Optimizations in Algebraic and Differential Cryptanalysis

Theodosis Mourouzis

Department of Computer Science

University College London

A thesis submitted for the degree of

Doctor of Philosophy

January 2015

Title of the Thesis:

Optimizations in Algebraic and Differential Cryptanalysis

Ph.D. student:

Theodosis Mourouzis

Department of Computer Science

University College London

Address: Gower Street, London, WC1E 6BT

E-mail: theodosis.mourouzis.09@ucl.ac.uk

Supervisors:

Nicolas T. Courtois

Department of Computer Science

University College London

Address: Gower Street, London, WC1E 6BT

E-mail: n.courtois@cs.ucl.ac.uk

Committee Members:

1. **Reviewer 1**: Professor Kenny Paterson
2. **Reviewer 2**: Dr Christophe Petit

Day of the Defense:

Signature from head of PhD committee:

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other English or foreign examination board.

The following thesis work was written by Theodosis Mourouzis under the supervision of Dr Nicolas T. Courtois at University College London.

Signature from the author:

Abstract

In this thesis, we study how to enhance current cryptanalytic techniques, especially in Differential Cryptanalysis (DC) and to some degree in Algebraic Cryptanalysis (AC), by considering and solving some underlying optimization problems based on the general structure of the algorithm.

In the first part, we study techniques for optimizing arbitrary algebraic computations in the general non-commutative setting with respect to several metrics [42, 44]. We apply our techniques to combinatorial circuit optimization and Matrix Multiplication (MM) problems [30, 44]. Obtaining exact bounds for such problems is very challenging. We have developed a 2-step technique, where firstly we algebraically encode the problem and then we solve the corresponding CNF-SAT problem using a SAT solver. We apply this methodology to optimize small circuits such as S-boxes with respect to a given metric and to discover new bilinear algorithms for multiplying sufficiently small matrices. We have obtained the best bit-slice implementation of PRESENT S-box currently known [6]. Furthermore, this technique allows us to compute the Multiplicative Complexity (MC) of whole ciphers [23], a very important measure of the non-linearity of a cipher [20, 44].

Another major theme in this thesis is the study of advanced differential attacks on block ciphers. We suggest a general framework, which enhances current differential cryptanalytic techniques and we apply it to evaluate the security of GOST block cipher [63, 102, 107].

We introduce a new type of differential sets based on the connections between the S-boxes, named "general open sets" [50, 51], which can be seen as a refinement of Knudsen's truncated differentials [84]. Using this notion, we construct 20-round statistical distinguishers and then based on this construction we develop attacks against full 32-rounds. Our attacks are in the form of Depth-First key search with many technical steps subject to optimization. We validate and analyze in detail each of these steps in an attempt to provide a solid formulation for our advanced differential attacks.

KEYWORDS:

block cipher, hash function, compression function, S-box, GOST, CTC2, PRESENT, collisions, black-box cryptanalysis, differential cryptanalysis, algebraic cryptanalysis, distinguisher, optimization, non-linearity, algebraic degree, annihilator immunity, multiplicative complexity, bitslice implementation, circuit complexity, computational complexity, hardware implementation, matrix multiplication, truncated differentials, aggregated differentials, general open sets, Gauss error function, SAT solvers.

Acknowledgements

I would like to take the opportunity and thank all people who supported me all these four years during my PhD studies. Without their support, things would be much more different.

Firstly, I would like to thank my supervisor Dr Nicolas Courtois for his tremendous and unlimited support and guidance during my studies. I would like to thank him for posing very interesting problems to me and being available anytime to discuss and help me with any problem I was facing. He is a highly influential researcher and I learned from him to be passionate about my research. He was always motivating me to set ambitious goals. He is the one who made me understand that cryptology is very multi-disciplinary and you can use techniques from different other sciences to accomplish a specific target. He also motivated me always trying to achieve the best possible I can.

In addition, I would like to thank my family for their support, patience and love during all these years of my studies. They all supported me by any mean when I needed them. Furthermore, I would like to thank my PhD colleague Guangyan Song and the post-doc researcher Dr Daniel Hulme, who introduced me to the area of SAT solvers and computational complexity.

Last but not least, a big thank to the Department of Security and Crime of University College London and EPSRC (Engineering and Physical Sciences Research Council) for their generous financial support during all four years of my PhD studies. They were my only source of income and they additionally sponsored me to participate in different crypto-related activities, such as ECRYPT Summer Schools.

Contents

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

AC: Algebraic Cryptanalysis
AE: Affine Equivalence
AES: Advanced Encryption Standard
AI: Algebraic Immunity
ANF: Algebraic Normal Form
BGC: Bit-slice Gate Complexity
CC: Circuit Complexity
CICO: Constrained Inputs Constrained Outputs
CNF: Conjunctive Normal Form
CTC: Courtois Toy Cipher
DC: Differential Cryptanalysis
DES: Data Encryption Standard
DNF: Disjunctive Normal Form
DRM: Digital Rights Management
GC: Gate Complexity
GE: Gate Equivalent
PRP: Pseudo Random Permutation
MC: Multiplicative Complexity
MM: Matrix Multiplication
MITM: Man-in-the-middle-attack
RMITM: Reflection-Meet-in-The-Middle
SAC: Strict Avalanche Criterion
SAT: Satisfiability
SPN: Substitution Permutation Network
UNSAT: Unsatisfiability

Notation

$\mathbb{R}$: Field of real numbers

$\mathbb{Z}$: Ring of Integers

$\mathbb{F}_2$: Galois Field with two elements

$\mathbb{F}_{2^n}$: Galois Field with 2^n elements

$\mathbb{F}_2^n$: n-dimensional vector space over $\mathbb{F}_2$

$GL(n, 2)$: The general linear group of order n over $\mathbb{F}_2$

$HW(a)$: Hamming weight of an element a

$HD(a, b)$: Hamming distance between two elements a and b

f^r: For a function $f : D \to R$, f^r is the composition of f r times

$A||B$: is a concatenation of words A, B

A^k: is a concatenation of k copies of the word A

Part I

Cryptography and Cryptanalysis

1

The Science of Cryptology

The word cryptology comes from the combination of the ancient Greek words "$\kappa\varrho\upsilon\pi\tau\acute{o}\varsigma$", which means secret, and "$\lambda\acute{o}\gamma o\varsigma$" which means word. Cryptology as a science is very multi-disciplinary and combines techniques from a variety of different scientific fields, such as pure mathematics, computer science and electrical engineering.

Cryptology is an umbrella term that encompasses *cryptography* and *cryptanalysis*. Cryptography comes from the combination of Greek words "$\kappa\varrho\upsilon\pi\tau\acute{o}\varsigma$" (secret) and "$\gamma\rho\acute{\alpha}\phi\epsilon\iota\nu$", which means write or study, while cryptanalysis comes from the words "$\kappa\varrho\upsilon\pi\tau\acute{o}\varsigma$" and "$\alpha\nu\alpha\lambda\acute{\upsilon}\epsilon\iota\nu$", which means to loosen or to untie.

Traditionally, cryptographic techniques were widely used in order to establish secure communication between two (or more) parties in the presence of unauthorized third parties. We know several examples from history, as we are going to briefly discuss in the next section where that simple forms of encryption were applied to messages for preventing unauthorized parties to read these messages, especially in the area of military operations. Thus, the main objective was confidentiality. Nowadays, due to the great development of telecommunication systems and digital information, there is a greater need for cryptography. Cryptographic mechanisms are employed to fulfill additional security requirements than merely providing confidentiality, such as authenticity, data integrity, non-repudiation and anonymity. Below we summarize some of these security requirements and the reader is referred to [110, 86] for more information.

1. **Confidentiality:** An unauthorized third party must not be able to get any information related to the original content of the communication by eavesdropping the messages while in transit. Additionally, stored data should be protected against unauthorized access.

2. **Authenticity:** The receiver of a message wants to verify the origin. Thus, the receiver wants to make sure that the message is originated from the expected source.

3. **Data Integrity:** The receiver would like to verify that the content of the data was not modified while in transit. Such modification can be either accidental or on purpose.

4. **Non-Repudiation:** The sender should not be able later to deny that he sent a message.

5. **Anonymity:** Another security goal is anonymous communication which allows users to send messages to each other without revealing their identity. An example where anonymity is needed is in electronic-voting schemes [60].

In general, cryptography is related to the design of mechanisms which need to fulfill certain security objectives. On the other hand, cryptanalysis aims to defeat the security objectives and identify potential weaknesses and flaws in the system, which when exploited may compromise its security. In cryptanalysis we need to take into account different scenarios and make several assumptions, for example the **adversary's goal**, the **available resources** and the **type of access to the system**.

For example, if some cryptographic techniques are used to ensure confidentiality, then cryptanalysis focuses on techniques related to recovering either the original content of an encrypted message or some partial information about the initial message from the encrypted message, (without necessarily the knowledge of the secret key) or recovering the secret key. William Friedman was the first who used the word cryptanalysis and formally described it in a series of technical monographs that he wrote starting from 1918 [78]. We discuss more details regarding cryptanalysis in Chapter 3. In what follows, we provide a historical overview regarding cryptology and we discuss how it evolved since antiquity.

