University Politehnica of Bucharest
Faculty of Automatic Control and Computer Science
Computer Science and Engineering Department

DIPLOMA PROJECT

A tool for automated discovery of differential characteristics with applications in real-world cryptographic algorithms

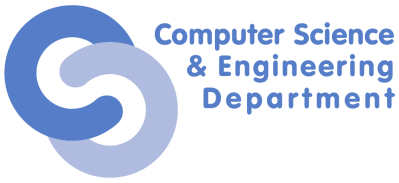
Theodor-Gabriel Tulbă-Lecu
gabitulbalecu@gmail.com

Thesis advisor:

Conf. Dr. Marios Omar Choudary

Bucharest
2023

Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare



PROIECT DE DIPLOMĂ

O unealtă pentru descoperirea automată a caracteristicilor diferențiale cu aplicații în algoritmi criptografici din lumea reală

Theodor-Gabriel Tulbă-Lecu
gabitulbalecu@gmail.com

Coordonator științific:

Conf. Dr. Marios Omar Choudary

București
2023

Contents

Abstract

Modern cryptography has evolved constantly as more and more private activities now happen online. That has led the research field to expand rapidly, especially cryptanalysis. As vast computational power has become cheaper and more accessible, the need for provably secure and fast algorithms has increased. Based on the previous results of Matsui [Mat95], we propose a generic and flexible tool for the automated finding of differential characteristics for a large class of block ciphers. We implemented the tool as an open-source library written in `C++`, accessible on GitHub: https://github.com/GabiTulba/GenericCryptanalyzer. Successful analysis of four block ciphers – a toy example, DES, PRESENT and Speck32 was achieved, and we obtained similar results to those in the literature.

Sinopsis

Criptografia modernă a evoluat constant pe măsură ce tot mai multe activități private s-au mutat in mediul online. Acest lucru a determinat expansiunea rapidă a domeniului de cercetare, în special criptanaliza. Pe măsură ce puterea computațională vastă a devenit tot mai ieftină și mai accesibilă, nevoia de algoritmi siguri și rapizi a crescut. Pe baza rezultatelor anterioare ale lui Matsui [Mat95], este propusă o unealtă generică și flexibilă pentru găsirea automată a caracteristicilor diferențiale pentru o clasă mare de cifruri bloc. Aceasta este implementată ca o bibliotecă open-source scrisă în `C++`, accesibilă pe GitHub: https://github.com/GabiTulba/GenericCryptanalyzer. S-au analizat cu succes patru cifruri bloc – un exemplu simplu, DES, PRESENT și Speck32, iar rezultatele obținute au fost similare cu cele din literatura de specialitate.

Acknowledgments

I want to thank my parents for supporting me all these years. Without their support, nothing I ever achieved would have been possible.

I also want to thank everyone from my cybersecurity team HTsP, with whom, during the last six years, I have developed extensive knowledge of cybersecurity and especially cryptography.

Finally, I want to thank my thesis advisor, Conf. Dr. Marios Omar Choudary, for his support, recommendations, and trust in me.

1 Introduction

1.1 Context

Cryptanalysis has been my primary interest ever since I started studying cryptography in 2018. That is when I started playing in Capture The Flag (CTF) competitions. As I got more experienced in this field, I started learning about more advanced techniques, such as differential cryptanalysis. With time, I grew curious about how the differential characteristics presented in scientific articles are found and thought of a way to automate this search process. After some time, I discovered multiple ways to do this, but I focused on the origins of automation. In 1995, Matsui proposed an algorithm to find linear and differential best characteristics for DES [Mat95] in less than 2 hours. After reading that paper, I decided to improve this algorithm by implementing a generic library to solve this problem for a large class of block ciphers.

1.2 Motivation

As the internet has evolved, more and more of our private lives have moved to the online medium, and the need for secure communication has increased dramatically. This phenomenon led to an increase in interest in cryptography to find better and safer ways to encrypt and protect our data *over the wire*.

Due to the massive amount of data we need to encrypt constantly, this process has to be fast. That is where symmetric cryptography comes into play, a way to encrypt information based on mixing the input with a secret key to form the encrypted data that is indistinguishable from random text rather than mathematically complex problems as in public-key cryptography. Because of limited time, the thesis only focuses on one half of symmetric-key cryptography – block ciphers, but the techniques can also be extended to stream ciphers.

Differential and linear cryptanalysis are the most well-known techniques to attack block ciphers that have stood the test of time. The former was introduced in [BS91], and the latter was first introduced in [MY93]. Over the years, these techniques have proven to be one of the most efficient and widely used in searching for vulnerabilities in block ciphers.

As computational power and interest in safe cryptographic encryption schemes have increased, automated search for differential and linear characteristics has become necessary to develop new, more secure ciphers and find new attacks against ones previously considered secure. This thesis presents an implementation of a tool that aims to help in the automated discovery of such vulnerabilities to speed up research times.

1.3 Problem

Finding an automated way to find differential characteristics of block ciphers in a reasonable time window and with little to no effort and to generalize known algorithms to support multiple classes of block ciphers while still maintaining a relatively simple API.

1.4 Objectives

This thesis aims to present a generic approach to differential cryptanalysis and implement an open-source `C++` library that can analyze different ciphers with little to no modifications between them and test the library against well-known modern block ciphers. We also aim to obtain similar results and performance with other tools.

1.5 Solution

The proposed solution is to implement and add optimizations to the algorithm presented in [Mat95] using an Object-Oriented approach that will allow a large class of ciphers to be analyzed. We add support for S-box based ciphers using Difference Distribution Tables and for modular addition using partial Difference Distribution Tables [BV13].

1.6 Results

The successful cryptanalysis of a simple Substitution-Permutation Network (SPN) cipher presented in [Hey01], DES [Sta77], PRESENT [BKL$^+$07] and Speck32 [BSS$^+$13] block ciphers was achieved. For DES, we found the same results as Matsui found using his algorithm in [Mat95]. For PRESENT, we got the same results as those found in [Wan07]. Finally, for Speck32, we found different but similar performance results to those in [BV13] and [DM18].

1.7 Outline of the Thesis

The remaining of this document is structured in the following way. In chapter 2 the needed background regarding cryptography and cryptanalysis is given, and we discuss previous and related work in this field. In chapter 3, we describe the project, the problem that needs to be solved, and a general algorithm to solve it. We discuss implementation and optimizations in chapter 4. In chapter 5, we apply the algorithm to well-known ciphers and discuss the results. Finally, in chapter 6, we give some final remarks and discuss further improvements.

2 Background

In this section, a summary of symmetric cryptography, block ciphers, and differential and linear cryptanalysis will be presented, as well as previous and related work that has been done on the subject of automated discovery of differential/linear characteristics.

2.1 Cryptograpy

"If I take a letter, lock it in a safe, hide the safe somewhere in New York, then tell you to read the letter, that is not security. That is obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world's best safecrackers can study the locking mechanism—and you still cannot open the safe and read the letter—that is security." – Bruce Schneier, Applied Cryptography [Sch96]

Cryptography can be defined as the study of secure communication between two or more parties. That is, an outside party should not be able to gain any information shared between two communicating parties without their knowledge, even when the protocol used by the parties is known.

This secure communication is usually achieved through a shared secret, called a *"key"*. Based on how the key is generated and used, cryptography splits into two subdomains – symmetric-key cryptography and public-key cryptography. For this thesis, we will only discuss the former.

The process of transforming plain text data into a code, usually called *"ciphertext"*, is called encryption. Similarly, the process of recovering the original data from ciphertext is called *"decryption"*.

2.1.1 Symmetric-Key Cryptography

Symmetric-key cryptography represents a class of cryptographic algorithms that: use the same key for both encryption and decryption; the two keys can be converted from one to another through a simple process or are derived from a shared secret. The task of securely generating a shared secret is achieved through public-key cryptography protocols such as the Diffie-Hellman key exchange.

