} \\ & & & & & \boxed{M^{(\alpha)}} & & & \\ & & & & & & \ddots & & \\ & & & & & & & \boxed{M^{(\alpha)}} & \\ & \boxed{M^{(\alpha)}} & & & & & & & \\ & & & \boxed{M^{(\alpha)}} & & & & & \\ & & & & & & \underbrace{}_{4 \text{ more } M^{(\alpha)} \text{ parted by } \mathbf{0}^{256\times 17+I_8}} & & \end{array}\right)$$

$$M^{(extended)} = M + \begin{pmatrix} N \\ P \end{pmatrix}$$

With this, construction of the A blocks is done. Continuing on to the B blocks, a division again simplifies both explanation as well as definition. The $B^{(bit-oriented)}$ blocks include the constraints introduced for the general bit-oriented modeling, while the $B^{(modeling)}$ blocks include the constraint of the additional modeling of the convex hull constraints.

▶ Definition 4.7 (B sub-blocks). *Let $B^{(bit-oriented)} \in \mathbb{Z}^{56\times 200+8I_8}$ and $B^{(modeling)} \in \mathbb{Z}^{64\times 200+8I_8}$ be matrices with*

$$B^{(bit-oriented)} = \left(\begin{array}{ccccccc} \overbrace{\qquad}^{4} & & \overbrace{\qquad}^{12+I_8} & & & & \overbrace{\qquad}^{8+I_8} \\ & \boxed{S} & & & & & \\ & & & \boxed{S} & & & \\ & & & & \ddots & & \\ & & & & & \boxed{S} & \\ & & & & \underbrace{\qquad\qquad\qquad}_{5 \text{ more } S \text{ parted by } \mathbf{0}^{256\times 12+I_8}} & & \end{array}\right)$$

$$B^{(modeling)} = \left(\begin{array}{ccccccc} \overbrace{\qquad}^{4} & & \overbrace{\qquad}^{12+I_8} & & & & \overbrace{\qquad}^{8+I_8} \\ & \boxed{S^{(1)}} & & & & & \\ & & & \boxed{S^{(2)}} & & & \\ & & & & \ddots & & \\ & & & & & \boxed{S^{(8)}} & \\ & & & & \underbrace{\qquad\qquad\qquad}_{5 \text{ more } S^{(i)} \text{ with } i \in \{3,\dots,7\} \text{ parted by } \mathbf{0}^{256\times 12+I_8}} & & \end{array}\right)$$

Furthermore, let $S \in \mathbb{Z}^{7\times 13}$ and $S^{(1)}, \dots, S^{(8)} \in \mathbb{Z}^{8\times 13}$ be matrices with

$$S = \left(\begin{array}{ccccccccccccc} & & & & & \overbrace{\qquad\qquad}^{4} & & & & & & & \\ -1 & & & & & & & & 1 & & & & \\ & -1 & & & & & & & 1 & & & & \\ & & -1 & & & & & & 1 & & & & \\ & & & -1 & & & & & 1 & & & & \\ 1 & 1 & 1 & 1 & & & & & -2 & & & & \\ -1 & -1 & -1 & -1 & & & & & & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & & & & & & -1 & -1 & -1 & -1 \end{array}\right)$$

$$S^{(1)} = \begin{pmatrix} -3 & 6 & -5 & -13 & \overbrace{\qquad\qquad}^{4} & -9 & -7 & -1 & 4 \\ 4 & -2 & -5 & 8 & & 7 & -1 & 3 & -1 \\ -3 & -7 & 4 & -1 & & -2 & 6 & 3 & 1 \\ 5 & 1 & 2 & 1 & & 4 & 2 & -3 & -2 \\ -3 & 2 & 1 & -1 & & -2 & -2 & & -1 \\ -1 & -3 & 5 & 4 & & & 2 & 1 & 4 \\ -1 & 2 & -4 & -4 & & -3 & -2 & -3 & 4 \\ 1 & & -2 & 1 & & & 1 & 2 & -2 \end{pmatrix}, \quad S^{(2)} = \begin{pmatrix} -3 & 6 & 5 & 13 & \overbrace{\qquad\qquad}^{4} & -9 & 7 & 4 & -1 \\ 6 & -3 & 4 & -5 & & 7 & -2 & -5 & 1 \\ -2 & -1 & -3 & -1 & & -3 & -2 & & -1 \\ 2 & & 1 & -4 & & 3 & 1 & 1 & 1 \\ & -2 & -1 & 2 & & 2 & -3 & -1 & 1 \\ -4 & 3 & -2 & 1 & & -3 & 4 & -2 & 1 \\ 6 & 1 & -2 & -2 & & 5 & -3 & -2 & -4 \\ -1 & -3 & -1 & & & & -2 & & 1 \end{pmatrix}$$

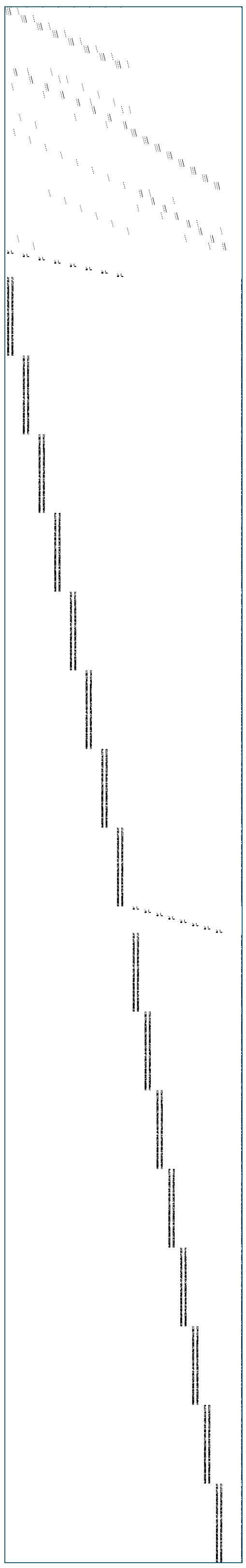
$$S^{(3)} = \begin{pmatrix} -3 & 6 & 5 & 13 & \overbrace{\qquad\qquad}^{4} & -1 & -9 & 4 & 7 \\ 6 & -3 & 4 & -5 & & 1 & 7 & -5 & -2 \\ -2 & -1 & -3 & -1 & & -1 & -3 & & -2 \\ 2 & & 1 & -4 & & 1 & 3 & 1 & 1 \\ & -2 & -1 & 2 & & 1 & 2 & -1 & -3 \\ -4 & 3 & -2 & 1 & & 1 & -3 & -2 & 4 \\ 6 & 1 & -2 & -2 & & -4 & 5 & -2 & -3 \\ -1 & -3 & -1 & & & 1 & & & -2 \end{pmatrix} \quad S^{(4)} = \begin{pmatrix} -3 & -6 & 5 & 13 & \overbrace{\qquad\qquad}^{4} & -7 & -1 & -9 & 4 \\ 6 & 3 & 4 & -5 & & 2 & 1 & 7 & -5 \\ -2 & 1 & -3 & -1 & & 2 & -1 & -3 & \\ 2 & & 1 & -4 & & -1 & 1 & 3 & 1 \\ & 2 & -1 & 2 & & 3 & 1 & 2 & -1 \\ -4 & -3 & -2 & 1 & & -4 & 1 & -3 & -2 \\ 6 & -1 & -2 & -2 & & 3 & -4 & 5 & -2 \\ -1 & 2 & & 1 & & 2 & 1 & & 1 \end{pmatrix}$$