1.1 The Evolution of Cryptology

It is believed that cryptography was developed immediately after the writing was invented. Several examples in the history provide strong evidence that cryptography was used since ancient years, even if it did not exist as a formal science. It seems that many cryptography-related techniques were widely used for the transmission of confidential information, especially during military operations [78].

The earliest known use of cryptography is dated back to 1900 BC and found in non-standard hieroglyphs appearing in Egyptian monuments [78]. However, it is not known if that was an attempt for secret communication. An attempt to use cryptography during military operations was attempted by the ancient Greek Spartan army [81]. Spartans were using a scytale transposition cipher, where each unit or character of the message was shifted to another place.

The Roman emperor Gaius Julius Caesar (100BC-44BC) developed a simple transposition cipher, named "Caesar cipher" to be used for secret communication. This was developed in order to protect the content of the messages which included highly classified information regarding military operations and strategic plans. The main idea was that the messages were written with an alphabet shifted by three positions to the right. For example, he wrote a "D" instead of an "A". Since knowledge related to cryptanalysis was very limited, enemies were prevented in this way to interpret the content of the message.

Cryptographic techniques started becoming more popular in the area of military operations, due to the invention of the radio. In 1895, the first wireless radio was invented. Radio provided greater flexibility by eliminating the need to run wires between camps and headquarters. Radio technology was widely used during military operations and without cryptography, messages transmitted to or from the front lines could easily be intercepted by the enemy since radio is public. For example, someone could easily use a radio receiver and listen to the communication. During World War I, radio was extensively used for communication and thus major world powers have developed and used cryptographic techniques.

During World War II (1939-1945), cryptographic mechanisms were widely used and considered as essential weapons for the operations of both Allies and the German Axis. Simple mathematical rules for encryption were substituted by electro-mechanical rotor

cipher machines [78]. For example, the Germans were using a rotor machine called *Enigma*, which was invented by the end of World War I and a rotor stream cipher machine called *Lorenz* cipher, for secret communication [78]. The use of advanced forms of encryption pushed the need for the enhancement of current cryptanalytic techniques. The Polish Cipher Bureau was the first to break military texts enciphered on the Enigma cipher in December 1932, using theoretical mathematics and material supplied by French military intelligence. During the war, British cryptologists decrypted a vast number of messages enciphered on Enigma and these messages were a substantial aid to the Allies [78]. Additionally, British cryptographers at Bletchley Park had deduced the operation of the Lorenz machine without ever having seen such machine, by exploiting a mistake made by a German operator who has sent same massage twice (actually with some small alterations, such as abbreviations) encrypted with the same key. The recovering of such high-level information contributed to the end of the war quicker and at a lower cost.

Nowadays, cryptology is one of the most fundamental tools that underpins the world of information security and the digital economy. The development of telecommunication systems and digital information opened a wider range of new possibilities for the application of cryptographic primitives and made them essential for our security. Its range of applicability is very wide and has many applications such as, in E-commerce, Digital Rights Management (DRM), electronic voting, access control, cloud computing, ATM cards, computer passwords and many other. Billions of people are connected in the Internet and exchange data throughout their networks on a daily basis. Such data may contain personal information, billing account information, confidential business operations and any leakage regarding this information may be very harmful for the parties concerned.

1.2 Symmetric and Asymmetric Cryptography

The general setting is that we have two parties, the sender S and the receiver R, who would like to communicate over an insecure channel, without allowing an eavesdropper to obtain any information about the content of the communication. We assume that this channel allows the eavesdropper to capture the raw data exchanged during their conversation and S and R are assumed to exchange a small amount of information beforehand, over a secure channel, called the secret key.

The purpose of the encryption algorithms is to protect unauthorized third parties to reveal the content of an encrypted message transmitted over an insecure channel. Generally, a cryptosystem is a collection of algorithms: the encryption and the decryption algorithms (and the key generation algorithm). The encryption function E_k, takes as inputs a plaintext p and a secret key k and outputs a ciphertext $c = E_k(p)$, and the decryption function $D_{k'}$ (inverse of E_k), takes as input the ciphertext c and the secret key k' and recovers the initial plaintext p. Thus, we have that $D_{k'}(c) = p$.

The encryption function should be designed in a way such that an unauthorized third party cannot deduce information about the initial plaintext by intercepting the ciphertext. In addition, only the authorized receiver must be able to decrypt the ciphertext and thus the decryption algorithm must be secret. In many cases, encryption and decryption functions are related and thus both functions need to be secret. However, it is very impractical to keep secret the algorithms, since it means several algorithms are needed for different communication parties. The idea to overcome this problem is to consider parametrized encryption algorithms depending on the secret key k.

If the same secret key (i.e. $k = k'$) is used in both encryption and decryption, then the cryptosystem is called *symmetric cryptosystem* (cf. Definition 1 and Figure 1.1). In general, symmetric encryption algorithms are divided into two categories: block ciphers and stream ciphers. A block cipher is a deterministic algorithm which encrypts fixed-length groups of bits, called blocks, using the same key, while in a stream cipher the input is a continuous stream of plaintext bits, which is encrypted according to an internal state. This internal state is initialized by the secret key and an initial value and it is updated during the encryption process independently of the message. We study block ciphers in details in next chapter, while the study of stream ciphers is out of the scope of this thesis and more details can be found in [110, 80].

Definition 1. *(Deterministic Symmetric cryptosystem)*

Let $\mathcal{P}$ be the set of plaintexts (which we assume to be finite) and $\mathcal{C}$ the finite set of ciphertexts. Let $\mathcal{K}$ be the finite key space, such that $\forall k \in \mathcal{K}$ there is an encryption function E_k with respect to k, which is efficiently computable $\forall k \in \mathcal{K}, \forall x \in \mathcal{P}$ and given by,

$$E_k : \mathcal{P} \to \mathcal{C}.$$

The corresponding decryption function D_k is,

$$D_k : \mathcal{C} \to \mathcal{P} \cup \perp,$$

such that $D_k(E_k(p)) = p$ for all plaintexts $p \in \mathcal{P}$ ($\perp$ means invalid).
We assume that the key k is generated by an algorithm Gen which takes as input a security parameter 1^n and outputs a key k.

Remark 1. *Efficiently computable means computable in polynomial time in the size of the input.*

Using a symmetric cryptosystem we have the drawback that the two parties who would like to communicate over an insecure channel have to exchange a secret key over a secure channel beforehand. In 1970 Diffie and Hellman solved this problem by considering cryptosystems which do not necessarily need the encryption function to be secret for being secure, which are known as Public-Key or *asymmetric* cryptosystems (cf. Definition 2, Figure 1.1). Such cryptosystems are based on functions called trap-door one-way functions which are easy to evaluate but hard to invert unless some extra information is known. This extra information is called the private or secret key. The basic idea is that we have a pair of keys, the private key which defines the decryption function and the public key, which is publicly available, and defines the encryption function. Then, for communication one party can request the public key of the other party from a trusted source (often known as certificate authority or trusted authority).

Definition 2. *(Asymmetric cryptosystem)*

Let $\mathcal{P}$ be the set of plaintexts (which we assume to be finite) and $\mathcal{C}$ the finite set of ciphertexts. Let $\mathcal{K}$ the finite key space, such that $\forall p_k \in \mathcal{K}$ and $x \in \mathcal{P}$ there is an efficiently computable encryption function E_{p_k}, with respect to p_k, $E_{p_k} : \mathcal{P} \to \mathcal{C}$ and a $s_k \in \mathcal{K}$ corresponding to a decryption function D_{s_k},

$$D_{s_k} : \mathcal{C} \to \mathcal{P} \cup \perp,$$

such that $D_{s_k}(E_{p_k}(p)) = p$ *for all plaintexts* $p \in \mathcal{P}$.
We refer to p_k *as the public key and to* s_k *as the secret key and we assume that they are generated by an algorithm Gen which takes as input a security parameter* 1^n *and outputs the pair* (p_k, s_k).

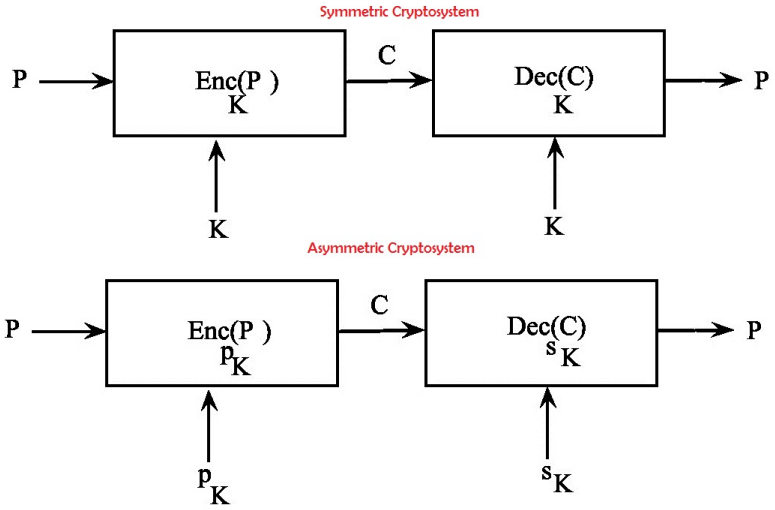

Figure 1.1: Symmetric and Asymmetric Cryptosystems - In a symmetric cryptosystem same key K is used for both encryption and decryption, while in asymmetric we have p_k for encryption and s_k for decryption

If a cryptosystem always outputs the same ciphertext for a given plaintext and key, then it is called *deterministic* cryptosystem. Deterministic encryption can leak information to an unauthorized third party because for example he can recognize that a given ciphertext corresponds to some interesting message and then he might be able to deduce more information about other ciphertexts when this message is transmitted.