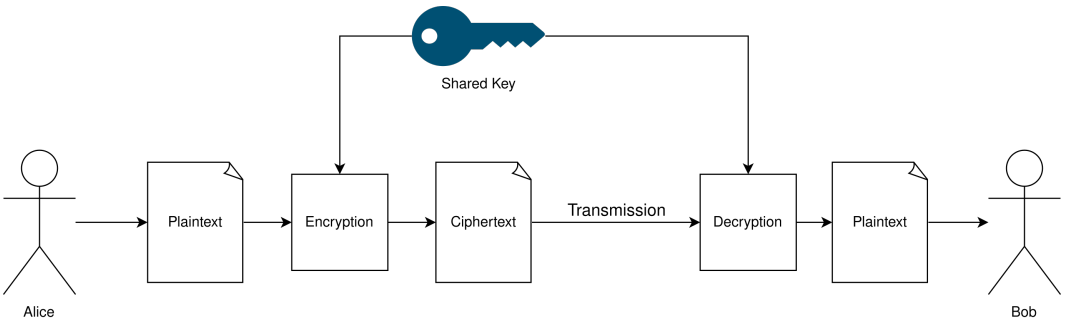


Figure 1: A typical communication process between two parties, Alice and Bob, using symmetric-key cryptography. Both Alice and Bob know the secret key that they previously computed securely.

The main reason symmetric-key cryptography exists is that it fits better the need for encrypting data fast, as the algorithms are less complex and provide better performance when compared to public-key cryptography, which due to the inherent complex mathematical operations used, is computationally expensive.

Symmetric-key cryptography splits into two classes of algorithms, stream ciphers, which encrypt and decrypt data one bit/byte at a time, and block ciphers which encrypt and decrypt data in blocks of various sizes defined by the protocol in use. Even though both their cryptanalysis share common properties, this thesis will only focus on the latter.

2.1.2 Block Ciphers

Block ciphers are a class of symmetric-key cryptographic algorithms that, as the name suggests, operate on blocks – groups of bits of fixed length. The main difference from their counterpart – stream ciphers, is that block ciphers take some plaintext and a key and mangle them into an unreadable ciphertext the same size as the plaintext. In other words, they mimic a random permutation. A more formal mathematical representation of a block cipher is the following:

$$E(P, K) : \{0,1\}^m \times \{0,1\}^k \to \{0,1\}^m, \tag{1}$$

$$D(P, K) : \{0,1\}^m \times \{0,1\}^k \to \{0,1\}^m \tag{2}$$

where m is the bit size of the block, and k is the secret key size. The two functions E (Encryption) and D (Decryption) must respect the following properties:

$$D(E(P, K), K) = P, \text{ and } E(D(C, K), K) = C \tag{3}$$

On the other hand, stream ciphers work by generating an arbitrarily long stream of bits generated from the secret key that is XOR-ed (bitwise eXclusive OR) with the plaintext.

Then one might say: *"Well then, how do you use block ciphers to encrypt a message of arbitrary length like stream ciphers"*. Here is where block cipher modes of operation come into play. A long message can be padded to an integer multiple of the block cipher's bit size and then split into blocks. Each block can then be encrypted with the block cipher individually with the same key. This process is called the Electronic Code Book mode of operation. EBC mode is insecure because two plaintexts will always map to the same

ciphertext. Because of this, some information about the original message is revealed to an external attacker/eavesdropper. Many modes of operation exist, but since they are not of major interest to this thesis, we will only enumerate some of the more popular modes of operation that only offer confidentiality and their mathematical formulae.[1]

Mode of operation	Encryption	Decryption
Electronic Code Book (ECB)	$C_i = E(P_i, K)$	$P_i = D(C_i, K)$
Cipher Block Chaining (CBC)	$C_i = E(P_i \oplus C_{i-1}, K)$, $C_0 = IV$	$P_i = D(C_i, K) \oplus C_{i-1}$, $C_0 = IV$
Cipher FeedBack (CFB)	$C_i = E(C_{i-1}, K) \oplus P_i$, $C_0 = IV$	$P_i = E(C_{i-1}, K) \oplus C_i$, $C_0 = IV$
Output FeedBack (OFB)	$C_i = P_i \oplus O_i$, $O_i = E(O_{i-1}, K)$, $O_0 = IV$	$P_i = C_i \oplus O_i$, $O_i = E(O_{i-1}, K)$, $O_0 = IV$
Counter (CTR)	$C_i = P_i \oplus E(CTR_i, K)$, $CTR_i = Nonce\|\|i$	$P_i = C_i \oplus E(CTR_i, K)$, $CTR_i = Nonce\|\|i$

Table 1: A list of common block cipher modes of operation.

2.2 Cryptanalysis

Cryptanalysis is the study of finding vulnerabilities in cryptographic protocols to extract as much information as possible from encrypted communication between two or more parties without prior knowledge of the secret key. It is usually assumed that the attacker/cryptanalyst has complete knowledge of the encryption procedure. That is, the entire security of the cipher stands in the secret key.

In [Sch96], Schneier splits cryptanalysis into the following six categories of attacks:

1. **Ciphertext-only attack** – The cryptanalyst has the ciphertext of several messages, all encrypted using the same encryption algorithm. The goal is to recover the plaintext of as many messages as possible, or better yet, to deduce the key used to encrypt the messages in order to decrypt other messages encrypted with the same key.

2. **Known-plaintext attack** – The cryptanalyst has access, in addition to the ciphertext of several messages, to the plaintext of those messages. The goal is to deduce the key used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key.

3. **Chosen-plaintext attack** – The cryptanalyst not only has access to the ciphertext and associated plaintext for several messages, but they also choose the plaintext that gets encrypted. This attack is more powerful than a known-plaintext attack because the cryptanalyst can choose specific plaintext blocks to encrypt, which might yield more information about the key. The goal is to deduce the key used to

[1] IV stands for initialization vector, C_i represents the i^{th} block of the ciphertext, P_i the i^{th} block of the plaintext and $\oplus$ represents the bitwise exclusive or operation, *Nonce* stands for *number used once* and $||$ is the binary concatenation operator.

encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key.

4. **Adaptive-chosen-plaintext attack** – This is a particular case of a chosen plaintext attack. Not only can the cryptanalyst choose the plaintext that is encrypted, but they can also modify their choice based on the results of previous encryption. In a chosen-plaintext attack, a cryptanalyst might be able to choose one large block of plaintext to be encrypted; in an adaptive-chosen-plaintext attack, they can choose a smaller block of plaintext and then another based on the results of the first, and so forth.

5. **Chosen-ciphertext attack** – The cryptanalyst can choose different ciphertexts to be decrypted and have access to the decrypted plaintext. The goal is to deduce the key.

6. **Chosen-key attack** – This attack does not mean that the cryptanalyst can choose the key; it means that they have some knowledge about the relationship between different keys.

2.3 Cryptanalysis of Block Ciphers

As presented in [SPQ03], the cryptanalysis of block ciphers began in 1981 with the first observations of some undesirable properties of the Data Encryption Standard (DES) [Sta77]. Most cryptanalytic techniques were developed in the 1990s, the most notable two advancements being differential and linear cryptanalysis. Differential cryptanalysis was first presented at CRYPTO 90 by Biham and Shamir. Linear cryptanalysis was developed by Matsui and presented at EUROCRYPT 93. These two techniques also led to the development of criteria for the security evaluation of block ciphers. Recently designed block ciphers like the Advanced Encryption Standard Rijndael (AES) [DR00] have been based on provable security against these two attacks and their improvements.

Due to their heavily hardware-oriented nature, block cipher designs most often resemble a circuit diagram, where bits flow from one component/function of the cipher to another. Both linear and differential cryptanalysis focus on inspecting how the plaintext bits flow through those circuits and how changing a small input changes the outcome of the encryption in the case of differential cryptanalysis or how plaintext and ciphertext bits are related in the case of linear cryptanalysis.

Next, we will briefly discuss each of those techniques and their relationship. A comprehensive tutorial on both can be found in [Hey01].

2.3.1 Differential Cryptanalysis

Differential cryptanalysis is a chosen-plaintext attack that, as mentioned in section 2.2, allows the attacker/cryptanalyst to choose any number of plaintexts to encrypt.

As the name suggests, differential cryptanalysis takes advantage of how differences between two input plaintexts propagate through the cipher, resulting in two different outputs[2] with presumably high probability.

[2] A distinguisher can be built using the ciphertext outputs. Although, that does not necessarily mean only the output ciphertexts are used in differential cryptanalysis. In order to mount a key-recovery

Such a pair of input difference ΔX and output differences ΔY together with the probability p is called a differential characteristic and is often noted as:

$$\Delta X \xrightarrow{\mathrm{p}} \Delta Y := P(Y_1 \oplus Y_2 = \Delta Y | X_1 \oplus X_2 = \Delta X) = p \tag{4}$$

where $Y_1 = E(X_1, K), Y_2 = E(X_2, K)$.