$$S^{(5)} = \begin{pmatrix} -3 & 6 & -5 & -13 & \overbrace{\qquad\qquad}^{4} & -1 & -9 & -7 & 4 \\ 4 & -2 & -5 & 8 & & 3 & 7 & -1 & -1 \\ -3 & -7 & 4 & -1 & & 3 & -2 & 6 & 1 \\ 5 & 1 & 2 & 1 & & -3 & 4 & 2 & -2 \\ -3 & 2 & 1 & -1 & & & -2 & -2 & -1 \\ -1 & -3 & 5 & 4 & & 1 & & 2 & 4 \\ -1 & 2 & -4 & -4 & & -3 & -3 & -2 & 4 \\ 1 & & -2 & 1 & & 2 & & 1 & -2 \end{pmatrix} \quad S^{(6)} = \begin{pmatrix} -3 & 6 & 5 & 13 & \overbrace{\qquad\qquad}^{4} & -9 & -1 & 7 & 4 \\ 6 & -3 & 4 & -5 & & 7 & 1 & -2 & -5 \\ -2 & -1 & -3 & -1 & & -3 & -1 & -2 & \\ 2 & & 1 & -4 & & 3 & 1 & 1 & 1 \\ & -2 & -1 & 2 & & 2 & 1 & -3 & -1 \\ -4 & 3 & -2 & 1 & & -3 & 1 & 4 & -2 \\ 6 & 1 & -2 & -2 & & 5 & -4 & -3 & -2 \\ -1 & -3 & -1 & & & & 1 & -2 & \end{pmatrix}$$

$$S^{(7)} = \begin{pmatrix} -3 & -6 & 5 & 13 & \overbrace{\qquad\qquad}^{4} & -9 & -7 & 4 & -1 \\ 6 & 3 & 4 & -5 & & 7 & 2 & -5 & 1 \\ -2 & 1 & -3 & -1 & & -3 & 2 & & -1 \\ 2 & & 1 & -4 & & 3 & -1 & 1 & 1 \\ & 2 & -1 & 2 & & 2 & 3 & -1 & 1 \\ -4 & -3 & -2 & 1 & & -3 & -4 & -2 & 1 \\ 6 & -1 & -2 & -2 & & 5 & 3 & -2 & -4 \\ -1 & 2 & & 1 & & & 2 & 1 & 1 \end{pmatrix} \quad S^{(8)} = \begin{pmatrix} -3 & 6 & -5 & -13 & \overbrace{\qquad\qquad}^{4} & -9 & -7 & 4 & -1 \\ 4 & -2 & -5 & 8 & & 7 & -1 & -1 & 3 \\ -3 & -7 & 4 & -1 & & -2 & 6 & 1 & 3 \\ 5 & 1 & 2 & 1 & & 4 & 2 & -2 & -3 \\ -3 & 2 & 1 & -1 & & -2 & -2 & -1 & \\ -1 & -3 & 5 & 4 & & & 2 & 4 & 1 \\ -1 & 2 & -4 & -4 & & -3 & -2 & 4 & -3 \\ 1 & & -2 & 1 & & & 1 & -2 & 2 \end{pmatrix}$$

The differences between the $S^{(i)}$ blocks come from the fact that each of the 8 different S-boxes used in LBlock has a different set of convex hull constraints and each $S^{(i)}$ is constructed in accordance with its corresponding ith S-box. We do not see a change in the general constraints in S since these constraints are the same for every 4×4 bijective S-box with a branch number equaling 2. The n-fold structure prevails with all convex hull constraints, not reduced by the greedy choice, i.e. 166 per S-box instead of 8, which can be seen in Figure 4.4.

The previously defined A blocks are entirely made up of the constraints from bit-oriented modeling and reoccur in the LBlock constraint matrices for all bit-oriented modelings of differential cryptanalysis. Meanwhile, the B block is made up of the constraints restricting a round's S-box transitions, and while only the $B^{(bit-oriented)}$ blocks are found in all bit-oriented modelings of differential cryptanalysis, the $B^{(modeling)}$ blocks change for every modeling. Now, given that the convex hull modeling contributes constraints for each S-box only made up of their input and output variables, we can generalize this n-fold structure to other modelings which do the same.



■ Figure 4.4. Constraint matrix for 2 rounds differential cryptanalysis LBlock with convex hull constraints that have not been reduced using the greedy algorithm from Algorithm 1.

4.2 n-fold MILP for the Logical Condition Modeling

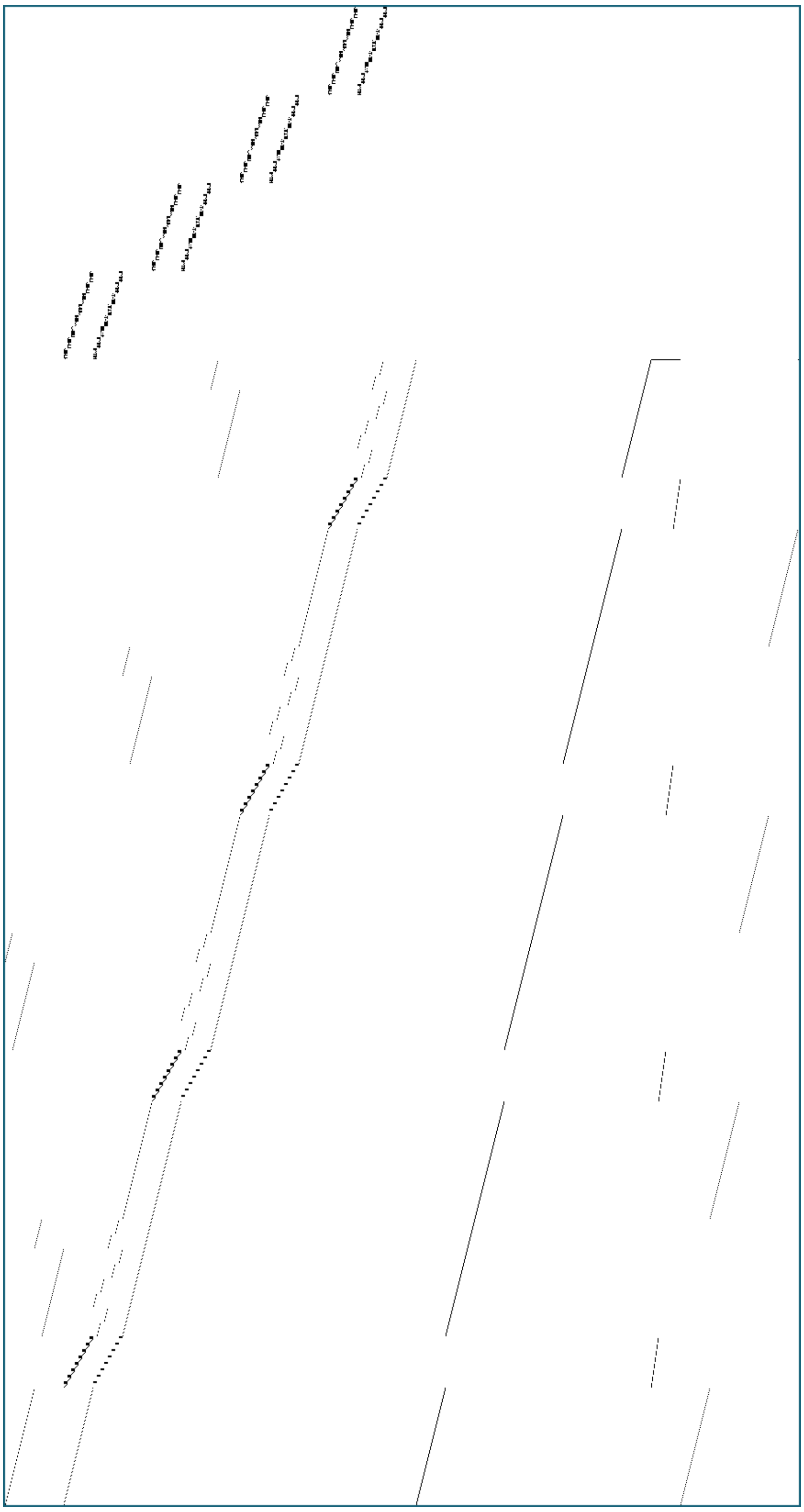
Following the discovery that we can generalize the previously defined n-fold structure to other modelings which only introduce constraints on the S-box input and output variables, we find an n-fold structure in the constraint matrix for the differential cryptanalysis of LBlock using logical condition modeling. The similarity in the structure between the modelings is exemplified when looking at the unsorted and sorted constraint matrices in Figure 4.5 and Figure 4.6 respectively.