To counter this problem, we consider the notion of probabilistic encryption, where randomness is used during encryption such that when encrypting the same message several times it will yield different ciphertexts. The notion of probabilistic encryption is very important, especially in an asymmetric cryptosystem where everyone can encrypt using the public key. For example, suppose we know that the plaintext is either "1" or "0" and a deterministic algorithm is used for encryption. Then, an adversary can encrypt both plaintexts using the public key and compare each result to the target ciphertext.

1.3 Hybrid Encryption

The main advantage of asymmetric cryptosystems is that they solve some crucial problems regarding key distribution and key management problems. In case of a symmetric cryptosystem, if the key is compromised, then an adversary gains total control over the system. Another drawback of symmetric cryptosystems is that a network requires a great number of keys to be used for communications among several parties and the keys must be distributed in secret. On the other hand, existing asymmetric schemes are slower ($\sim$1000 times slower than the symmetric ones on average [110]) and thus they are not very efficient for the encryption of long messages. Hence, we need to combine both cryptosystems to build a cryptographic scheme which exploits the advantages of both.

Combining both types of cryptosystems results in hybrid encryption, which combines the efficiency of a symmetric-key cryptosystem and the ability of an asymmetric-key cryptosystem to establish a secret key for the symmetric encryption [116]. In a hybrid construction, a symmetric cryptosystem is used for the encryption of a message, while an ephemeral secret key is generated using a protocol which is based on public-key cryptography. Assuming two parties S and R, the protocol for sharing the secret keys proceeds as follows,

1. S receives R's public key p_k from a trusted source, often referred to as certificate authority or trusted authority.
2. S generates a random session key K and computes $c = E_{p_k}(K)$
3. R uses his private key s_k to decrypt c and reveal the session key K
4. K is used to encrypt (and possibly authenticate) all the subsequent communications.

The authenticity of the public key is very important. If the public key is not obtained from a trusted source then Man-In-The-Middle (MITM) attacks are possible. MITM attacks require an attacker to have the ability to both monitor and alter or inject messages into a communication channel [110]. A Public Key Infrastructure is an arrangement that is used to bind public keys with respective user identities and another way is by digital signatures. On the other hand, proving the authenticity of

a symmetric key is a bit more complex. We need a way for nodes over a non-secure network to prove their identity in a secure manner and one example of such protocol is Kerberos [110].

1.4 Security Considerations

In the previous section we discussed the purpose of encryption and the two types of cryptosystems which are used for secure communication between two or more parties over an insecure channel. However, one very important consideration is what does "secure" means. Security is a multi-dimensional notion and depends on different parameters such as the goal, the abilities of an attacker and the available resources.

A major concern regarding the security of a cryptosystem is whether its security is enhanced by keeping the exact specifications of the algorithms used secret or make them publicly available. On one hand, we have security through obscurity, where the design or implementation is kept secret. A system relying on security through obscurity might proved in the future to have theoretical or implementation vulnerabilities. Thus, it is not recommended by standard bodies. On the contrary, we have Kerckhoff's principle [82] (cf. Theorem 1), which states that the security of a cryptosystem should solely depend on the secret key only. In case of keeping secret the underlying algorithm this automatically implies that several algorithms would be needed for different communication parties and this is not practical.

Theorem 1. *(Kerckhoffs' principle)*
A cryptosystem should remain secure even if everything about the cipher, except the secret key, is made public

Some of the advantages of this *open cryptographic design* are [80]:

1. The public design enables the establishments of standards
2. If the security relies on the secrecy of the algorithm, then reverse engineering the algorithm poses a threat against the system.
3. If the algorithm is public, then all researchers (including ethical hackers), will find and reveal any flaws in the design of the algorithm. This will improve the security of the algorithm.

4. It enhances the knowledge regarding cryptography and especially cryptanalysis.

However, we know several examples from history where Kerckhoff's principle was not followed. Suppose that the exact specifications of the "Caesar" cipher were leaked. Then, this would be fatal for the Romans as the enemies could easily decrypt all the messages. Another example is the German Enigma system in which the un-keyed mapping between keyboard keys and the input to the rotor array were kept secret and that was actually a big obstacle for the British cryptologists. Another example is the Russian encryption standard GOST (developed in the 1970s by the Soviets) which was kept secret until the dissolution of the USSR and finally released in 1994 [63]. It is claimed that GOST was used for the encryption of highly classified information and was not only a commercial block cipher [110]. Interestingly, there is a speculation that the Russian government would supply weak GOST S-boxes for those it would like to spy [110]. We will refer explicitly to GOST in the latter chapters.

Generally, there are several approaches to evaluate the security that a cryptosystems offers. In the rest of this section we consider two different concepts: unconditional and computational security.

A cryptosystem is unconditionally secure if it cannot be broken even with unlimited computational resources. However, this strongly depends on the attack scenario (cf. Chapter 3). A more formal definition is given by Shannon in his fundamental paper "Communication Theory of Secrecy systems" [113] under the name of *perfect secrecy* (Definition 3).

Definition 3. *(Perfect Secrecy)*
A cryptosystem has perfect secrecy if,

$$P(P = p|C = c) = P(P = p),$$

for all $p \in \mathcal{P}$ *and* $c \in \mathcal{C}$, *where* $\mathcal{P}$ *and* $\mathcal{C}$ *the plaintext and ciphertext space respectively.*

Perfect secrecy requires that the key is random and uniformly distributed and has the same size as the plaintext. In contrast, the message should not be assumed to be random. Informally, a cryptosystem has perfect secrecy, if an adversary cannot obtain any information about the plaintext by observing the ciphertext. An example of a symmetric cryptosystem, which has perfect secrecy is the *One-Time* pad (Definition 4).

Definition 4. *(One-Time pad, Frank Miller 1882)*

Let $n > 1$ be the length of the message, $\mathcal{P} = \mathcal{K} = \mathcal{C} = \mathbb{F}_2^n$. Then, the encryption and decryption are given as follows,

$$E_k(p) = (p_1 \oplus k_1, .., p_n \oplus k_n),$$

$$D_k(c) = (c_1 \oplus k_1, .., c_n \oplus k_n),$$

where p_i the i-th component of P and $\oplus$ the bitwise exlusive-or.

One-time pad is perfectly-secure if the key $K = (k_1, k_2, ..., k_n)$ is chosen uniformly at random from $\mathcal{K}$. However, unfortunately it has a number of drawbacks. Firstly, a long key must be securely stored and this is already highly problematic. Secondly, this limits the applicability of the scheme if we would like to send very long messages or if we do not know in advance an upper bound on the length of the messages that we would like to send. Lastly, if two distinct plaintexts p_1, p_2 are encrypted under the same key k, then the XOR of the resulting ciphertexts equals to $p_1 \oplus p_2$. By obtaining the ciphertexts we simultaneously obtain information regarding the XOR of the plaintexts. If the messages correspond to English-language text, then given the XOR of two sufficiently long messages, it is possible to perform frequency analysis and recover the messages themselves [80].

Note that unconditional security and perfect secrecy are not equivalent definitions. For example, a cipher which has perfect secrecy is unconditional secure against a ciphertext only attack scenario but might not be unconditionally secure in any other attack scenario (cf. Chapter 3). Perfect secrecy can be achieved only when the length of the key equals or exceeds the length of the message. Thus, it is not very practical to be implemented in real-life applications since real-time environments are subject to hardware constraints and efficiency in both software and hardware implementation is of great importance.

Additionally, assuming that an adversary has unlimited computational resources is a very unrealistic scenario and thus a more flexible model of security is required. This idea is captured by the notion of *computational security* (cf. Definition 6), which guarantees security up to a desired level.

Definition 5. *(Exhaustive Search)*

Exhaustive search (or brute force) is the method of trying all possible elements in the search space until the correct one is found.

Definition 6. *(Computational Security)*

A cryptosystem provides n bits of security, if the most efficient attack requires a computational effort equivalent to an exhaustive search (cf. Definition 5) on the set of all possible assignments on n bits.

Definition 6 implies that a cryptosystem is computationally secure and provides n bits of security if 2^n operations are computationally infeasible with the resources that are currently available or that will be in the near future. However, the parameter n is not clearly specified yet. The first who attempted to specify such a security parameter was John Nash in 1955. In a letter he sent to NSA, he mentioned that the security of any enciphering process, equivalently the effort in terms of computations to recover the key, should increase with increasing length of the key. The best would be that security grows exponentially with some parameter n, which is the length of the key [97].