The main advantage of differential cryptanalysis is that, in each round, the key addition process can be eliminated from the cipher since the key will be XOR-ed with both parts of the characteristic and, therefore, they cancel out:

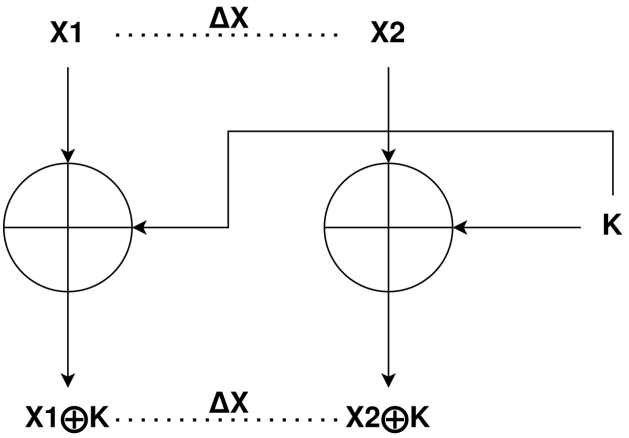


Figure 2: Illustration of key independence of differential characteristic propagation.

Similarly, it can be shown that multiple other components of block ciphers are linear in rapport with the XOR operator and thus do not directly affect the probability of the characteristic, such as expansion boxes and permutation boxes.

The only non-deterministic components in differential characteristics are non-linear equations in rapport with the bitwise XOR operator, such as S-boxes. For such components, we construct a Difference Distribution Table (DDT) – a look-up table that encapsulates all the probabilities of differential characteristics for that component. For example, consider the following S-box:

i	0	1	2	3	4	5	6	7
S(i)	1	2	0	0	3	1	3	2

Table 2: An example of a 3-bit input, 2-bit output S-box.

The corresponding DDT for it is the following[3]:

attack, we are interested in the inputs to the last round of the cipher. The same turns out to be the case with linear cryptanalysis.

[3]In the DDT, the row represents the input difference, and the column represents the output difference. The value of a cell (i, j) represents the number of equations that respect the equation $\Delta Y = S(X_1) \oplus S(X_2) = j$, given that $\Delta X = X_1 \oplus X_2 = i$. The probability of the characteristic can be obtained by dividing the cell's value by the input size of the component.

	$\Delta Y = 0$	$\Delta Y = 1$	$\Delta Y = 2$	$\Delta Y = 3$
$\Delta X = 0$	8	0	0	0
$\Delta X = 1$	2	2	2	2
$\Delta X = 2$	2	2	2	2
$\Delta X = 3$	0	4	4	0
$\Delta X = 4$	0	0	4	4
$\Delta X = 5$	2	2	2	2
$\Delta X = 6$	2	2	2	2
$\Delta X = 7$	0	4	0	4

Table 3: The corresponding DDT for the S-box described in Table 2.

The whole cipher differential characteristics can be constructed from simple components and tracing how the bits flow through the cipher. The final probability of the entire differential can be lower-bounded by the product of the probabilities of all the non-linear components. In more detail, if we chain two differential characteristics $\Delta X \xrightarrow{\text{p}} \Delta Y$ and $\Delta Y \xrightarrow{\text{q}} \Delta Z$, we know that the final differential characteristic $\Delta X \xrightarrow{\text{r}} \Delta Z$ has $r \geq pq$[4].

It is generally difficult to estimate the number of plaintext pairs required to mount a key-recovery attack using a differential characteristic. However, a good approximation is the following formula:

$$N_D \approx \frac{c}{p_D} \tag{5}$$

where p_D is the probability of the differential and c is a small constant (typically between 2 and 4). A more in-depth analysis of the probability of success of differential cryptanalysis can be found in Section 3 of [Sel08].

2.3.2 Linear Cryptanalysis

Linear cryptanalysis is a known-plaintext attack in which, as explained in section 2.2, the attacker/cryptanalyst is presented with a set of plaintext-ciphertext pairs. That means that compared to differential cryptanalysis, linear cryptanalysis works in tighter attack scenarios.

As was the case with differential cryptanalysis, the name is self-explanatory. Linear cryptanalysis approximates portions of a block cipher with linear equations (again, in rapport with the bitwise XOR operation), with the remark that those equations hold only with a certain probability. In general, such an equation is written as:

$$X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_k} \oplus Y_{j_1} \oplus Y_{j_2} \oplus \cdots \oplus Y_{j_l} = 0 \tag{6}$$

a shorthand that is often used in literature is:

$$\Gamma X := X_{i_1} \oplus X_{i_2} \oplus \cdots \oplus X_{i_k}, \Gamma Y := Y_{j_1} \oplus Y_{j_2} \oplus \cdots \oplus Y_{j_l} \tag{7}$$

where ΓX are the input bits and ΓY are the out bits that form the linear equation.

[4]That comes from the fact that there are many more differential characteristic pairs $\Delta X \xrightarrow{\text{p'}} \Delta Y'$ and $\Delta Y' \xrightarrow{\text{q'}} \Delta Z$, but for those, the product $p'q'$ is probably small enough to be ignored. The subfield of differential cryptanalysis that considers multiple differential characteristics is called, unsurprisingly, multiple differential cryptanalysis. More details can be found in [BG11].

A linear approximation/characteristic can be defined as follows:

$$(\Gamma X, \Gamma Y) = p : P(\Gamma X \oplus \Gamma Y = 0) = p \tag{8}$$

or from the perspective of probability biases:

$$(\Gamma X, \Gamma Y) = \frac{1}{2} + \epsilon \tag{9}$$

where $-\frac{1}{2} \leq \epsilon \leq \frac{1}{2}$ is called the probability bias[5].

The main property of linear cryptanalysis is how linear characteristics interact with key bits. Suppose we have a linear equation of the form:

$$\Gamma X \oplus \Gamma Y \oplus \bigoplus_{i=1}^{l} K_i = b \tag{10}$$

that holds with probability $p = \frac{1}{2} + \epsilon$, where K_i represents the i^{th} bit of the expanded key and b is a constant.

Since $\bigoplus_{i=1}^{l} K_i \oplus b \in \{0, 1\}$, the linear equation can be transformed into a linear characteristic with the same probability bias magnitude of $|\epsilon|$ by discarding this term. By doing so, either the probability remains the same or flips, but the magnitude of the bias remains unchanged. Therefore, the value of the key bits is irrelevant, as was the case with differential cryptanalysis.

The only non-deterministic components in linear cryptanalysis are non-linear equations in rapport with the bitwise XOR operator. Such an example are S-boxes. For those components, we build a Linear Approximation Table (LAT) that, similar to a DDT, encapsulates all biases of linear characteristics for that component. For example, we will build the LAT[6] for the same S-box we used to construct the DDT:

	$\Gamma Y = 0$	$\Gamma Y = 1$	$\Gamma Y = 2$	$\Gamma Y = 3$
$\Gamma X = 0$	$\frac{1}{2}$	0	0	0
$\Gamma X = 1$	0	$-\frac{1}{4}$	0	$\frac{1}{4}$
$\Gamma X = 2$	0	$-\frac{1}{4}$	0	$-\frac{1}{4}$
$\Gamma X = 3$	0	0	0	0
$\Gamma X = 4$	0	$\frac{1}{4}$	$\frac{1}{4}$	0
$\Gamma X = 5$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$
$\Gamma X = 6$	0	0	$-\frac{1}{4}$	$-\frac{1}{4}$
$\Gamma X = 7$	0	$-\frac{1}{4}$	$\frac{1}{4}$	0

Table 4: The corresponding LAT for the S-box described in Table 2.

[5]We are only interested in the magnitude of the biases, since as $|\epsilon| \to \frac{1}{2}$, the equation tends to be linear, and as $|\epsilon| \to 0$, the equation tends to have a uniform distribution.

[6]In the LAT, the row and column indices represent the binary representation of the coefficients of the linear equation for ΓX and ΓY, respectively. For example, in the sixth row $5 = 101_{(2)}$ corresponds to $\Gamma X = X_2 \oplus X_0$. The value in the cell (i, j) represents the probability bias for that characteristic.