► Theorem 4.8 (n-fold MILP for the logical condition modeling). *Let $n \geq 4$ be the number of rounds. The MILP for LBlock has an n-fold structure, i.e. we have*

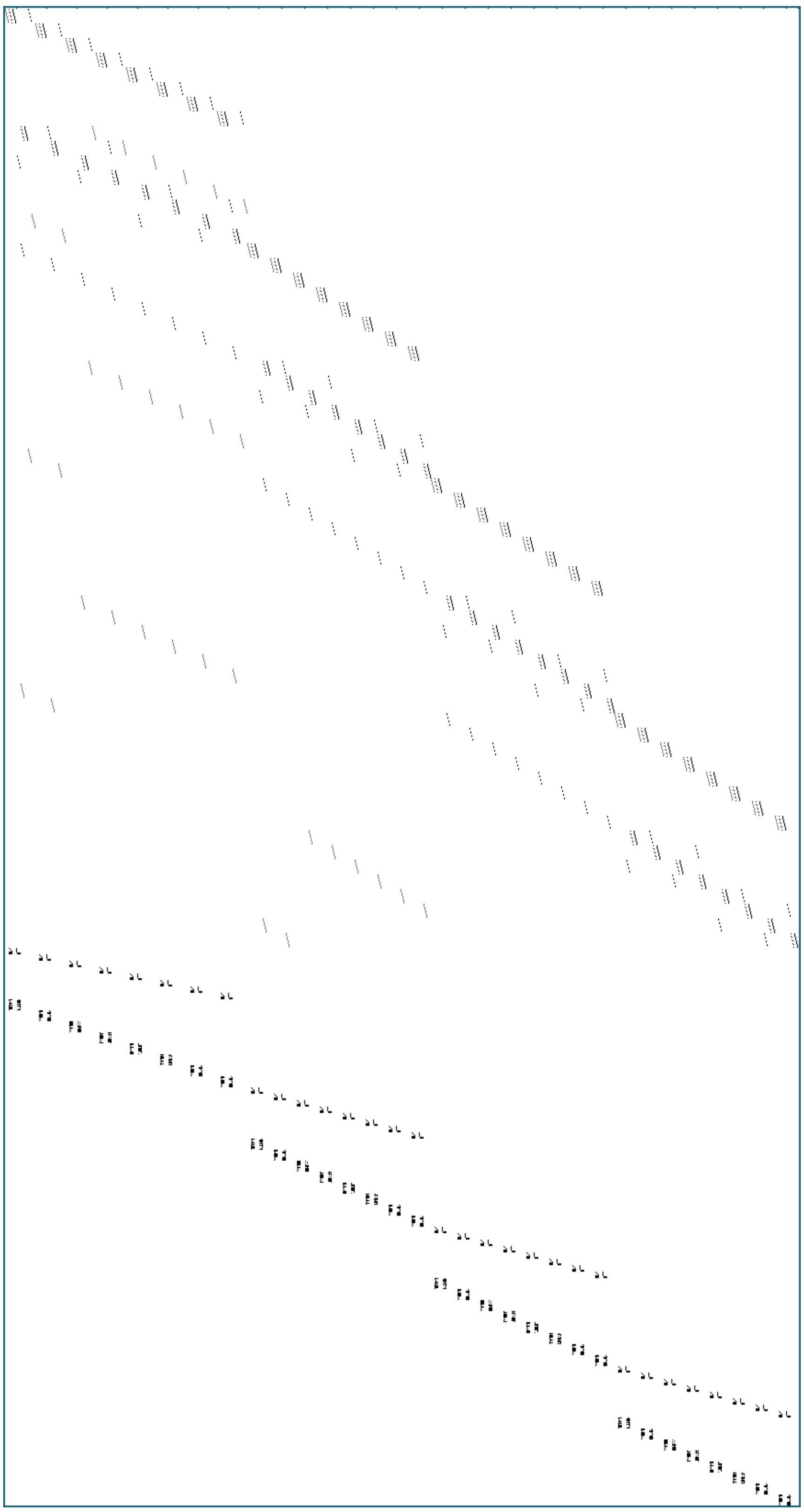
$$\begin{pmatrix} A^{(1)} & A^{(2)} & \cdots & A^{(n-1)} & A^{(n)} \\ B & & & & \\ & B & & & \\ & & \ddots & & \\ & & & B & \\ & & & & B \end{pmatrix}$$

where the matrix is from $\mathbb{Z}^{408n+1\times 200n+64}$, with $A^{(1)} \in \mathbb{Z}^{256n+1\times 264}$ and $A^{(2)},\dots,A^{(n)} \in \mathbb{Z}^{256n\times 200}$, $B \in \mathbb{Z}^{152\times 264}$ for the one vertically aligning with $A^{(1)}$ and $B \in \mathbb{Z}^{120\times 200}$ for $i \neq 1$.

The A blocks are the same as in the convex hull approach in Definition 4.3. The definition for B blocks hold too, although the small $S^{(i)}$ blocks with $i \in \{1,\dots,8\}$ in the $B^{(modeling)}$ block need to be redefined.



■ Figure 4.5. Constraint matrix for 4 rounds differential cryptanalysis LBlock with logical condition modeling constraints prior to begin rearranged.