However, the parameter n does not need to be the length of the key in all cases. For example, in the case of RSA we get approximately 80 bits security using a 1024-bit key [19]. Practical block ciphers used in symmetric encryption, as we will study in Chapter 2, have an iterative structure and make use of many iterations of substitution and permutation functions to obtain enough security. This iterative structure can be exploited by MITM attacks to break the cipher faster than 2^k, where k the length of the key. A recent paper claims, that the time complexity for breaking more or less any practical block cipher, can be written as $2^k.(1-\epsilon)$, for some $\epsilon > 0$ and thus there is always inevitable key bits loss [74].

In general, the aim of crypto designers is to design a cryptographic primitive which fulfills certain security objectives and is sufficiently secure related to the available computing power. However, the security level is quantified under several attack models and scenarios (cf. Chapter 3). Simultaneously, cryptographic primitives should be efficient enough to be implemented in both software and hardware. In the next chapter, we study the main paradigms followed in the area of design of block ciphers.

1.5 Structure of Thesis

The rest of the thesis consists of 8 different chapters. The content of each chapter is as follows.

- *Chapter 2:* We discuss the two basic design paradigms for block ciphers; the Feistel Network and the Substitution-Permutation Network (SPN). For each design paradigm, we study examples of such ciphers such as GOST and DES for Feistel and CTC2 for SPN. In addition, we discuss the Merkle-Damgård design paradigm for hash functions [59] and we study in details the GOST hash function which deviates from this most widely used design principle. Lastly, we present a black-box linear attack on the compression function of the GOST hash function.

- *Chapter 3:* We discuss the most important probabilistic cryptanalytic attacks and their variants. We focus on Differential Cryptanalysis (DC) and Linear Cryptanalysis (LC).

- *Chapter 4:* We briefly study *algebraic* attacks and *automated software cryptanalysis*. The basic idea behind such attacks is to derive the key by considering the underlying multivariate system of equations which describes the cipher and involves the plaintext, key, ciphertext and internal variables bits. In automated software cryptanalysis, we study automation of this process by converting it to different representations such that available software can derive the key. In this thesis we use SAT solvers at solving stage.

- *Chapter 5:* A basic requirement for a cipher to be secure is that it is distant enough from linear. In most block ciphers, the S-boxes are the only non-linear parts. Thus, the selection of S-boxes is a very crucial task for the designers. In this section, we provide a brief introduction to the theory of Boolean functions and discuss the four known measures of non-linearity, with more emphasis towards the notion of Multiplicative Complexity (MC) [20].

- *Chapter 6:* We study two fundamental NP-hard problems, the problem of Matrix Multiplication (MM) and the problem of combinatorial circuit optimization. We discuss a 2-step methodology based on SAT-solvers, which can solve small

instances of such problems. We construct new formulas for efficient MM of (sufficiently small) matrices and construct more efficient circuit representations for S-boxes used in prominent ciphers with respect to any meaningful metric related to cryptography and cryptanalysis.

- *Chapters 7:* We study DC with statistical distinguishers. We reduce the problem of recovering bits of the key to the problem of distinguishing two distributions, one corresponding to the right key and one corresponding to a wrong key.

- *Chapters 8:* An advanced form of DC is discussed and applied to GOST block cipher. Due to the key insertion via modulo 2^{32} addition the transitional probabilities between single differences are zero frequently for the majority of the keys. For this reason naive DC fails. We suggest a form of DC based on specific sets of differences constructed based on the connections between S-boxes from round to round, named *general open sets*. General open sets can be seen as a refinement of truncated differentials proposed by Knudsen [84]. We discuss a framework for construction of reduced-round distinguishers based on such sets and apply it to GOST.

- *Chapters 9:* We present a generic parametric framework of attacks, which can be used to attack GOST and some of its variants faster than brute force. This is achieved by exploiting the *self-similarity* property due to the weak key schedule and the *poor diffusion* inside the cipher.

Some of the results presented in this thesis were published in the following conferences, journals and book chapters:

1. Nicolas T. Courtois, Theodosis Mourouzis: *Advanced Differential Cryptanalysis and GOST Cipher*. Accepted for a 30 minute oral presentation and 6-pages paper in CD-ROM and web-proceedings at the 3rd IMA Conference on Mathematics in Defence, 2013.

2. Nicolas T. Courtois, Theodosis Mourouzis: *Enhanced Truncated Differential Cryptanalysis of GOST*. In SECRYPT 2013, 10th International Conference on Security and Cryptography.

3. Nicolas T. Courtois, Theodosis Mourouzis: *Propagation of Truncated Differentials in GOST*. In SECURWARE 2013, The Seventh International Conference on Emerging Security Information, Systems and Technologies.

4. Nicolas T. Courtois, Daniel Hulme, Theodosis Mourouzis: *Multiplicative Complexity and Solving Generalized Brent Equations With SAT Solvers*. In COMPUTATION TOOLS 2012, The Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking. (Best Paper Award).

5. Nicolas T. Courtois, Daniel Hulme, Theodosis Mourouzis: *Solving Circuit Optimisation Problems in Cryptography and Cryptanalysis*. In SHARCS 2012.

6. Nicolas T. Courtois, Daniel Hulme and Theodosis Mourouzis: *Solving Circuit Optimisation Problems in Cryptography and Cryptanalysis*. Appears in electronic proceedings of 2nd IMA Conference Mathematics in Defence 2011. A longer version is available at eprint: eprint/2011/475/.

7. Nicolas Courtois, Theodosis Mourouzis: *Black-Box Collision Attacks on the Compression Function of the GOST Hash Function*. In SECRYPT 2011, the 6th International Conference on Security and Cryptography.

8. Nicolas Courtois, Theodosis Mourouzis, Michal Misztal, Jean-Jacques Quisquater and Guangyan Song: *Can GOST Be Made Secure Against Differential Cryptanalysis ?* To appear in journal Cryptologia 2014.

9. Nicolas Courtois, Theodosis Mourouzis and Daniel Hulme : *Exact Logic Minimization and Multiplicative Complexity of Concrete Algebraic and Cryptographic Circuits*. Submitted to International Journal On Advances in Intelligent Systems, v 6 n 3&4 2013.

10. Theodosis Mourouzis and Nicolas Courtois : *Advanced Truncated Differential Attacks Against GOST Block Cipher and its Variants*. To appear in Springer Book entitled as Computation, Cryptography and Network Security 2015.

2

Symmetric Cryptography

In general, the designers of cryptographic primitives follow certain design principles that allow them to derive lower bounds on the complexities of successful attacks under some studied attack models. In this chapter, we study specific architectures and design criteria for symmetric primitives applied to block ciphers and hash functions. We study the two main design paradigms in the area of block ciphers; *Feistel Networks* and *Substitution Permutation Networks* (SPN) and we present one example of such a block cipher. In particular, we study the GOST block cipher [90], which follows the Feistel Network paradigm and the CTC/CTC2 [32] cipher, which follows the SPN paradigm. Furthermore, we briefly discuss the Data Encryption Standard (DES) in comparison with GOST since GOST was considered as the Soviet alternative of DES [28]. Lastly, we discuss the GOST hash function and we study the underlying compression function in terms of collision resistance properties.

2.1 Block Ciphers

Block ciphers (cf. Figure 2.1) are deterministic algorithms, that operate on fixed-length groups of bits, called blocks, with a bijective transformation specified by a symmetric key. In modern cryptographic primitives, such blocks typically have length 64 or 128 bits. Block ciphers are fundamental cryptographic primitives used for constructing other important cryptographic primitives, such as one-way functions, hash functions, message authentication codes, pseudorandom number generators and even stream ciphers [85].

Block ciphers mainly consists of two algorithms, the encryption algorithm denoted by $Enc(K, P)$ and the decryption algorithm denoted by $Dec(K, C)$ (the inverse of $E(K, P)$). They both accept as inputs n-bit data and a k-bit key K. More formally, these algorithms are defined in the following way.

$$Enc(K, P) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n,$$

$$Dec(K, C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n,$$

Correctness: $Dec(K, Enc(K, P)) = P, \forall K \in \{0,1\}^k, \forall P \in \{0,1\}^n$

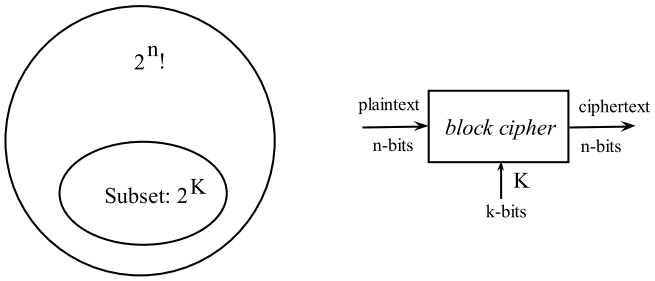

Figure 2.1: Block Cipher - A block cipher is used to encrypt blocks of n bits using a k-bit key K and its output is a n-bit ciphertext.