Since we are working with linear equations, we can linearly combine them. For example, suppose we have two linear characteristics $(\Gamma X_1, \Gamma Y_1, p)$ and $(\Gamma X_2, \Gamma Y_2, q)$, we can combine them into a third characteristic like so: $(\Gamma X_{1,2} := \Gamma X_1 \oplus \Gamma X_2, \Gamma Y_{1,2} := \Gamma Y_1 \oplus \Gamma Y_2, r)$. To calculate the probability r of the new linear characteristic, we can use Matsui's piling-up lemma[Mat94]:

$$P(X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0) = \frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} \epsilon_i \tag{11}$$

where

$$P(X_i = 0) = \frac{1}{2} + \epsilon_i \tag{12}$$

Now let us go back to our simpler case. Let $p = \frac{1}{2} + \epsilon_1, q = \frac{1}{2} + \epsilon_2, r = \frac{1}{2} + \epsilon_{1,2}$, then $\epsilon_{1,2} = 2\epsilon_1\epsilon_2$.

Using this property of linear characteristics, we can build complete cipher characteristics from LATs and trace how the bits flow through the cipher's components similarly to how differential characteristics are built.

It is generally difficult to estimate the number of plaintexts required to mount a key-recovery attack using linear cryptanalysis. However, a good approximation is the following formula:

$$N_L \approx \frac{1}{\epsilon^2} \tag{13}$$

where ϵ is the probability bias of the linear approximation. A more in-depth analysis of the probability of success of differential cryptanalysis can be found in Section 2 of [Sel08].

2.3.3 Duality Between Differential and Linear Cryptanalysis

In both [Mat95] and [Bih95], we can find discussions on the duality between differential and linear cryptanalysis, Biham goes as far as using the same notation as used in [BS91]. Indeed, as illustrated in the previous two sections, both techniques are very similar. That allowed Matsui in [Mat95] to almost effortlessly change his algorithm from an automated search of differential characteristics to linear characteristics. The same should also be valid for the algorithm presented in this project.

2.4 Previous Work

2.4.1 Matsui's Search Algorithm For The Best Differential and Linear Characteristics

This thesis was heavily inspired by the algorithm presented in [Mat95], which was used to find the best differential characteristics of DES. The final algorithm presented here started from that exact algorithm introduced in the article mentioned above and then slowly evolved into an optimized and generalized form of it. That single article has set the foundation of the entire subdomain of automated cryptanalysis.

2.5 Related Work

In this section, we briefly mention other works from the literature that achieve the common goal of automated cryptanalysis and which helped with the development or as a source of inspiration for this work.

2.5.1 Branch-and-Bound Search Algorithms

This thesis falls into this category. Branch-and-bound search algorithms represent a class of algorithms that exhaustively search all possible characteristics while simultaneously trying to minimize the number of visited states. Most of the literature from this category is based on Matsui's original algorithm and improves on it by adding various optimizations, either general or that work on specific cipher structures. Some initial improvements have been made soon after the original algorithm. In [AKM97] and [OMA95], Aoki and Moriai brought two significant contributions that sped up the search and allowed them to successfully apply automated differential characteristic discovery to FEAL [SM87].

More recently, in [BZL14], Bao et al. further improved this technique by carefully rearranging the order in which the search tree of the branch-and-bound algorithm processes information.

In 2013 Biryukov et al. used the results from [LM01] to modify Matsui's algorithm to work for ARX block ciphers[BV13]. They were the first to find and publish results on the security of the Speck cipher family. We also use Biryukov's algorithm for computing pDDTs in our algorithm.

2.5.2 Satisfability Modulo Theorem Solvers

The SMT solver-based automated cryptanalysis class has been at the center of this research field for the last decade and is currently considered state-of-the-art. One of the first results obtained using this technique was presented in [MP13], in which the authors successfully built a tool to search for optimal differential characteristics for ARX ciphers and applied it to the stream cipher family Salsa20 [Ber08]. A more recent and powerful open-source tool, written in Python, from this category, is CASCADA [RR22].

2.5.3 Machine Learning-Based Distinguishers

Machine learning-based distinguishers are a very young branch of automated cryptanalysis. Ghor introduced the technique at CRYPTO 2019 in [Goh19]. In that article, the author successfully built a deep learning neural network distinguisher, called a *"neural distinguisher"* for round-reduced Speck32/64. Even though the results beat the state-of-the-art and opened up new opportunities for machine learning-aided cryptanalysis, they still need to offer new insights into why machine learning is a practical tool in cryptanalysis. A thorough analysis and detailed explanation of Ghor's results can be found in [BGPT21].

3 Project Description

In this chapter, a detailed description of the tool and the ideas behind it are provided. We will first describe the search algorithm that Matsui used. Then, we will talk about how we can model the internal structure of a block cipher. Finally, we will present our algorithm and compare it to other existing tools.

3.1 Description of the Search Algorithm

3.1.1 Matsui's original algorithm

The algorithm presented in [Mat95] is based on mathematical induction on the number of rounds of the cipher. It finds the best differential characteristic probability of a n-round cipher, B_n, based on the previously computed differential characteristic probabilities B_i, for $1 \leq i < n$. The algorithm also takes as input an initial value for B_n, noted as $\overline{B_n}$. The algorithm will find a correct answer as long as $\overline{B_n} \leq B_n$ and the search time decreases as $\overline{B_n} \to B_n$.

Algorithm 1: Matsui's algorithm for finding the best differential characteristic probability for a n-round DES

Input : $\overline{B_n}$ = initial value for the best probability for the n^{th} round;
$B_i,\ 1 \leq i < n$ = the best probabilities for all previous rounds.
Output: B_n = the best probability for the n^{th} round.
function *Search Procedure Round(1)*:
    foreach *candidate for* ΔX_1 **do**
        Let $\Delta X_1 \xrightarrow{p_1} \Delta Y$ be the best differential characteristic for input ΔX_1 for the first round;
        if $p_1 B_{n-1} \geq \overline{B_n}$ **then**
            call *Search Procedure Round(2)*;
    return $\overline{B_n}$;
function *Search Procedure Round(2)*:
    foreach *candidate for* ΔX_2 *and* ΔY_2 **do**
        Let $\Delta X_2 \xrightarrow{p_2} \Delta Y_2$ be the differential characteristic for input ΔX_2 and output ΔY_2 for the second round;
        if $p_1 p_2 B_{n-2} \geq \overline{B_n}$ **then**
            call *Search Procedure Round(3)*;

function *Search Procedure Round(i)*:
    Let $\Delta X_i \longleftarrow \Delta X_{i-2} \oplus \Delta Y_{i-1}$;
    foreach *candidate for* ΔY_i **do**
        Let $\Delta X_i \xrightarrow{p_i} \Delta Y_i$ be the differential characteristic for input ΔX_i and output ΔY_i for the i^{th} round;
        if $\prod_{j=1}^{i} p_j B_{n-i} \geq \overline{B_n}$ **then**
            call *Search Procedure Round(i + 1)*

function *Search Procedure Round(n)*:
    Let $\Delta X_n \longleftarrow \Delta X_{n-2} \oplus \Delta Y_{n-1}$;
    Let $\Delta X_n \xrightarrow{p_n} \Delta Y$ be the best differential characteristic for input ΔX_n for the last round;
    if $\prod_{i=1}^{n} p_j > \overline{B_n}$ **then**
        $\overline{B_n} \longleftarrow \prod_{i=1}^{n} p_j$;

If we apply this exact algorithm, the loops in the first two rounds would be computationally impractical since the loop size is about 2^{32} and 2^{64}, respectively. Matsui provides an optimization in his paper that allows for a more efficient approach. We used a similar approach to him, splitting the cipher into rounds and the rounds into simple components called boxes, which will be described in the next section.

3.2 The Structure of a Block Cipher

In this section we will discuss our representation of a block cipher in order to prepare for presenting the final algorithm.

A block cipher comprises multiple chained rounds, where the i^{th} round's output is the $i + 1^{\text{th}}$ round's input. Each round comprises multiple components that we will call from now on *"boxes"*. The structure of a round is that of a directed acyclic graph, where the graph's source is the round's input, and the sink is the output.

3.2.1 What is a Box?

We will represent a box as a mathematical black-box function that takes some input, applies some transformations, and outputs the result alongside the probability of that output.