■ Figure 4.6. Constraint matrix for 4 rounds differential cryptanalysis LBlock with logical condition modeling constraints after begin rearranged.

▶ Definition 4.9 ($S^{(i)}$ blocks). *Let $S^{(i)} \in \mathbb{Z}^{12\times 13}$ for $i \in \{1,\dots,8\}$ be the block representing the constraints from logical condition modeling added for a round's i-th S-box.*

$$S^{(1)} = \left(\begin{array}{ccccccccccccc}
 & & & & \multicolumn{5}{c}{\overbrace{\qquad\qquad\qquad}^{5}} & & & & \\
 & & & 1 & & & & & & -1 & -1 & -1 & 1 \\
 & & 1 & & & & & & & -1 & -1 & 1 & -1 \\
 & & & -1 & & & & & & -1 & -1 & 1 & -1 \\
 & & -1 & & & & & & & 1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & & & & & & -1 & & & \\
1 & 1 & -1 & -1 & & & & & & & 1 & & \\
-1 & 1 & -1 & -1 & & & & & & 1 & & & \\
1 & 1 & -1 & 1 & & & & & & & -1 & & \\
 & & 1 & & & & & & & 1 & -1 & 1 & -1 \\
 & & & 1 & & & & & & -1 & -1 & 1 & 1 \\
-1 & -1 & -1 & 1 & & & & & & & 1 & & \\
1 & -1 & -1 & -1 & & & & & & 1 & & &
\end{array}\right)$$

$$S^{(2)} = \left(\begin{array}{ccccccccccccc}
 & & & & \multicolumn{5}{c}{\overbrace{\qquad\qquad\qquad}^{5}} & & & & \\
-1 & 1 & -1 & -1 & & & & & & & 1 & & \\
 & & & 1 & & & & & & -1 & -1 & -1 & 1 \\
 & & 1 & & & & & & & -1 & -1 & 1 & -1 \\
 & & & -1 & & & & & & -1 & -1 & 1 & -1 \\
1 & 1 & -1 & 1 & & & & & & -1 & & & \\
1 & 1 & -1 & -1 & & & & & & 1 & & & \\
1 & 1 & -1 & -1 & & & & & & & -1 & & \\
 & & -1 & & & & & & & -1 & 1 & -1 & -1 \\
 & & & 1 & & & & & & -1 & -1 & 1 & 1 \\
1 & -1 & -1 & -1 & & & & & & & 1 & & \\
 & & 1 & & & & & & & -1 & 1 & 1 & -1 \\
-1 & -1 & -1 & 1 & & & & & & 1 & & &
\end{array}\right)$$

$$S^{(3)} = \left(\begin{array}{cccccccccc}
 & & & & \overbrace{\qquad\qquad}^{5} & & & & \\
-1 & 1 & -1 & -1 & & & 1 & & \\
1 & 1 & -1 & 1 & & & & & -1 \\
 & & & 1 & & 1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 & & & -1 & & \\
1 & 1 & -1 & -1 & & & & & 1 \\
 & & 1 & & & 1 & -1 & -1 & -1 \\
 & & & -1 & & 1 & -1 & -1 & -1 \\
1 & -1 & -1 & -1 & & & 1 & & \\
 & & -1 & & & -1 & 1 & -1 & -1 \\
 & & 1 & & & 1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & & & & & 1 \\
 & & & 1 & & -1 & -1 & 1 & -1
\end{array}\right)$$

$$S^{(4)} = \left(\begin{array}{cccccccccc}
 & & & & \overbrace{\qquad\qquad}^{5} & & & & \\
-1 & -1 & -1 & 1 & & & & 1 & \\
 & & & 1 & & -1 & 1 & -1 & -1 \\
1 & 1 & -1 & 1 & & & & -1 & \\
 & & -1 & & & 1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & & 1 & & & \\
 & & 1 & & & 1 & -1 & -1 & 1 \\
 & & 1 & & & -1 & -1 & -1 & 1 \\
 & & & -1 & & -1 & -1 & -1 & 1 \\
 & & & 1 & & -1 & 1 & -1 & 1 \\
1 & 1 & -1 & -1 & & -1 & & & \\
1 & 1 & -1 & -1 & & & & 1 & \\
1 & -1 & -1 & -1 & & 1 & & &
\end{array}\right)$$

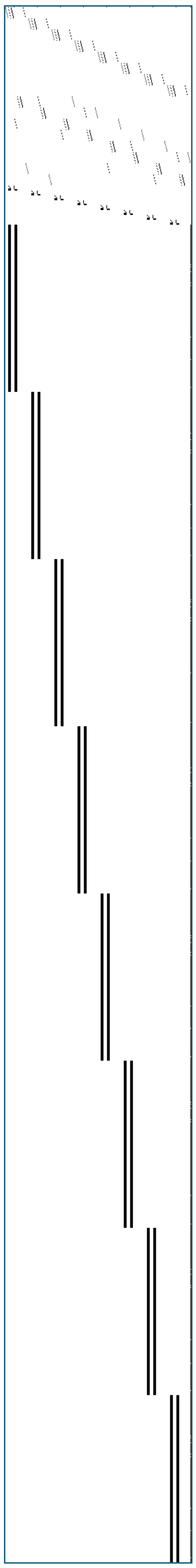
$$S^{(5)} = \left(\begin{array}{cccccccccc}
 & & & & \overbrace{\qquad\qquad}^{5} & & & & \\
1 & 1 & -1 & 1 & & & & & -1 \\
 & & 1 & & & -1 & 1 & -1 & -1 \\
 & & & -1 & & -1 & 1 & -1 & -1 \\
 & & -1 & & & 1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & & 1 & & & \\
1 & 1 & -1 & -1 & & -1 & & & \\
1 & 1 & -1 & -1 & & & & & 1 \\
 & & & 1 & & -1 & 1 & 1 & -1 \\
 & & 1 & & & 1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & & & & & 1 \\
 & & & 1 & & -1 & -1 & 1 & -1 \\
1 & -1 & -1 & -1 & & 1 & & &
\end{array}\right)$$

$$S^{(6)} = \begin{array}{ccccccccc}
 & & & & \overbrace{\qquad}^{5} & & & & \\
\left(\begin{array}{ccccccccc}
-1 & -1 & -1 & 1 & & & & 1 & \\
 & & & 1 & & -1 & -1 & -1 & 1 \\
1 & 1 & -1 & 1 & & & & -1 & \\
 & & 1 & & & -1 & 1 & -1 & -1 \\
 & & & -1 & & -1 & 1 & -1 & -1 \\
 & & -1 & & & 1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & & 1 & & & \\
 & & 1 & & & 1 & 1 & -1 & -1 \\
 & & & 1 & & -1 & 1 & -1 & 1 \\
1 & 1 & -1 & -1 & & -1 & & & \\
1 & 1 & -1 & -1 & & & & 1 & \\
1 & -1 & -1 & -1 & & 1 & & &
\end{array}\right)
\end{array}$$