Generally, a block cipher which operates on n-bit blocks is a permutation of the form $\{0,1\}^n \rightarrow \{0,1\}^n$. We have in total $2^n! \backsimeq 2^{(n-1)2^n}$ permutations (obtained by Stirling's approximation). A block cipher which operates on n-bit blocks and uses k-bit keys, corresponds to 2^k distinct permutations on n bits. Informally, the design aim is to choose the 2^k permutations more or less uniformly at random from the set of all $2^n!$ permutations, i.e under a secret randomly chosen key it is computationally indistinguishable from a randomly chosen n-bit permutation (cf. Figure 2.1).

More formally, we think of a blockcipher E with a k-bit key and a n-bit blocksize as being uniformly chosen from the set of all possible permutations. For each key, there are $2^n!$ permutations and since any permutation may be assigned to a given key, there are $(2^n!)^{2^k}$ possible blockciphers. This is known as the Ideal-Cipher model [80].

Our basic intuitive understanding of block ciphers is due to the work of Claude Shannon. In his landmark paper "Communication Theory of Secrecy Systems" [113], he introduced the fundamental concepts of **confusion** and **diffusion**, which are still considered as the most widely used design principles in the design of ciphers.

Confusion refers to making the ciphertext dependent on the plaintext in a complex and unpredictable way due to the key insertion [86]. The aim is to make very hard for an attacker to find the symmetric key even if a large amount of known plaintext-ciphertext pairs obtained under the same key is available. Confusion is usually interpreted as local *substitution*, where certain groups of bits are replaced by other group of bits, following certain rules. Examples of such functions are integer addition, integer multiplication and S-boxes.

Diffusion refers to making each bit of the ciphertext dependent on the initial data in a complex and unpredictable way [86]. The main aim is to dissipate the statistical structure of the plaintext over the bulk of the ciphertext. Diffusion is realized by *permutations*, which means change of the order of the bits according to some algorithm. Any non-uniformity of plaintext bits must be redistributed across much larger structures in the ciphertext, making it in this way hard to detect non-uniformity.

In general, there are no formal definitions for confusion and diffusion and they are not absolutely quantifiable concepts. A very good description of these properties is provided by Massey [91]:

Confusion: The ciphertext should depend on the plaintext statistics in a manner too complicated to be exploited by the cryptanalyst.

Diffusion: Each digit of the plaintext and each digit of the secret key should influence many digits of the ciphetext.

Another important notion which is directly related to both confusion and diffusion is that of the **avalanche effect**. Informally, the avalanche effect is evident if when a slight change in the inputs causes a significant change in the output. For example, flipping a single bit in the output makes half of the output bits to flip. We refer to more details to this effect in Chapter 5.

Block ciphers are designed based on these twin principles. There are no formal rules for achieving these properties. It is in the tasks of the designer to use the appropriate combination of components that will result in a "secure" cipher. An important class of such ciphers based on these concepts is that of SPN, which we study in a later section.

2.1.1 Feistel Ciphers

Feistel cipher (cf. Definition 7) is a symmetric structure used in the design of block ciphers. It is named after the German physicist and cryptographer Horst Feistel. He (joint work with Don Coppersmith) introduced the concept of Feistel networks, while working on IBM's "Lucifer" cipher on 1973 [67]. Their work gained the respect of the U.S Federal Government, who adopted it to become the Data Encryption standard (DES). DES is based on Lucifer project with some changes made by the NSA [100].

Definition 7. *(Feistel Network)*

A Feistel cipher is an iterated cipher, which maps a 2n-bit plaintext block denoted by (L_0, R_0)*, where* L_0, R_0 *the left and right n-bit halves respectively, to a* $2n$*-bit block* (L_r, R_r) *after r-rounds of encryption.*

The two halves of the encrypted data after i-rounds $(1 \leq i < r-1)$ *are given by,*

$$L_i = R_{i-1} \tag{2.1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_{i-1}) \tag{2.2}$$

where K_i *is the i-th subkey derived from the secret key* K *and* f *a round function used for mapping n bits to n bits (cf. Figure 2.2). Key in normally introduced via the XOR operation and XORed with either the right or left half depending on the design. A separate Key Schedule Algorithm is used to derived round keys from the initial keying material.*

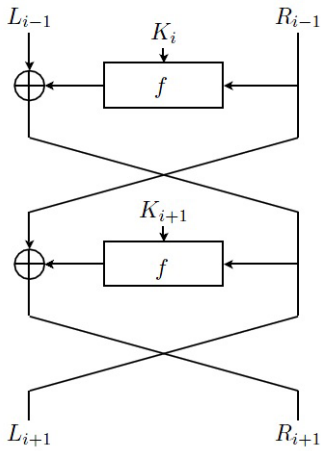

Figure 2.2: Feistel Network - The Feistel network design paradigm

In the last round there is no swap between the two halves L_{i-1}, R_{i-1} and the final output is given by,

$$L_r = L_{r-1} \oplus f(R_{r-1}, K_{r-1}) \tag{2.3}$$

$$R_r = R_{r-1} \tag{2.4}$$

The round function f does not need to be invertible, in contrast to SPN as we will see later. In many cases, a cryptographically weaker round function is used in order to decrease the cost of both software and hardware implementations. Intuitively, we expect this weakness to be compensated by increasing the number of iterations.

By construction, assuming that round keys are used in the reverse order, then the decryption process is exactly the same as the encryption process. This is an advantage in both software and hardware implementations, as the code or circuitry required for implementation is nearly halved.

2.1.2 Substitution-Permutation Network (SPN)

In an SPN (Definition 8, Figure 2.3) the round function is a combination of invertible functions, mainly *substitutions* and *permutations*. As a result of this the round function is invertible.

The *substitution layer* realizes the non-linear part of the cipher and usually consists of a parallel application of substitution tables (S-boxes), which operate on smaller blocks of data. Substitution layer introduces confusion. We also have a *permutation layer* which is a transformation that operates on the full block and is used for diffusion.

Definition 8. *An SPN is an iterated cipher, where the round function consists of three layers; the substitution layer, the permutation layer and the subkey application.*

In many cases, the only non-linear part of the cipher is the substitution layer. Informally, non-linearity means how distant a function is from being linear [89, 20]. Four measures of non-linearity are discussed in [20]. Non-linearity is a major theme in this thesis and it is studied separately in Chapter 4.

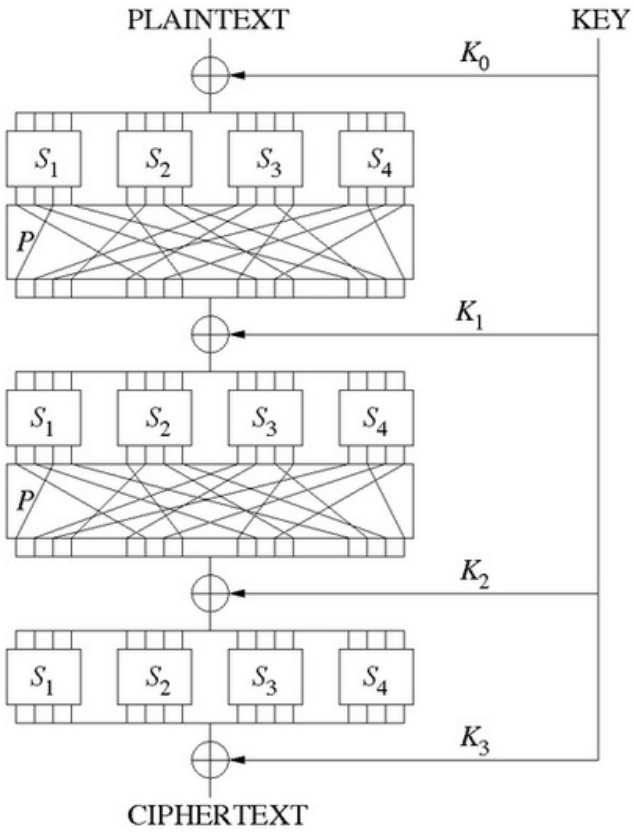

Figure 2.3: SPN - Substitution-Permutation Network design paradigm

For introducing non-linearity we usually employ S-boxes which are functions of the form

$$S : \{0,1\}^s \rightarrow \{0,1\}^{s'}. \tag{2.5}$$

Usually s, s' take small values for efficient implementation purposes (like around 3-8). Another reason for selecting small values is to make feasible the study of all such functions in this space with respect to their cryptographic properties. This will alow us to exhaustively search space and select to implement the ones which are optimal with respect to certain cryptographic criteria [89]. They can be implemented as *look-up* tables with 2^s entries.

2.1.3 CTC/CTC2

Courtois Toy Cipher (CTC) is a block cipher designed by Courtois, as a research tool cipher for performing experiments with algebraic attacks using a PC with a reasonable quantity of RAM [32]. Algebraic attacks is the class of attacks which aim to derive bits of the secret key or the full secret key by solving the underlying multivariate system of equations which involves bits of the plaintext, the ciphertext, the intermediate states and the key. We refer explicitly to algebraic attacks in Chapter 4.

The reason we describe the CTC cipher and not AES for example, is because of its small in size S-box (3-bits to 3-bits) which is selected at random and has no special algebraic structure. This makes it an ideal candidate for applying the circuit complexity algorithms described in Chapter 6. The motivation behind the design of CTC is to demonstrate that it is possible to break a cipher with sufficiently good diffusion using a small number of known (or chosen) plaintexts. Intuitively, if the input parameters (e.g. number of rounds N_r, number of S-boxes in each round denoted by B) are large enough, then the cipher is expected to be more secure.