Formally a box B that takes as input inp_{sz} bits and outputs out_{sz} bits is defined as:

$$B(input) = (output, prob) : \{0,1\}^{inp_{sz}} \rightarrow \{0,1\}^{out_{sz}} \times [0,1] \tag{14}$$

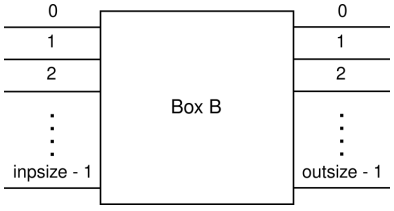


Figure 3: Black-box representation of a box

Once a box's input is set, it will produce all the possible outputs in decreasing order of the magnitude of the probability.

3.2.2 Linear Boxes

As the name suggests, a linear box linearly transforms the input bits in rapport with the bitwise XOR operation. Such boxes are[1]:

1. **Expansion Box** – a box where the input bits are copied, possibly duplicated, and shifted around;
2. **Permutation Box** – a box where the input bits are permuted;
3. **Identity Box** – a box in which the output equals the output. This box is not an essential component but is often helpful in constructing ciphers.
4. **Xor Box** – a box that performs bitwise XOR on the first and second half of the input bits.

All linear boxes output only one output with probability 1. That is, they are deterministic.

3.2.3 Non-Linear Boxes

A *non-linear box* is a box that represents a non-linear transformation in rapport with the bitwise XOR operation. Such boxes are:

1. **Substitution Box** – A box in which a look-up input table defines the output.
2. **Modular Addition** – A box that computes the output as the modular addition between the two halves of the input.

A non-linear box is non-deterministic and, therefore, outputs any number of outputs (possibly 0 in the case of the modular addition) [LM01], and the output is determined using a DDT.

3.2.4 Computing DDTs for S-Boxes

The DDT of an S-box can be computed as follows:

[1] this list is not exhaustive, the enumerated boxes are some of the more common components of block ciphers

Algorithm 2: Algorithm to compute Difference Distribution Table for an S-box

Input : $sbox$ = the look-up table; $input_{sz}$ = the bit size of the S-box's input; $output_{sz}$ = the bit size of the S-box's output

Output: DDT = the Difference Distribution Table – $DDT(X, Y) = p$, where $\Delta X \xrightarrow{p} \Delta Y$

function *Compute DDT(sbox)*:

- **foreach** $X \in \{0,1\}^{input_{sz}}$ **do**
  - **foreach** $Y \in \{0,1\}^{output_{sz}}$ **do**
    - $DDT(X, Y) \longleftarrow 0$;
- **foreach** $X_1 \in \{0,1\}^{input_{sz}}$ **do**
  - **foreach** $X_2 \in \{0,1\}^{input_{sz}}$ **do**
    - Let $\Delta X \longleftarrow X_1 \oplus X_2$;
    - Let $\Delta Y \longleftarrow sbox(X_1) \oplus sbox(X_2)$;
    - $DDT(\Delta X, \Delta Y) \longleftarrow DDT(\Delta X, \Delta Y) + \frac{1}{2^{output_{sz}}}$;
- **return** DDT;

The time complexity of this algorithm is $\mathcal{O}(2^{2input_{sz}} + 2^{input_{sz}+output_{sz}})$ and the spatial complexity is $\mathcal{O}(2^{input_{sz}+output_{sz})})$.

3.2.5 Computing partial DDTs for Modular Addition

Modular addition can be treated as an S-box where the input is the two terms of the addition, and the output is their sum. Because of this, it is impractical to compute the entire DDT for a big $word_{sz}$ since, as mentioned before, the time and spatial complexities would be (taking into consideration the fact that $input_{sz} = 2output_{sz} = 2word_{sz}$) $\mathcal{O}(2^{4word_{sz}})$ and $\mathcal{O}(2^{3word_{sz}})$, respectively.

In [BV13], Biryukov et al., based on previous analysis done in [LM01], presented an algorithm that could compute the partial DDT that only keeps track of the differential characteristics with a probability greater than a certain threshold. The procedure works as follows:

Algorithm 3: Biryukov's algorithm to compute partial Difference Distribution Table for modular addition

Input : $word_{sz}$ = the word size of the modular addition operation; p_{thresh} = the probability threshold of a differential characteristic

Output: $pDDT$ = the partial DDT – $pDDT(a, b, c) = p \geq p_{thresh}$

function *eqal(a, b, c)*:
  return $(\neg a \oplus b) \wedge (\neg a \oplus c)$;

function *compute probability(a, b, c)*:
  if $eqal(a \ll 1, b \ll 1, c \ll 1) \wedge (a \oplus b \oplus c \oplus (b \ll 1)) \neq 0$ **then**
    return 0;
  return $2^{-hw(equal(a,b,c))}$;

function *Search pDDT*$(word_{sz}, p_{thresh}, pDDT, k, p, a, b, c)$:
  if $k = word_{sz}$ **then**
    $pDDT(a, b, c) \longleftarrow p$;
    return;
  foreach $x, y, z \in \{0, 1\}$ **do**
    Let $a' \longleftarrow x|a,\ b' \longleftarrow y|b,\ c' \longleftarrow z|c$;
    Let $p' \longleftarrow$ *compute probability*(a', b', c');
    if $p' \geq p_{thresh}$ **then**
      call *Search pDDT*$(word_{sz}, p_{thresh}, pDDT, k + 1, p', a', b', c')$;

function *Compute pDDT*$(word_{sz}, p_{thresh})$:
  Let $pDDT \longleftarrow \emptyset$;
  call *Search pDDT*$(word_{sz}, p_{thresh}, pDDt, 0, 1, \emptyset, \emptyset, \emptyset)$;
  return pDDT;

In the algorithm above, $hw(x)$ is the Hamming weight of the binary representation of x, $\ll$ represents the bitwise left shift operation, $|$ the concatenation of two binary numbers, $\neg$ bitwise negation, and $\wedge$ bitwise AND.

3.2.6 Inter-Box interaction

As shown above, boxes have multiple inputs and outputs of a single bit. Therefore, the output of a single box can be the input or the partial input of multiple boxes, and the same bits from the output can be the input of multiple boxes. We define a box's interaction with other boxes as a list of edges in the graph of boxes of the following form: the destination box, its input bits range, and the output bits range of the current box. When the output of a box is determined, it will automatically notify all the boxes in its output list with the output bits. That allows for high flexibility in implementing block ciphers within our algorithm.

Algorithm 4: Algorithm to notify all boxes in the output list of a box

function *set input(box, range, bits)*:
    box.input[range.start, ..., range.end] = bits;

function *notify all(src)*:
    foreach $(dst, range_{inp}, range_{out}) \in src.dests$ **do**
        Let $bits \longleftarrow src.output[range_{out}.start, \ldots, range_{out}.end]$;
        call *set input(dst, $range_{inp}$, bits)*;

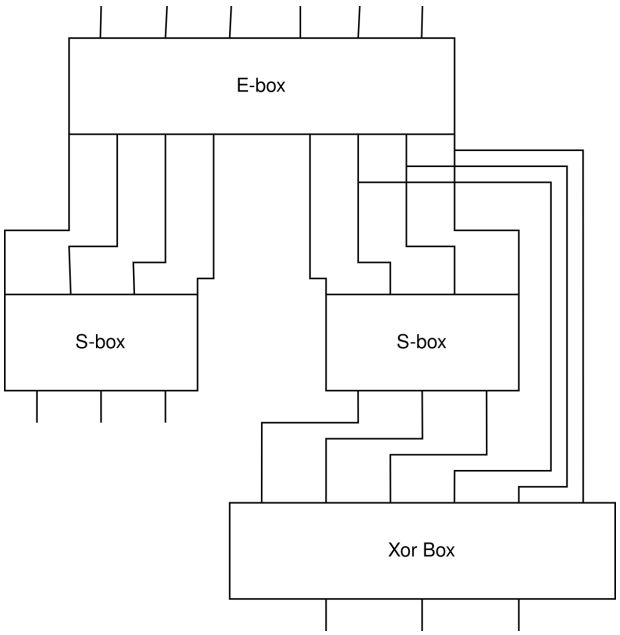


Figure 4: An example of box interaction.

In order to make sure that a box has its entire input set by the boxes before it when it notifies its output boxes, we will store the boxes of a round in the order of its topological sorting order. For example, in Figure 4, the E-box must come before all other boxes, and the Xor box must come after the S-box on the right.