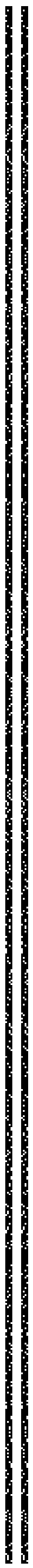
$$S^{(7)} = \begin{array}{ccccccccc}
 & & & & \overbrace{\qquad}^{5} & & & & \\
\left(\begin{array}{ccccccccc}
-1 & 1 & -1 & -1 & & & 1 & & \\
 & & & 1 & & -1 & -1 & -1 & 1 \\
 & & 1 & & & -1 & -1 & 1 & -1 \\
 & & & -1 & & -1 & -1 & 1 & -1 \\
1 & 1 & -1 & 1 & & -1 & & & \\
1 & 1 & -1 & -1 & & 1 & & & \\
1 & 1 & -1 & -1 & & & -1 & & \\
 & & -1 & & & -1 & 1 & -1 & -1 \\
 & & & 1 & & -1 & -1 & 1 & 1 \\
1 & -1 & -1 & -1 & & & 1 & & \\
 & & 1 & & & -1 & 1 & 1 & -1 \\
-1 & -1 & -1 & 1 & & 1 & & &
\end{array}\right)
\end{array}$$

$$S^{(8)} = \begin{array}{ccccccccc}
 & & & & \overbrace{\qquad}^{5} & & & & \\
\left(\begin{array}{ccccccccc}
-1 & 1 & -1 & -1 & & & 1 & & \\
 & & & 1 & & -1 & -1 & -1 & 1 \\
 & & 1 & & & -1 & -1 & 1 & -1 \\
 & & & -1 & & -1 & -1 & 1 & -1 \\
1 & 1 & -1 & 1 & & -1 & & & \\
1 & 1 & -1 & -1 & & 1 & & & \\
1 & 1 & -1 & -1 & & & -1 & & \\
 & & -1 & & & -1 & 1 & -1 & -1 \\
 & & & 1 & & -1 & -1 & 1 & 1 \\
1 & -1 & -1 & -1 & & & 1 & & \\
 & & 1 & & & -1 & 1 & 1 & -1 \\
-1 & -1 & -1 & 1 & & 1 & & &
\end{array}\right)
\end{array}$$

Where the pattern of $\delta_0, \ldots, \delta_3$ implying δ is nicely seen thanks to the divide in the middle.



■ Figure 4.7. The constraint matrix of a differential cryptanalysis for 1 round LBlock using the exclusion of impossible transitions modeling.



■ Figure 4.8. The scheme of a single $S^{(1)}$ block from the constraint matrix of a differential cryptanalysis for LBlock using the efficient logical condition techniques.

4.3 n-fold MILP for the Exclusion of Impossible Transitions Modeling and the Efficient Logical Condition Modeling

Following the discovery of the generalization again, we find that both the MILPs resulting from using the exclusion of impossible transitions modeling and efficient logical condition modeling are n-fold MILPS.

► **Theorem 4.10 (n-fold MILP for the exclusion of impossible transitions modeling).** *Let $n \geq 4$ be the number of rounds. The MILP for LBlock has an n-fold structure, i.e. we have*

$$\begin{pmatrix} A^{(1)} & A^{(2)} & \cdots & A^{(n-1)} & A^{(n)} \\ B & & & & \\ & B & & & \\ & & \ddots & & \\ & & & B & \\ & & & & B \end{pmatrix}$$

where the matrix is from $\mathbb{Z}^{2176n+1\times 200n+64}$, with $A^{(1)} \in \mathbb{Z}^{256n+1\times 264}$ and $A^{(2)}, \ldots, A^{(n)} \in \mathbb{Z}^{256n\times 200}$, $B \in \mathbb{Z}^{1920\times 264}$ for the one vertically aligning with $A^{(1)}$ and $B \in \mathbb{Z}^{1920\times 200}$ for $i \neq 1$.

And as in Section 4.2, the new constraints are captured in redefining the $S^{(i)}$ blocks. However, for the exclusion of impossible transitions modeling, each $S^{(i)}$ would be from $\mathbb{Z}^{240\times 12}$. This is simply too large for us to properly depict on a reasonable scale, therefore we will not define them here.

Nonetheless, in Figure 4.7 we have visualized the constraint matrix for a 1 round LBlock differential cryptanalysis using the exclusion of impossible transitions modeling. The reason why these constraints are just squares is that each constraint from the exclusion of impossible transitions modeling $S^{(1)}$ blocks includes all input and output variables from the S-box whose transitions are being modeled.

In their paper, Boura and Coggia [8] do not include all of the constraints found using Algorithm 2. They use an algorithm by Abdelkhalek, Sasaki, Todo, Tolba, and Youssef [1] to narrow down the number of constraints. As the inclusion of said algorithm was out of scope for this work, we are looking at all constraints provided and not the narrowed-down version. Given that all the constraints are both in the same block and handle the same variables, an n-fold structure can be found in both. However, since we have not implemented the creation of the constraint matrix with the narrowed down set of constraints, we will only define the structure for all the constraints.

► **Theorem 4.11 (n-fold MILP for efficient logical condition modeling).** *Let $n \geq 4$ be the number of rounds. The MILP for LBlock has an n-fold structure, i.e. we have*

$$\begin{pmatrix} A^{(1)} & A^{(2)} & \cdots & A^{(n-1)} & A^{(n)} \\ B & & & & \\ & B & & & \\ & & \ddots & & \\ & & & B & \\ & & & & B \end{pmatrix}$$

where the matrix is from $\mathbb{Z}^{7424n+1\times 200n+64}$, with $A^{(1)} \in \mathbb{Z}^{256n+1\times 264}$ and $A^{(2)}, \ldots, A^{(n)} \in \mathbb{Z}^{256n\times 200}$, $B \in \mathbb{Z}^{7168\times 264}$ for the one vertically aligning with $A^{(1)}$ and $B \in \mathbb{Z}^{7168\times 200}$ for $i \neq 1$.

Similar to the exclusion of impossible transitions modeling, this is simply too large for us to properly depict on a reasonable scale. With each $S^{(i)}$ being from $\mathbb{Z}^{896\times 12}$, we will not define them here. However, in Figure 4.8 the scheme of an efficient logical condition techniques $S^{(i)}$ block can be seen.

4.4 2-stage for the Exact Transition Modeling

For the MILP representing differential cryptanalysis for LBlock using exact transition modeling, we found an extended two-stage structure.