CTC follows the SPN paradigm, where we have in each round a substitution layer followed by a diffusion layer. The S-boxes used in the substitution layer are random permutations on three bits with no special structure (cf. Table 2.1). From now on, for abbreviation we denote an S-box by $\{7,6,0,4,2,5,1,3\}$.

As a result of its small size, this S-box is considered as "algebraically weak", in a sense that it can be described by a small system of multivariate non-linear equations.

Table 2.1: CTC 3-bit to 3-bit S-box

Input	0 1 2 3 4 5 6 7
Output	7 6 0 4 2 5 1 3

It particular it can be described by 14 quadratic equations in 22 monomials [32]. It is conjectured in [54] that all ciphers with low I/O degree (cf. Definition 9) and described by sufficiently many I/O relations might be easier to be broken by algebraic attacks. However, this is just a speculations based on experimental results and further analysis is out of the scope of this thesis. More details can be found in [54] about the importance of I/O degree and the relation with algebraic attacks.

Definition 9. *(I/O Degree, Courtois, [32, 54])*

The I/O degree of a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, $f(x) = y$, *with* $x = (x_0, ..., x_{n-1})$, $y = (y_0, ..., y_{m-1})$ *is defined as the smallest degree of any algebraic relation*

$$g(x_0, .., x_{n-1}, y_0, ..., y_{m-1}) = 0,$$

that holds for every pair (x, y) *such that* $y = f(x)$.

Full avalanche effect is achieved after approximately 3-4 rounds, but at the same time the linear parts of the cipher are described by linear equations that are still quite sparse. Any algebraic attack against the cipher should be practically implemented and successful using a standard PC and only a few plaintext-ciphertext pairs.

Figure 2.4 illustrates the structure of the cipher for N_r rounds and with 2 S-boxes in each round ($B = 2$). The bits of the block size are ordered as $0, .., Bs - 1$ and bits in position 0,1,2 enter the first S-box, 3,4,5 the second S-box and so on. Each round i consists of the XOR with the derived key K_{i-1}, a parallel application of the B S-boxes and a linear diffusion layer D. The key size equals the block size. Thus, on average one known plaintext-ciphetext pair should be sufficient to recover the secret key.

We denote by $X_{i(j)}$ (for $i = 1, ..., N_r, j = 1, .., Bs-1$ the input bits to the i-th round after XORing with the derived key, while we denote by $Z_{i(j)}$ the corresponding output

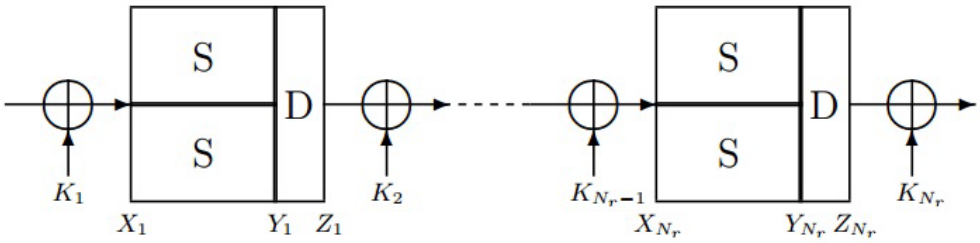

Figure 2.4: CTC Cipher - A toy cipher with B=2 S-boxes per round.

bits. The derived key K_i (in round i) is obtained from the secret key K_0 by a simple permutation of wires as follows,

$$K_{i(j)} = K_{0(j+i \mod Bs)} \tag{2.6}$$

The diffusion part D of the cipher is as follows:

$$Z_{i(j.1987+257 \mod Bs)} = Y_{i(j)}, \forall i = 1..N_r, j = 0 \tag{2.7}$$

$$Z_{i(j.1987+257 \mod Bs)} = Y_{i(j)} \oplus Y_{i(j+137 \mod Bs)}, j \neq 0, \forall i \tag{2.8}$$

Dunkelman and Keller [64] have shown that using LC one can recover a few bits of the key. This does not really compromise the security of a large key. Courtois suggested a tweaked version of CTC, named CTC2, which is expected to be more secure and flexible [33]. The main difference is that the key schedule of CTC2 has been extended to use keys of any size, independently of the block size.

In CTC2, the key size of key K, denoted by H_k, is not necessarily equal to the block size $B.s$ and it is computed in the following way.

$$K_{i(j)} = K_{(j+i.509 \mod (H_k))} \tag{2.9}$$

The diffusion part D of the cipher CTC2 is defined as follows:

$$Z_{i(j.1987+257 \mod Bs)} = Y_{i(j)} \oplus Y_{i(j+137 \mod Bs)} \oplus Y_{i(j+274 \mod Bs)}, j = 257 \mod (B*s) \tag{2.10}$$

$$Z_{i(j.1987+257 \mod Bs)} = Y_{i(j)} \oplus Y_{i(j+137 \mod Bs)} \tag{2.11}$$

As we have already mentioned, the reason we describe this cipher is because of its small S-box which we try to optimize with respect to several meaningful metrics in cryptanalysis in Chapter 6. It is a good example to illustrate the provable optimal aspects of the methodology presented in Chapter 6, which allows to compute exact bounds on the complexity of different circuit representations with respect to a given metric.

2.1.4 Data Encryption Standard (DES)

In 1972, the US Standards body National Bureau of Standards (NBS) (now known as National Institute of Standards and Technology (NIST)) identified the need for an encryption standard to be used for the encryption of unclassified and sensitive information. Until 1973, many designs were proposed, which were all rejected by NIST in collaboration with NSA. However, in a second round of the competition, the IBM proposal was finally accepted to be used by the government. The IBM research team proposed a block cipher based on a previous cipher, called Lucifer.

DES is a block-cipher based on the Feistel Network iterative structure. It maps 64 bits to 64 bits using keys of 56 bits and consists of 16 rounds. The key is actually 64 bits but only 56 are used. The remaining 8 bits are used for checking parity and they can be discarded.

In addition, we have an initial and final permutation, IP and FP respectively, which are such that $IP \circ FP = FP \circ IP = Id$. These permutations are of no cryptographic significance and they are used for facilitating loading blocks in and out of mid-1970s 8-bit based hardware [110].

The operations involved in the round function are as follows:

1. **Expansion**: The Expansion function takes as input a 32-bit half block and expands it into a 48-bit block by repetition of certain bits.

2. **Key Addition**: The key schedule of DES derives 16 48-bit sub-keys from the initial 56-bit key and each key is introduced in a different round via XOR operation.

3. **Substitution Table**: After expansion and key addition take place, the 48-bit vector is split in 8 consecutive 6-bit sub-vectors $x_1, x_2, ..., x_6$. Then, each x_i is given as input to a (6-bit to 4-bit) S-box giving a 4-bit output y_i.

4. **Permutation**: A permutation of bits is applied to the 32-bit output $y_1||y_2||...||y_6$. The permutation is such that each set of S-box's output bits are spread across 4 different S-boxes in the next round.

Remark 2. *(**Key-Schedule**) Given the initial 64-bit key, the function* Permuted Choice *1 (PC-1) derives 56 bits. The remaining eight bits are either discarded or used as parity check bits. The selected 56 bits are split in two 28-bit halves and each half is treated separately. In each round, both halves are rotated to the left by one or two bits specified for each round and then 48 bits are derived by Permuted Choice 2 (PC-2) function (24 bits from each half). More details are found in [100].*

Nowadays, DES is considered insecure to be used for many applications since brute force attack is possible due to the 56-bit key size being too small. In January 1999, *distributed.net* in collaboration with the Electronic Frontier Foundation broke a DES key in 22 hours and 15 minutes. In addition, many other theoretical weaknesses were revealed. Even though DES is insecure, an extension of DES named Triple-DES is considered practically secure [110]. Finally, after all these results, NIST withdrew DES from being the encryption standard and it was replaced by the Advanced Encryption Standard (AES) [56]. The security requirement for AES was that it must be at least as secure as 2-key 3-DES.

2.1.5 GOST Block Cipher

GOST is a 256-bit symmetric-key block cipher that operates on 64-bit designed by the former Soviet Union [122]. It is an acronym for "Gosudarstvennyi Standard" or Government Standard, as translated in English [90]. This standard was given the number 28147-89 by the Government Committee for Standards of the USSR [63].

GOST was developed in the 1970's and was classified as "Top Secret". In 1989, it was standardized for being used as an official standard for the protection of confidential information, but its specification remained confidential [122]. In 1990, it was downgraded to "Secret" and it was finally declassified and published in 1994, a short

period after the dissolution of the USSR. Then, the standard was published and translated to English [122, 90].

According to the Russian standard, GOST is safe to be used for the encryption of secret and classified information, without any security limitation. At the beginning of the standard it states that, "*GOST satisfies all cryptographic requirements and does not limit the grade of security of information to be protected*". There are some claims which state that it was initially used for high-grade communication, including military communications [110].