3.3 Algorithm Optimisations

This section will discuss some optimizations made to Matsui's algorithm to reduce the search tree size for the branch-and-bound algorithm.

3.3.1 Starting from Best Previous Round Results

An intuitive first optimization is to keep some of the best differential characteristics found when running the algorithm for $n-1$ rounds and use them as inputs for the n rounds. There are two main reasons for doing this:

1. Accelerate the search by starting with an input likely to have a high probability. That will significantly reduce the number of searched states in the algorithm in the beginning

2. Quickly find iterative characteristics if they exist

3.3.2 Non-exhaustive S-boxes

In some cases, there is no need for the search algorithm to go through all the outputs of an S-box; instead, taking only the best one is enough. We call those boxes non-exhaustive.

For example, consider the S-boxes from the first F-function from DES. Their output gets combined in an XOR operation with the other half of the user's input and then forms the input of the second F-function. It suffices to take only the best output for those S-boxes because we can modify the user input to any value to obtain any input value for the second F-function.

In addition to that, all S-boxes from the final round that directly influence the output can be non-exhaustive.

3.3.3 Prefetching the Best Probability Inside a Round

Consider that the search algorithm is trying to determine the output of the i^{th} round. It will have to output a probability $p \geq \dfrac{\overline{B_n}}{\prod_{j<i} p_j}$.

If, at some point inside the round, we have set the result of the first k boxes and determine that even if we take the best probability for all the remaining boxes starting from the $k+1^{\text{th}}$ of that round, we cannot satisfy the condition above, we can prematurely skip this state and backtrack in order to find other good states, instead of continuing on this branch and having to backtrack later regardless. This technique further reduces the size of the search tree.

In order to do this, we need to keep track of the boxes that still need to determine their output but have their input completely determined. For those, we can calculate the best probability. For the rest of the boxes, we will assume that the probability is 1 to preserve correctness. We can efficiently do all this if we ensure that all the boxes whose input is determined that come after that box form a contiguous sequence in the round's box ordering for each box. That can be done by modifying the topological sort to have a Breadth-First Search approach instead of the classical Depth-First Search.

3.4 The Proposed Algorithm

Finally, we present the final form of our search algorithm. We split the algorithm into two parts: the cipher level search, where the user directly interacts with the cipher data structure, and the second round level part, where the rounds internally interact with the boxes it comprises.

Algorithm 5: Our search algorithm for finding best differential characteristics
Part 1 – Cipher level search

Input : C = the cipher to be analyzed; inp = the input to the cipher
Output: (out, p) = the best possible output and it's probability for the input
function *advance_state(C)*:
    while $C.round\_idx > 0 \wedge C.rounds(C.round\_idx).is\_determined$ **do**
        $C.round\_idx \longleftarrow C.round\_idx - 1$;
    if $C.round\_idx = 0 \wedge C.rounds(C.round\_idx).is\_determined$ **then**
        return *false*;
    Let $(out, p) \longleftarrow get\_next\_state(C.rounds(C.round\_idx))$;
    if $C.round\_idx = 0$ **then**
        $C.rounds(0).prob \longleftarrow p$;
    else
        $C.rounds(C.round\_idx) \longleftarrow p \cdot C.rounds(C.round\_idx - 1)$;
    if $C.round\_idx > 0 \wedge C.rounds(C.round\_idx).prob \cdot B_{n-C.round\_idx-1} < \overline{B_n}$ **then**
        $C.round\_idx \longleftarrow C.round\_idx - 1$;
        return *true*;
    if $C.round\_idx = 0 \wedge C.rounds(C.round\_idx).prob \cdot B_{n-C.round\_idx-1} < \overline{B_n}$ **then**
        return *false*;
    if $C.round\_idx < n - 1$ **then**
        Let $range \longleftarrow (0, out.size)$;
        $set\_input(C.rounds(C.round\_idx + 1), out, range)$;
        $C.rounds(C.round\_idx + 1).tresh \longleftarrow \dfrac{\overline{B_n}}{C.rounds(C.round\_idx).prob}$;
    if $C.round\_idx < n$ **then**
        $C.round\_idx \longleftarrow C.round\_idx + 1$;
    return *true*;
function *set_input(C, inp)*:
    Let $range \longleftarrow (0, C.rounds(0).input.size - 1)$;
    $set\_input(C.rounds(0), inp, range)$;
    $C.rounds(0).thresh \longleftarrow \dfrac{\overline{B_n}}{B_{n-1}}$;
    $C.round\_idx \longleftarrow 0$;
function *get_next_state(C)*:
    while $C.round\_idx < n$ **do**
        if *advance_state(C) = false* **then**
            return $(\emptyset, 0)$;
    $C.round\_idx \longleftarrow C.round\_idx - 1$;
    $\overline{B_n} \longleftarrow max(\overline{B_n}, C.rounds(-1).prob)$;
    return $(C.rounds(-1).output, C.rounds(-1).prob)$;

```
function find best differential(C, inp):
    Let out_best ⟵ ∅, p_best ⟵ 0;
    call set_input(C, inp);
    while true do
        Let (out, p) ⟵ get_next_state(C);
        if p > p_best then
            out_best ⟵ out, p_best ⟵ p;
        if p = 0 then
            break;
    return (out_best, p_best);
```

In this algorithm, accesing the -1^{th} element of an array returns the last element of that array.

Algorithm 6: Our search algorithm for finding best differential characteristics Part 2 – Round level search

```
function advance_state(R):
    while R.box_idx > 0 ∧ R.boxes(R.box_idx).is_determined do
        R.box_idx ⟵ R.box_idx − 1;
    if R.box_idx = 0 ∧ R.boxes(0).is_determined then
        return false;
    Let p ⟵ determine_next(R.boxes(R.box_idx));
    if R.box_idx = 0 then
        R.boxes(R.box_idx).prob ⟵ p;
    else
        R.boxes(R.box_idx).prob ⟵ p · R.boxes(R.box_idx − 1).prob;
    call notify all(R.boxes(R.box_idx));
    if R.box_idx = 0 then
        Let start_idx ⟵ 1;
    else
        Let start_idx ⟵ R.boxes(R.box_idx − 1).last_set + 1;
    for idx from start_idx to R.boxes(R.box_idx).last_set do
        R.boxes(idx).best_prob ⟵
          R.boxes(idx − 1).best_prob · get_best_prob(R.boxes(idx));
    Let actual_best_prob ⟵ R.boxes(R.box_idx).prob ·
      R.boxes(R.boxes(idx).last_set).best_prob/R.boxes(idx).best_prob;
    if R.box_idx > 0 ∧ actual_best_prob < R.thresh then
        R.box_idx ⟵ R.box_idx − 1;
        return true;
    else if R.box_idx = 0 ∧ actual_best_prob < R.thresh then
        return false;
    if R.box_idx < R.boxes.size then
        R.box_idx ⟵ R.box_idx + 1;
    return true;
```

```
function set_input(R, inp, range):
    R.boxes(0)[range.start, ..., range.end] ⟵ inp;
    R.boxes(0).is_determined ⟵ false;
    R.is_determined ⟵ false;
    R.box_idx ⟵ 0;
function get_next_state(R):
    while R.box_idx < R.boxes.size do
        if advance_state(R) = false then
            R.is_determined ⟵ true;
            return (∅, 0);
    R.box_idx ⟵ R.box_idx − 1;
    return (R.boxes(−1).output, R.boxes(−1).prob);
```

In the algorithm above, $R.boxes(R.box\_idx).last\_set$ represents the index of the last box that has it's input completely set if all boxes from 0 to $R.box\_idx$ have been determined.

3.5 Comparison Between our Algorithm and Other Tools

A handful of tools achieve similar goals to ours, so to justify the creation of another such tool, we will briefly discuss their advantages and disadvantages alongside ours.