▸ Theorem 4.12 (2-stage MILP for the exact transition modeling). *The MILP for differential cryptanalysis on a n round LBlock with exact transition modeling constraints is an extended 2-stage MILP*

$$\begin{pmatrix} C & & & & \\ A^{(1)} & B & & & \\ A^{(2)} & & B & & \\ \vdots & & & \ddots & \\ A^{(8n)} & & & & B \end{pmatrix}$$

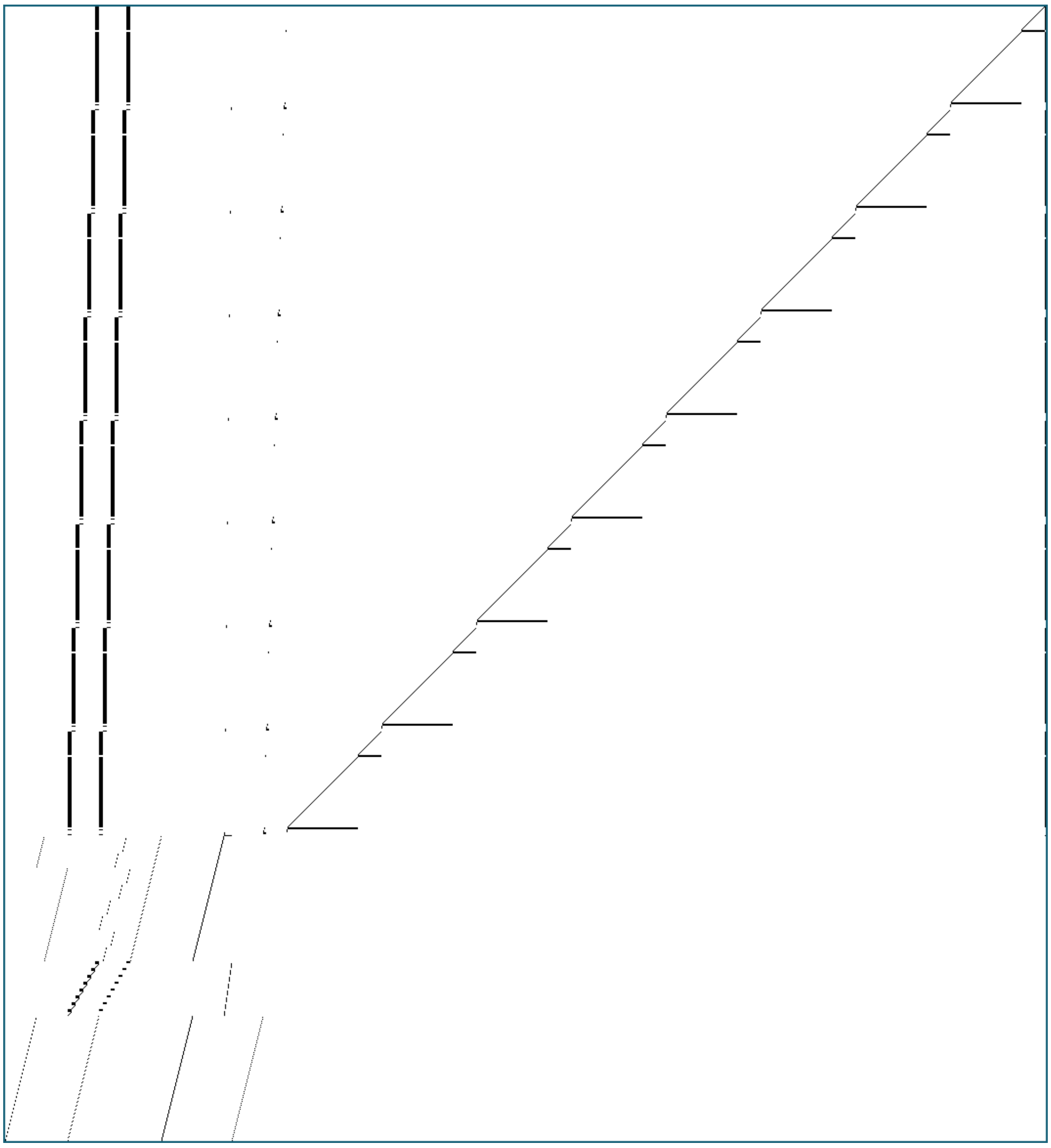
where the matrix is from $\mathbb{Z}^{1104n+1\times 1084n+64}$, with $C \in \mathbb{Z}^{256n+1\times 200n+64}$, $A^{(1)}, \dots, A^{(8n)} \in \mathbb{Z}^{106\times 200n+64}$, and $B \in \mathbb{Z}^{106\times 98}$.

This block structure can be seen, and has been highlighted, in Figure 4.10, although the diagonal B block structure became rather clear in the constraint matrices initial arrangement as seen in Figure 4.9. When looking closely, a strong similarity with the previous n-fold becomes clear. In fact, when analyzing just the C and A blocks, an n-fold structure can be seen.

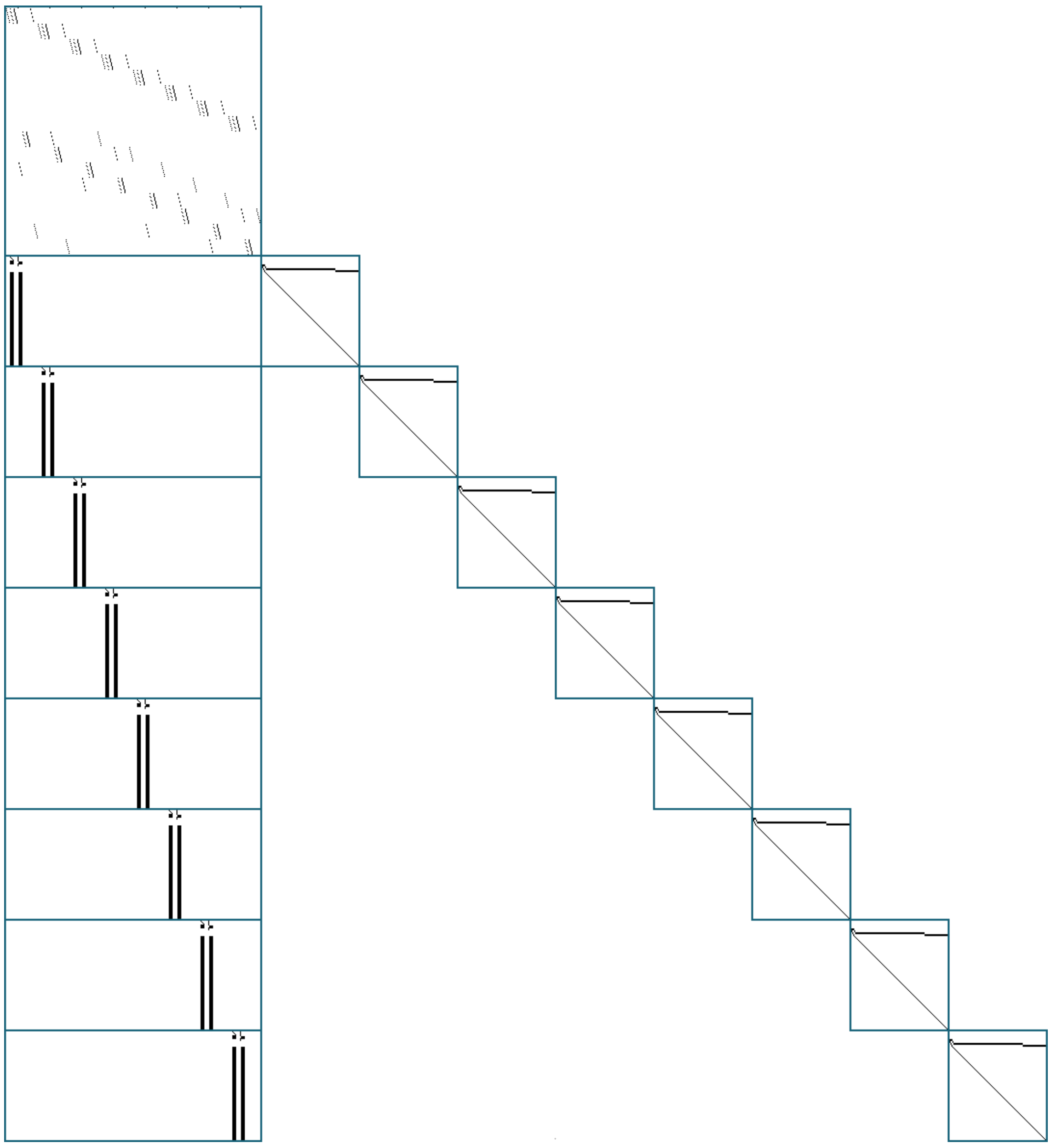
▸ Definition 4.13 (C block). *Let $A^{(1)}_{n-fold}, \dots, A^{(n)}_{n-fold}$ be the A blocks of a n round LBlock's n-fold structure. The C block for a n rounds LBlock's extended 2-stage is the concatenation of these A blocks.*