It seems that GOST was considered by the Soviets as an alternative to DES but also as a replacement of the rotor encryption machine FIALKA which was successfully cryptanalyzed by the Americans [110]. At the end of this chapter, we make an extensive comparative study between GOST and DES. Schneier states that the designers of GOST tried to achieve a balance between efficiency and security and thus they modified the existing US DES to design an algorithm, which has a better software implementation. The same source states that the designers were not so sure of their algorithm's security and they have tried to ensure high-level security by using a large key, keeping the set of S-boxes secret and doubling the number of rounds from 16 to 32. However, it is not true that GOST was just a Soviet alternative to DES since DES is a commercial algorithm used for short-term security for, while GOST has a very long 256-bit key which offers military-grade security. According to Moore's law, computing power doubles every 18-24 months, thus a 256-bit key cipher will remain secure for many years if no other shortcut attacks could be found (assuming computing power allows to recover approximately 80-bit keys at the moment). Additionally, GOST has been shown to have a very efficient hardware implementation and this makes it a plausible alternative for AES-256 and triple DES [102].

A comparison among several versions of GOST and other industrial ciphers in terms of Gate Equivalence (GE) (cf. Definition 10) is presented in [102] and in Table 2.2.

Definition 10. *([Informal], More details in [102]) One Gate Equivalent (GE) is equivalent to the silicon area of a 2-input NAND gate.*

From Table 2.2 we observe that the variant of GOST called GOST-PS, a fully Russian standard compliant variant (where the S-boxes of PRESENT are used) requires

only 651 GE. Additionally, the Russian Central Bank version, named GOST-FB, requires 800GE. AES-128 and DES require 3400 and 4000 GE respectively. These fact illustrate that GOST has a very efficient implementations and it is no surprise that it is implemented in many standard crypto libraries such as, OpenSSL, Crypto++, RSA security products and in many recent Internet Standards [63, 101].

Table 2.2: The GE required for the implementation of different block ciphers

Set Name	Gate Equivalent
GOST-PS	651
GOST-FB	800
DES	4000
AES-128	3400
PRESENT-128	1900

GOST was studied by many cryptographers such as Schneier, Biham, Biryukov, Dunkelman, Wagner and ISO cryptography experts [102, 110, 62]. All researchers always seemed to agree that it could be or should be secure, since no better way to break it than brute force was discovered. As a result of consensus among the cryptographic community, GOST was submitted to ISO 18033-3 in 2010 to become an international standard. Until 2010, all researchers in the cryptographic community claimed that "*Despite considerable cryptanalytic efforts spent in the past 20 years, GOST is still not broken*" [102].

Shortly after the submission, two attacks were published. One *single-key* attack against the full GOST block cipher was presented by Takanori Isobe at FSE 2011 [75]. Then, Courtois then suggested a new general paradigm for effective symmetric cryptanalysis called *Algebraic Complexity Reduction* [35]. Using this methodology, he constructed many more efficient attacks against GOST by reducing the problem of attacking the full 32 rounds to the problem of attacking 8 rounds, where a dedicated solver such as a SAT solver can be employed to derive the secret key.

2.1.5.1 Structure of GOST

The GOST block cipher is a 32-round Feistel structure of 256-bit level security. It uses its 256-bit key to encrypt 64-bit blocks (cf. Figure 2.5).

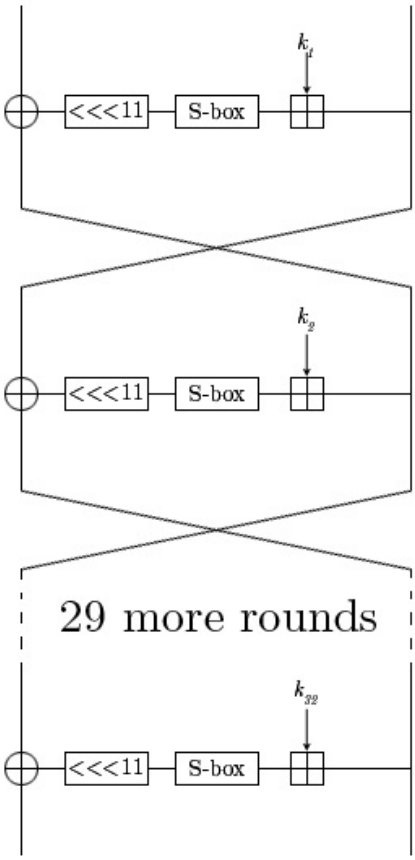

Figure 2.5: GOST Cipher - 32-rounds of a Feistel network to encrypt a 64-bit plaintext using a 256-bit key

A given 64-bit block P is split into its left and right halves P_L, P_R respectively. Given the key k_i for round i, the plaintext P is mapped to

$$(P_L, P_R) \rightarrow (P_R, P_L \oplus F_i(P_R)), \tag{2.12}$$

where F_i is the GOST round function. Given the round key k_i, the round function consists of the following sub-functions (cf. Figure 2.6).

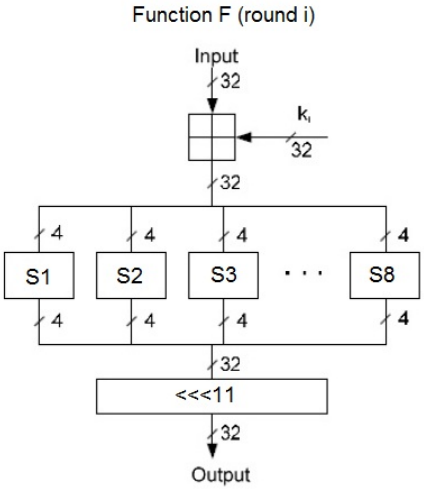

Figure 2.6: GOST's Round Function - Round function F_i

Firstly, the 32-bit right half is added with k_i (modulo 2^{32}). Then, the result is divided into eight 4-bit consecutive blocks and each block is given as input to a different S-box. The first 4 bits go into the first S-box S_1, bits 5-8 go into S_2 and so on. Then, the 32-bit output undergoes a 11-bit left circular shift and finally the result is xored to the left 32-bit half of the data.

Remark 3. *(Internal Connections in GOST) Let S_i for $i = 1, 2, ..., 8$ be the i-th S-box used in each round. Then, we can number the inputs of the S-box S_i by integers from $4i+1$ to $4i+4$ out of $1, .., 32$ and its outputs are numbered according to their final positions after the rotation by 11 positions: for example the inputs of $S6$ are 20,21,22,23 and the outputs are 32,1,2,3. We refer explicitly to internal connections since we use them to define sets of differentials that are suitable in DC as we will study in Chapter 8.*

2.1.5.2 Key Schedule

GOST has a relatively simple key schedule and this is exploited in several cryptanalytic attacks like in [34]. Its 256-bit key K is divided into eight consecutive 32-bit words $k_0, k_1, .., k_7$. These subkeys are used in this order for the first 24 rounds, while for the rounds 24-32 they are used in the reverse order (Table 2.3). Note that decryption is the same as encryption but with keys k_i used in the reverse order.

Rounds 1-8	**Rounds 9-16**
$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7$	$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7$
Rounds 17-24	**Rounds 25-32**
$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7$	$k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0$

Table 2.3: GOST: Key Schedule Algorithm

2.1.5.3 S-boxes

The Russian standard GOST 28147-89 does not give any recommendation regarding the generation of the S-boxes [63]. On the one hand, the fact that the S-boxes can be kept secret adds an extra security layer with approximately 354 extra bits of security (cf. *Lemma* 1). On the other hand, some problems might arise if the set of S-boxes is kept secret. For example, the generation and implementation of a set of S-boxes which is not cryptographically good would make the cipher less secure. Additionally, different algorithm implementations can use different set of S-boxes and thus can be incompatible with each other.

Even though the set of S-boxes can be kept secret, there are techniques to extract them from a chip very efficiently. We can reveal the values of the secret S-boxes by a simple black-box chosen-key attack with approximately 2^{32} encryptions [108, 70]. Informally, a black-box chosen-key technique is the technique where we are allowed to make queries to a given function using different keys but we are not aware of the internal structure of the function [108]. In all of the attacks we describe, we assume that the S-boxes are known to the attacker.

Table 2.4: Gost-R-3411-94-TestParamSet

S-boxes	GostR3411-94-TestParamSet
S1	4,10,9,2,13,8,0,14,6,11,1,12,7,15,5,3
S2	14,11,4,12,6,13,15,10,2,3,8,1,0,7,5,9
S3	5,8,1,13,10,3,4,2,14,15,12,7,6,0,9,11
S4	7,13,10,1,0,8,9,15,14,4,6,12,11,2,5,3
S5	6,12,7,1,5,15,13,8,4,10,9,14,0,3,11,2
S6	4,11,10,0,7,2,1,13,3,6,8,5,9,12,15,14
S7	13,11,4,1,3,15,5,9,0,10,14,7,6,8,2,12
S8	1,15,13,0,5,7,10,4,9,2,3,14,6,11,8,12

Lemma 1. *Suppose that the 8 4-bit to 4-bit S-boxes in GOST block cipher are kept secret. Then, the effective key size becomes 610 bits.*

Proof. Each S-box is a bijective Boolean function S of the form

$$S : \mathbb{F}_2^4 \to \mathbb{F}_2^4.$$

Thus, each function S is a permutation on the set $\{0, 1, 2, 3, ..., 15\}$.