Article	Advantages	Disadvantages
[BZL14]	• Implements a multitude of optimizations • Best performance out of branch-and-bound class of algorithms	• Optimisations are cipher specific, hard to generalize • Complex and difficult to implement
[BV13]	• Allows for different types of differentials • Memory-precision trade-off for good, practical approximations	• Designed specifically to cryptanalyze ARX ciphers • Cannot prove that the results are optimal
[RR22]	• Supports a wide variety of cryptanalysis techniques • Implements a multitude of cryptographic primitives	• Currently still in development
[Goh19]	• Better than state-of-the-art results	• Results obtained from the DNN are little understood • Long training times and hih resource usage
This thesis	• Simple, concise codebase • Supports a wide class of block ciphers	• Still in development • Currently only supports differential cryptanalysis

In conclusion of this comparison, our algorithm could be better in terms of performance. However, more can be done in that aspect, and with a better API, we could allow users to add custom optimizations based on cipher-specific properties. That would allow users to achieve performances as well as the other tools. Where we have an advantage is that we provide more flexibility regarding what can be done and analyzed with this tool. The library is also written in such a manner that allows us to implement other cryptanalysis techniques, such as multiple differential, truncated differential, and linear cryptanalysis, with little effort. That would make our tool comparable to CASCADA's wide range of techniques.

4 Implementation

In this chapter, we briefly discuss the tool's development process and the algorithm's implementation described in the previous section.

4.1 GenericCryptanalyzerLib

The tool is implemented in `C++17` and only depends on `Boost C++ Libraries`. The tool takes full advantage of Object-Oriented Programming, implementing different design patterns to keep a small, easy-to-maintain, and easy-to-extend codebase. It is is open-source and can be found on GitHub:
https://github.com/GabiTulba/GenericCryptanalyzer.

4.1.1 Ease of Use

We designed the tool with ease of use in mind. Users should be able to understand how to use the tool and build their desired cipher to cryptanalyze quickly and without much effort. We used builder classes for every type of box, a whole round, and a whole cipher.

4.1.2 Multithreading

The tool takes full advantage of the available resources. The search is divided between multiple threads in a `boost::thread_pool`, constantly fed tasks from a main runner thread. The parallelization process is straightforward, the only difficulty being synchronizing the global threshold of all cipher instances in the thread pool. That was done using as few mutexes as possible to maximize performance.

4.2 Documentation

Most of the code base is documented, and the documentation is publicly available at https://gabitulba.github.io/GenericCryptanalyzer/. The documentation was automatically generated using `Doxygen` from `JavaDoc`-like `C++` comments.

4.3 Testing

Unit tests for most of the basic classes and functionality have been added. Unit testing was done with the Boost.Test library. More detailed functionality was manually tested in the examples, and the results were confirmed with other results from the literature. See chapter 5 for more details.

5 Results

In this chapter, we discuss how we tested the correctness of our algorithm and what results we obtained compared to the literature. For the differential characteristics of DES, PRESENT, and Speck32, check the appendix of this thesis.

The following results were obtained on a machine with the following specifications:

- AMD Ryzen 7 5800H CPU, x86_64 architecture
- 8 CPU cores, two threads per core
- 256 KiB L1d and L1i caches
- 4MiB L2 cache
- 16MiB L3 cache
- 16GiB of DDR4 RAM

All results were obtained using multithreading with 16 threads.

5.1 Toy Example

Initial testing of the tool's functionality was done on the simple Substitution-Permutation Network block cipher presented in [Hey01] since the cipher had a small enough search space and allowed for fast total searches.

The cipher in that article was only analyzed up to three rounds, and the best differential characteristic found was $(\Delta X = 0b00_H, \Delta Y = 0606_H)$ with probability $\frac{27}{1024}$. We found a better differential characteristic: $(\Delta X = b0b0_H, \Delta Y = 0808_H)$ with probability $\frac{36}{1024}$.

The search time for the entire input space for this cipher was under a second.

5.2 Differential Cryptanalysis of DES

DES is a block cipher with a Feistel structure. Because of this, the cryptanalysis proved to be more complicated than expected and required all the optimizations presented above to achieve a decent running time. We analyzed the cipher up to 16 rounds for all inputs with a binary Hamming weight of at most 8. We got the same results as Matsui found in [Mat95] and we found the same iterative characteristic as in [BS91]: $(\Delta X = 1960000000000000_H, \Delta Y = 0000000019600000_H)$ with probability $\frac{1}{234}$.

Rounds 1-4	1	2^{-2}	2^{-4}	$\simeq 2^{-9.6076}$
Rounds 5-8	$\simeq 2^{-13.2153}$	$\simeq 2^{-19.9638}$	$\simeq 2^{-23.6121}$	$\simeq 2^{-30.4828}$
Rounds 9-12	$\simeq 2^{-31.4825}$	$\simeq 2^{-38.3532}$	$\simeq 2^{-39.3528}$	$\simeq 2^{-46.2235}$
Rounds 13-16	$\simeq 2^{-47.2232}$	$\simeq 2^{-54.0939}$	$\simeq 2^{-55.0936}$	$\simeq 2^{-61.9643}$

Table 5: Probabilities of the best characteristics for each round of DES up to 16 rounds.

The total search time for the entire cipher was $\sim 4h$.

In order to calculate the maximum number of rounds that can be broken faster than an exhaustive search, we will use Equation 5. We want to find the highest round index such that:

$$2N_D \leq |\mathcal{K}| \tag{15}$$

where $|\mathcal{K}|$ is the size of the key space, in this case, 2^{56}. For DES, 14 rounds are the most that respect the inequality.

5.3 Differential Cryptanalysis of PRESENT

PRESENT is a super lightweight block cipher with an SPN structure. For all that our algorithm is concerned, it is just a scaled-up version of the toy example, and therefore, it was simple to analyze. We analyzed it up to 15 rounds, for all inputs with a binary Hamming weight of at most 8, and got the same results as Wang in [Wan07]:

Rounds 1-5	2^{-2}	2^{-4}	2^{-8}	2^{-12}	2^{-20}
Rounds 6-10	2^{-24}	2^{-28}	2^{-32}	2^{-36}	2^{-42}
Rounds 11-15	2^{-46}	2^{-52}	2^{-56}	2^{-62}	2^{-66}

Table 6: Probabilities of best characteristics for each round of PRESENT up to 15 rounds

The search time for PRESENT was $\sim$ 45min.

Similarly to the previous section, using Equation 15, and knowing that $|\mathcal{K}| = 2^{64}$ we get that we can break at most 14 rounds of PRESENT.

5.4 Differential Cryptanalysis of Speck32

Speck32 is the smallest cipher from the Speck cipher family, and it is relatively new, being publicly released in 2013. It is an add-rotate-xor (ARX) cipher, the only non-linear component in rapport with the XOR operation being modular addition. We implemented and tested Speck32 to test our implementation of the algorithm presented by Biryukov et al. for computing pDDTs for modular addition.

We analyzed Speck32 up to 8 rounds for all inputs with a binary Hamming weight of up to 9 and with a probability threshold for the pDDT of 2^{-5} (which was the maximum size taking into consideration the memory limitation of the machine we did the testing on). We got the following results:

Rounds 1-4	1	2^{-1}	2^{-3}	2^{-5}
Rounds 5-8	2^{-9}	2^{-13}	2^{-18}	2^{-25}

Table 7: Probabilities of the best characteristics for Speck32 up to 8 rounds.

We obtained different differentials with equal probabilities as those found in [BV13] and [DM18]. The total search time for Speck32 was $\sim$ 30min, and the precomputation of the pDDT, which was not, but could easily be parallelized, took $\sim 2h$. Herefore, our algorithm performs better than the one from [BV13] since their search time was $\sim$ 240min.

Using the same Equation 15, we can see that all the differential characteristics found can be used to break the cipher.

6 Closing Remarks

6.1 Conclusion

We implemented a tool that does automated differential cryptanalysis based on Matsui's original algorithm, and we added a series of new improvements. The tool was designed with flexibility and extensibility in mind to allow for further improvements. Ease of use on the user's side was an essential aspect of the development process, but many improvements can be made.

Finally, we tested our tool on a series of block ciphers: DES, PRESENT, and Speck32, and we obtained the same results as in the literature. That proves that the tool has the potential to help with further research in both the development of new block ciphers and the cryptanalysis of already existing ciphers.

6.2 Further work and improvements

Much work is still to be done to extend the tool's capabilities regarding both the supported ciphers and the performance.

Here is a non-exhaustive list of improvements that still need to be done:

- Optimizations from [BZL14], [OMA95] and [AKM97];
- Support for linear cryptanalysis;
- Support for truncated differential cryptanalysis;
- Detailed documentation and extensive testing;
- Implement more primitives;
- Do profiling in order to spot eventual bottlenecks;
- Automatically detect non-exhaustive S-boxes;
- Ensure that the multithreaded algorithm properly scales with the number of threads;
- Implement a way to more concisely define ciphers by providing only a description file;
- Find a better way to generate inputs to the cipher analyzer.