$$C = \left(A^{(1)} \dots A^{(n)}\right)$$

Which corresponds to the constraints given by the bit-oriented modeling for the XOR operations. The A blocks accordingly correspond to the convex hull constraints.



■ Figure 4.9. The constraint matrix of a differential cryptanalysis for 1 rounds LBlock using the exact transition modeling prior to being sorted



■ Figure 4.10. The constraint matrix of a differential cryptanalysis for 1 round's LBlock using the exact transition modeling. Additional borders have been added to signify the underlying two-stage structure.

▶ Definition 4.14 (A blocks). *Let $S_{n-fold} \in \mathbb{Z}^{8\times13}$ be the S block from the $B^{(bit-oriented)}$ part of a n round LBlock's n-fold structure's B block. Each $A^{(i)} \in \mathbb{Z}^{106\times200n+64}$ block for $i \in \{1,\dots,8n\}$ is defined as*

$$A^{(1)} = \cdots = A^{(8)} = \begin{pmatrix} \overbrace{}^{4+37\cdot(i-1)} & \begin{array}{c} S_{n-fold} \\ \hline A^{(modeling)} \end{array} & \overbrace{}^{20+102n+8\cdot(8-i)} \end{pmatrix}$$

$$A^{(9)} = \cdots = A^{(8n)} = \begin{pmatrix} \overbrace{}^{300+29\cdot(i-9)} & \begin{array}{c} S_{n-fold} \\ \hline A^{(modeling)} \end{array} & \overbrace{}^{12+29\cdot(8n-i)} \end{pmatrix}$$

$$A^{(modeling)} = \begin{pmatrix} -1 & -1 & -1 & -1 & \overbrace{}^{4} & 4 & -1 & -1 & -1 & -1 \\ & & & & & -1 & & & & \\ & & & & & 1 & & & & \\ & & & & \mathbf{0}^{6\times13} & & & & & \\ & S^{(1)} & & & & & & S^{(2)} & & \end{pmatrix}$$

with $S^{(1)}, S^{(2)} \in \{-1,1\}^{97\times4}$.

Given the sizes of the $S^{(1)}$ and $S^{(2)}$ blocks and that they would be different for each of the eight S-boxes used in LBlock, we do not offer an exact definition beyond that each entry is either 1 or −1. Lastly, the B blocks, the ones laying on the diagonal, are left over which include the $Q^{p}_{i,j}$ and $Q^{p}_{i,j,\ell}$ variables.

▶ Definition 4.15 (B Blocks). *Each B block in the two-stage MILP of an LBlock differential cryptanalysis using exact transition modeling is defined as*

$$B = \left(\begin{array}{cccccccccc} & & & & & \overbrace{}^{69} & & & \overbrace{}^{21} & \\ 1 & 1 & 1 & & & & & & & \\ -1 & -1 & -1 & & & & & & & \\ 1 & & & -1 & & & & & & \\ -1 & & & 1 & & & & & & \\ & 1 & & & -1 & \cdots & -1 & & & \\ & -1 & & & 1 & \cdots & 1 & & & \\ & & 1 & & & & & -1 & \cdots & -1 \\ & & -1 & & & & & 1 & \cdots & 1 \\ & & & 8 & & & & & & \\ & & & & 8 & & & & & \\ & & & & & & \ddots & & & \\ & & & & & & & & & 8 \end{array}\right)$$

These B blocks include the variables introduced by the exact transition modeling and their relation. The first rows of 1s and -1s ensure an equality relation between $Q_{i,j}$ and the $Q^{p}_{i,j}$ variables as well as between the $Q^{p}_{i,j}$ and their respective $Q^{p}_{i,j,\ell}$. In the following diagonal, each 8 corresponds to a line $A^{(modeling)}$ block which the respective $B^{(i)}$ block aligns with horizontally.

4.5 Comparison of Block-Structured Modelings

As stated in a prior section, our initial expectation does not hold. To be exact, each modeling that primarily tries to restrict the values of the S-boxes' input and output variables to those corresponding to feasible transitions end up leading to an n-fold MILP. And while the MILP resulting from the exact transition modeling includes an n-fold structure, its two-stage structure is more prevalent to the eye. As stated in Klein [18], the worst-case complexity for two-stage MILP is larger than that for n-fold. Given that Baksi [2] not only restricted the S-box transition but also finds the exact transition taken, and subsequently sets the objective function in relation to the exact diffusion or bias introduced by the transitions, it is expected to be more complex. While this work is not a comprehensive review of all possible modelings, the underlying structure in the sample we worked with has not changed from Sun et al. [32] in 2013 to Boura and Coggia [8] in 2020, while the number of constraints required to restrict the transition to only feasible ones has decreased.

4.6 Open Problems

A full analysis of linear cryptanalysis for LBLock as well as both differential and linear cryptanalysis for Gift64 and AES were out of the scope of this work. However given the inherent difference in structures between Feistel networks and SPNs, they might be of interest, especially given that one might be able to generalize from the findings of the two already implemented SPN to all of them. However, this also collides with a second problem which is the computational complexity of simply building the constraints for AES. Given weeks of time, our machines were unable to compute the convex hull for the AES S-box. Sadly, even if our machines had been able to compute all of the constraints, the numbers of constraints and variables were beyond what we could reasonably visualize.

As mentioned in Section 2.8, Boura and Coggia [8] offer some further approaches to modeling S-box transitions as well as alternative modeling for cipher's linear layers which were out of scope for this work but might provide new structures. Meanwhile, the works of Abdelkhalek, et al. [1] as well as Sasaki and Todo [27] appear to include more approaches to modeling S-boxes in MILPs, which we did not get into, but could offer some new structures. Lastly, Zhou et al. [36] provides some more constraints when performing a divide-and-conquer approach in a parallel setting which could overall change the structure as well. Given the modularity of the framework, including these approaches should be well within reach.