Since the project is open-source, contributions can be added on GitHub: https://github.com/GabiTulba/GenericCryptanalyzer.

References

[AKM97] Kazumaro Aoki, Kunio Kobayashi, and Shiho Moriai. Best differential characteristic search of feal. In Eli Biham, editor, *Fast Software Encryption*, pages 41–53, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[Ber08] Daniel J. Bernstein. The salsa20 family of stream ciphers. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs: The eSTREAM Finalists*, pages 84–97, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BG11] Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: Theory and practice (corrected). Cryptology ePrint Archive, Paper 2011/115, 2011. `https://eprint.iacr.org/2011/115`.

[BGPT21] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. Cryptology ePrint Archive, Paper 2021/287, 2021. `https://eprint.iacr.org/2021/287`.

[Bih95] Eli Biham. On matsui's linear cryptanalysis. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, pages 341–355, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, Jan 1991.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Paper 2013/404, 2013. `https://eprint.iacr.org/2013/404`.

[BV13] Alex Biryukov and Vesselin Velichkov. Automatic search for differential trails in arx ciphers (extended version). Cryptology ePrint Archive, Paper 2013/853, 2013. `https://eprint.iacr.org/2013/853`.

[BZL14] Zhenzhen Bao, Wentao Zhang, and Dongdai Lin. Speeding up the search algorithm for the best differential and best linear trails. In *Inscrypt 2014: Information Security and Cryptology*, pages 259–285, 12 2014.

[DM18] Ashutosh Dhar Dwivedi and Pawel Morawiecki. Differential cryptanalysis of round-reduced speck. Cryptology ePrint Archive, Paper 2018/899, 2018. `https://eprint.iacr.org/2018/899`.

[DR00] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications*, pages 277–284, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Goh19] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. Cryptology ePrint Archive, Paper 2019/037, 2019. `https://eprint.iacr.org/2019/037`.

[Hey01] Howard Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26, 06 2001.

[LM01] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. Cryptology ePrint Archive, Paper 2001/001, 2001. `https://eprint.iacr.org/2001/001`.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 386–397, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[Mat95] Mitsuru Matsui. On correlation between the order of s-boxes and the strength of des. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, pages 366–375, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for arx: Application to salsa20. Cryptology ePrint Archive, Paper 2013/328, 2013. `https://eprint.iacr.org/2013/328`.

[MY93] Mitsuru Matsui and Atsuhiro Yamagishi. A new method for known plaintext attack of feal cipher. In Rainer A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT' 92*, pages 81–91, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[OMA95] Kazuo Ohta, Shiho Moriai, and Kazumaro Aoki. Improving the search algorithm for the best linear expression. In Don Coppersmith, editor, *Advances in Cryptology — CRYPT0' 95*, pages 157–170, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[RR22] Adrián Ranea and Vincent Rijmen. Characteristic automated search of cryptographic algorithms for distinguishing attacks (cascada). Cryptology ePrint Archive, Paper 2022/513, 2022. `https://eprint.iacr.org/2022/513`.

[Sch96] Bruce Schneier. *Applied cryptography : protocols, algorithms, and source code in C*. Second edition. New York : Wiley, ©1996, 1996.

[Sel08] Ali Aydın Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, Jan 2008.

[SM87] Akihiro Shimizu and Shoji Miyaguchi. Fast data encipherment algorithm feal. In *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 1987.

[SPQ03] François-Xavier Standaert, Gilles Piret, and Jean-Jacques Quisquater. Cryptanalysis of block ciphers: A survey, 06 2003.

[Sta77] United States. *Data encryption standard / U. S. Department of Commerce, National Bureau of Standards.* The Bureau ; National Technical Information Service Washington : Springfield, Va, 1977.

[Wan07] Meiqin Wang. Differential cryptanalysis of present. Cryptology ePrint Archive, Paper 2007/408, 2007. `https://eprint.iacr.org/2007/408`.

Appendices

A Best Differential Characteristics Discovered for DES

Rounds	Input Δ	Output Δ	Probability
1	$(\alpha_H, 00000000_H)$	$(00000000_H, \alpha_H)$	1
2	$(40080000_H, 04000000_H)$	$(00000000_H, 04000000_H)$	2^{-2}
3	$(40080000_H, 04000000_H)$	$(04000000_H, 40080000_H)$	2^{-4}
4	$(00000401_H, 00000040_H)$	$(02000400_H, 40204058_H)$	$\simeq 2^{-9.6076}$
5	$(400046d0_H, 02000000_H)$	$(02000000_H, 400046d0_H$	$\simeq 2^{-13.2153}$
6	$(405c0000_H, 04000000_H)$	$(405c0000_H, 84411346_H)$	$\simeq 2^{-19.9638}$
7	$(19600000_H, 00000000_H)$	$(00000000_H, 19600000_H)$	$\simeq 2^{-23.6121}$
8	$(1b600000_H, 00000000_H)$	$(1b600000_H, 00084010_H)$	$\simeq 2^{-30.4828}$
7	$(19600000_H, 00000000_H)$	$(00000000_H, 19600000_H)$	$\simeq 2^{-31.4825}$
9	$(1b600000_H, 00000000_H)$	$(1b600000_H, 00084010_H)$	$\simeq 2^{-38.3532}$
11	$(19600000_H, 00000000_H)$	$(00000000_H, 19600000_H)$	$\simeq 2^{-39.3528}$
12	$(1b600000_H, 00000000_H)$	$(1b600000_H, 00084010_H)$	$\simeq 2^{-46.2235}$
13	$(19600000_H, 00000000_H)$	$(00000000_H, 19600000_H)$	$\simeq 2^{-47.2232}$
14	$(1b600000_H, 00000000_H)$	$(1b600000_H, 00084010_H)$	$\simeq 2^{-54.0939}$
15	$(19600000_H, 00000000_H)$	$(00000000_H, 19600000_H)$	$\simeq 2^{-55.0936}$
16	$(1b600000_H, 00000000_H)$	$(1b600000_H, 00084010_H)$	$\simeq 2^{-61.9643}$

Table 8: List of the best differential characteristics found by our algorithm for DES. The numbers are written in hexadecimal notation. α can be any 32-bit value.

B Best Differential Characteristics Discovered for PRESENT

Rounds	Input Δ	Output Δ	Probability
1	$f000000000000000_H$	8888000000000000_H	2^{-2}
2	$f000000000000000_H$	0000888800000000_H	2^{-4}
3	$f000000000009000_H$	0000444400000000_H	2^{-8}
4	$0f00000000000900_H$	0000040400000404_H	2^{-12}
5	$0f00000000000900_H$	0000050500000505_H	2^{-20}
6	$000000000000f009_H$	0000050500000000_H	2^{-24}
7	$000000000000f009_H$	0000440000004400_H	2^{-28}
8	0000000000007007_H	0000050500000000_H	2^{-32}
9	0000000000007007_H	0000440000004400_H	2^{-36}
10	0007000000000007_H	0000040400000404_H	2^{-42}
11	0000000000007007_H	0000440000004400_H	2^{-46}
12	$0000000c00000002_H$	0000440000004400_H	2^{-52}
13	0000000000770000_H	0000440000004400_H	2^{-56}
14	$0000f00900000000_H$	0000050500000000_H	2^{-62}
15	$0000f00900000000_H$	0000440000004400_H	2^{-66}

Table 9: List of the best differential characteristics found by our algorithm for PRESENT. The numbers are written in hexadecimal notation.

C Best Differential Characteristics Discovered for Speck32

Rounds	Input Δ	Output Δ	Probability
1	00408000_H	00000002_H	1
2	00408000_H	$0002000a_H$	2^{-1}
3	28000010_H	81008102_H	2^{-3}
4	28000010_H	$8000840a_H$	2^{-5}
5	$02110a04_H$	$8000840a_H$	2^{-9}
6	$02110a04_H$	$850a9520_H$	2^{-13}
7	$0a604205_H$	$850a9520_H$	2^{-18}
8	94685008_H	$850a9520_H$	2^{-25}

Table 10: List of the best differential characteristics found by our algorithm for Speck32. The numbers are written in hexadecimal notation.