Finally, so far, we have not yet found anyone mentioning the usage of algorithms tailored to specific block structures. We hope this work's result of finding block structures in LBlock and providing a framework motivates more research both into finding and using MILPs' underlying structures in the calculation of cipher's security bounds against linear and differential cryptanalysis.

Bibliography

[1] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. "MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics". In: *IACR Transactions on Symmetric Cryptology* 2017.4 (Dec. 2017), pp. 99–129. DOI: 10.13154/tosc.v2017.i4.99-129. URL: https://tosc.iacr.org/index.php/ToSC/article/view/805.

[2] Anubhab Baksi. "New Insights On Differential And Linear Bounds Using Mixed Integer Linear Programming (Full Version)". In: *IACR Cryptol. ePrint Arch.* (2020), p. 1414. URL: https://eprint.iacr.org/2020/1414.

[3] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Siang Meng Sim, Yosuke Todo, and Yu Sasaki. "GIFT: A Small Present". In: *IACR Cryptol. ePrint Arch.* (2017), p. 622. URL: http://eprint.iacr.org/2017/622.

[4] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

[5] 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.

[6] Alex Biryukov and Christophe De Cannière. "Linear Cryptanalysis for Block Ciphers". In: *Encyclopedia of Cryptography and Security, 2nd Ed.* Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Springer, 2011, pp. 722–725. DOI: 10.1007/978-1-4419-5906-5_589. URL: https://doi.org/10.1007/978-1-4419-5906-5%5C_589.

[7] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Springer, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31. URL: https://doi.org/10.1007/978-3-540-74735-2%5C_31.

[8] Christina Boura and Daniel Coggia. "Efficient MILP Modelings for Sboxes and Linear Layers of SPN ciphers". In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 327–361. DOI: 10.13154/tosc.v2020.i3.327-361. URL: https://doi.org/10.13154/tosc.v2020.i3.327-361.

[9] Canonical Foundation. *Ubuntu Linux*. 2023. URL: https://ubuntu.com/.

[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] Joan Daemen and Vincent Rijmen. “The Block Cipher Rijndael”. In: *Smart Card Research and Applications, This International Conference, CARDIS ’98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*. Ed. by Jean-Jacques Quisquater and Bruce Schneier. Vol. 1820. Lecture Notes in Computer Science. Springer, 1998, pp. 277–284. DOI: 10.1007/10721064_26. URL: https://doi.org/10.1007/10721064%5C_26.

[12] Marius Hagemann. “Block-Structured Mixed Integer Linear Programs”. MA thesis. Goethe University Frankfurt, July 2023.

[13] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[14] 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.

[15] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[16] IBM. *IBM ILOG CPLEX Optimizer*. 2023. URL: http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

[17] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Third Edition*. Milton: CRC Press LLC, 2019. URL: https://www.routledge.com/Introduction-to-Modern-Cryptography/Katz-Lindell/p/book/9780815354369.

[18] Kim-Manuel Klein. “About the complexity of two-stage stochastic IPs”. In: *Math. Program.* 192.1 (2022), pp. 319–337. DOI: 10.1007/s10107-021-01698-z. URL: https://doi.org/10.1007/s10107-021-01698-z.

[19] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011. ISBN: 978-3-642-17341-7. DOI: 10.1007/978-3-642-17342-4. URL: https://doi.org/10.1007/978-3-642-17342-4.

[20] Sofia Lohr. *Analyzing MILPs for Linear and Differential Cryptanalysis*. Dec. 2022.

[21] 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.

[22] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[23] Nicky Mouha and Bart Preneel. *Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20*. Cryptology ePrint Archive, Paper 2013/328. https://eprint.iacr.org/2013/328. 2013. URL: https://eprint.iacr.org/2013/328.

[24] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. “Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming”. In: *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*. Ed. by Chuankun Wu, Moti Yung, and Dongdai Lin. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76. DOI: 10.1007/978-3-642-34704-7_5. URL: https://doi.org/10.1007/978-3-642-34704-7%5C_5.

[25] Kenji Ohkuma, Hideo Shimizu, Fumihiko Sano, and Shin-ichi Kawamura. “Security Assessment of Hierocrypt and Rijndael against the Differential and Linear Cryptanalysis (Extended Abstract)”. In: *IACR Cryptol. ePrint Arch.* (2001), p. 70. URL: http://eprint.iacr.org/2001/070.

[26] Python Software Foundation. *Python 3*. 2023. URL: https://www.python.org/.

[27] Yu Sasaki and Yosuke Todo. “New Algorithm for Modeling S-box in MILP Based Differential and Division Trail Search”. In: *Innovative Security Solutions for Information Technology and Communications - 10th International Conference, SecITC 2017, Bucharest, Romania, June 8-9, 2017, Revised Selected Papers*. Ed. by Pooya Farshim and Emil Simion. Vol. 10543. Lecture Notes in Computer Science. Springer, 2017, pp. 150–165. DOI: 10.1007/978-3-319-69284-5_11. URL: https://doi.org/10.1007/978-3-319-69284-5%5C_11.

[28] Georg Schnitger. *Skript zur Vorlesung „Komplexitätstheorie“*. July 2020.

[29] Douglas R. Stinson. *Cryptography - theory and practice*. Discrete mathematics and its applications series. CRC Press, 1995. ISBN: 978-0-8493-8521-6.

[30] Siwei Sun, Lei Hu, Ling Song, Yonghong Xie, and Peng Wang. “Automatic Security Evaluation of Block Ciphers with S-bP Structures against Related-key Differential Attacks”. In: *IACR Cryptol. ePrint Arch.* (2013), p. 547. URL: http://eprint.iacr.org/2013/547.

[31] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, and Ling Song. “Automatic Enumeration of (Related-key) Differential and Linear Characteristics with Predefined Properties and Its Applications”. In: *IACR Cryptol. ePrint Arch.* (2014), p. 747. URL: http://eprint.iacr.org/2014/747.

[32] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. “Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-oriented Block Ciphers”. In: *IACR Cryptol. ePrint Arch.* (2013), p. 676. URL: http://eprint.iacr.org/2013/676.

[33] The Sage Developers. *SageMath, the Sage Mathematics Software System*. Version 9.5. DOI 10.5281/zenodo.6259615. 2022. URL: https://www.sagemath.org.

[34] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[35] Wenling Wu and Lei Zhang. “LBlock: A Lightweight Block Cipher”. In: *IACR Cryptol. ePrint Arch.* (2011), p. 345. URL: http://eprint.iacr.org/2011/345.

[36] 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”. In: *IACR Trans. Symmetric Cryptol.* 2019.4 (2019), pp. 438–469. DOI: 10.13154/tosc.v2019.i4.438-469. URL: https://doi.org/10.13154/tosc.v2019.i4.438-469.

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

gemäß § 34, Abs. 16 der Ordnung für den Masterstudiengang Informatik vom 17. Juni 2019

Hiermit erkläre ich

____

(Nachname, Vorname)

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass die von mir eingereichten schriftlichen gebundenen Versionen meiner Masterarbeit mit der eingereichten elektronischen Version meiner Masterarbeit übereinstimmen.

Frankfurt am Main, den

____

Unterschrift der/des Studierenden