There are in total 16! such permutations.

If all 8 S-boxes are kept secret, this is equivalent of $log_2(2^8.16!) = 354$ bits of secret information. Thus, the effective key size is increased to 610 bits from 256. □

One set of S-boxes called "*id-GostR3411-94-CryptoProParamSet*", was published in 1994, as part of the Russian standard hash function specification GOST R 34.11-94. Schneier claims that this set of S-boxes is used by the Central Bank of the Russian Federation [110]. At least two sets of S-boxes have been identified as being used by two major Russian banks and institutions [110].

We are aware of the following sets of S-boxes,

1. *Gost-R-3411-94-TestParamSet*: (Table 2.4) This set is used by the Central Bank of the Russian Federation [110].

2. *Gost28147-TestParamSet*: (cf. Appendix A.3, Table A.2) This set is used when GOST is used to process large amounts of data, e.g. in CBC Mode [101].

3. *GostR3411-94-SberbankHashParamset*: (cf. Appendix A.3, Table A.3) This set was used by a large bank, as part of the Russian standard hash function specification GOST R 34.11-94.

4. *GostR3411-94-CryptoProParamSet*: (cf. Appendix A.3, Table A.4) As appearing in RFC4357, this set was published in 1994 as a part of the Russian standard hash function specification GOST R 34.11-94 [63]. It has another four versions: A,B,C,D.

5. *GOST ISO 18033-3*: This set is specified in 1WD ISO/IEC 18033-3/Amd1 and was submitted for standardization [107]. This is claimed by Russian cryptologists to be the most secure version to use [107]. However, in the last chapter we show that an attack faster than brute-force can be also applied to this version and there is no evidence that it is more secure.

2.1.5.4 Addition Modulo 2^{32}

In addition to S-boxes, the GOST cipher uses addition modulo 2^{32} for key insertion. Modular addition is another source of introducing non-linearity in the cipher. There are ciphers which do not have S-boxes and the only non-linearity is via modular additions, like ARX ciphers [83]. The modular addition of two n-bit words x, y is algebraically described as follows,

$$(x, y) \mapsto z = x + y \mod 2^n \tag{2.13}$$

The resulting n-bit word $(z_{n-1}, .., z_0)$ is given by,

$$\begin{cases} z_0 = x_0 + y_0 \\ z_1 = x_1 + y_1 + c_1 \\ z_2 = x_2 + y_2 + c_2 \\ . \\ . \\ z_i = x_i + y_i + c_i \\ . \\ . \\ z_{n-1} = x_{n-1} + y_{n-1} + c_{n-1} \end{cases}$$

$$\text{where, } \begin{cases} c_1 = x_0.y_0 \\ c_2 = x_1.y_1 + c_1.(x_1 + y_1) \\ . \\ . \\ c_i = x_{i-1}.y_{i-1} + c_{i-1}(x_{i-1} + y_{i-1}) \\ . \\ . \\ c_{n-1} = x_{n-2}.y_{n-2} + c_{n-2}(x_{n-2} + y_{n-2}) \end{cases}$$

As we will explain in a later section, Multiplicative Complexity (MC), or equivalently the required number of multiplications, can be seen as a measure for the non-linearity of the cipher. The importance of MC is also discussed in [20]. The MC of the addition modulo 2^{32} is computer in Theorem 2.

Theorem 2. *(MC($\boxplus$), [42])*

The modular 2^n addition can be computed using at least $n-1$ multiplications. In other words its Multiplicative Complexity is $n-1$.

Proof. In characteristic 2 we have that

$$xy + (x+y)c = (x+c)(y+c) + c$$

Thus, we can compute the variables c_i, $1 \leq i \leq n$ using 1 multiplication for each, so $n-1$ in total.

On the other hand, each c_i contains a multiplication of two new variables so at least one multiplication is needed per c_i.

Thus, the multiplicative complexity of this operation is exactly $n-1$. □

The existence of modular addition 2^{32} makes the study of the cipher with respect to known forms of cryptanalytic attacks such as LC and DC much more complex. We refer explicitly to DC in a later chapter.

2.1.5.5 Comparative Study between GOST and DES

The designers of GOST block cipher aimed to achieve a balance between efficiency and security. One the one hand, they aimed to design a cipher which has very efficient software and hardware implementation and this was achieved by the very simple structure of its round function and the very simple key schedule. On the other hand, they wanted to achieve a high level security using a large number of rounds and a large key.

The main aim was the design of an algorithm which can be considered as a plausible alternative for existing industrial symmetric block ciphers like DES. GOST and DES have a very similar structure and they follow the Feistel Network design paradigm. Below we mention all major differences between GOST and DES.

1. GOST has 32 rounds, while DES has 16.

2. GOST has a 256-bit key, while DES has a 56-bit key. The secret key of GOST can be increased to 610 bits, if the S-boxes are kept secret.

3. GOST is cheaper than DES in software implementation since it has a simpler round function and a simpler key schedule algorithm.

4. GOST is cheaper to implement in hardware. One reason is due to its very simple round function. The DES round function has the substitution layer (which consists of 8 6-bit to 4-bit S-boxes) and it employs an expansion function and a permutation at the end of the round function. The substitution layer of GOST is the one-fourth in size of the substitution layer in DES.

5. The presence of modular addition 2^{32} makes the transitional probability in DC vary not only with the value of the input-output difference, but also with the value of the sub-key. Thus, as we will see in later sections GOST is not a Markov cipher. On the other hand, in DES we have a simple bitwise XOR, which makes the differential probability independent of the plaintext itself and thus DES is a Markov cipher [88]. We study Markov ciphers in Chapter 4.

6. The avalanche effect is slower to occur in GOST than in DES. In GOST, a change in one bit of input affects sometimes only one S-box after one round, which then affects two S-boxes in the next round and so on. Thus, a single bit change in the input affects the whole output after exactly 8 rounds, while in DES only 5 rounds are needed. This is due to the fact that in the DES round function we have an expansion permutation from 32 bits to 48 bits at the beginning and a very complex and carefully designed final permutation. In the case of GOST, we only have an 11-bit left circular shift and thus the diffusion is poor.

7. In GOST, each round function uses 32 out of 256 bits of the key, while in DES 48 out of 56 bits are used. That implies only approximately 12.5% of the key bits are used in one round in the case of GOST, while 86% in DES. In addition, due to the very simple key schedule in GOST, one can remove two rounds by guessing only 32 bits. This simplicity implies that GOST may not encrypt equally well, if the key has lots of zeros or large blocks of many consecutive zeroes. Thus, it may be possible to construct distinguishers for distinguishing keys of small Hamming weight.

2.2 Hash Functions

A one-way hash function $H(M)$ is a function which operates on a message M of arbitrary, but finite length and maps it to a fixed-length value h, called the hash value (cf. Figure 2.7).

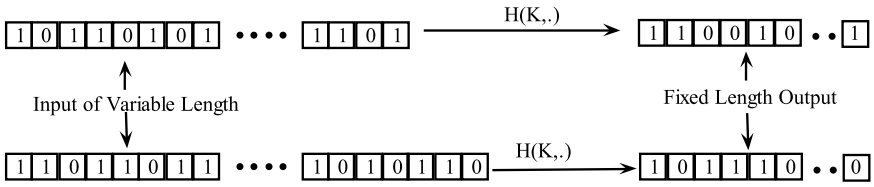

Figure 2.7: Hash Function - Hash function's operation

Formally, we will deal with a family of hash functions indexed by a "key" k. Thus, H is a 2-input function that takes as inputs a key k and a string M, and outputs a string $H^k(M) = H(k, M)$ (cf. Definition 11).

Definition 11. *(Hash function,[13])*

A hash function is a family of functions $H : K \times D \to R$, where D, R are the domain and the range of H and if $k \in K$ is a particular key, then $H^k : D \to R$ is an efficiently computable function defined for all $M \in D$ by $H^k(M) = H(k, M)$. This is the instance of H defined by k.

For a hash function $H : K \times D \to R$ and for any $k \in K$ and $y \in R$, we call as the pre-image of y under H^k, any element contained in the set $(H^k)^{-1}(y) = \{x \in D : H^k(x) = y\}$. Note that the key is not a usual cryptographic key and there are two main differences. Firstly, this key is not kept secret and secondly not all strings k correspond to valid keys and thus the key k is not chosen uniformly at random.

Hash functions are used in cryptography for ensuring certain security objectives such as integrity. However, they have to fulfill certain properties in order to be cryptographically good. For example, it should be computationally hard to find collisions (cf. Definition 12).

Definition 12. *(Collision)*
A pair $x_1, x_2 \in D$ with $x_1 \neq x_2$, such that $H^k(x_1) = H^k(x_2)$ for some k is said to be a collision.

A hash function for which it is difficult to find collisions, is called *collision-resistant*. Except of collision-resistance properties, a cryptographically good hash function needs also to satisfy the following security notions.

Definition 13. *(Security notions, [80])*
Given a key k (generated by a probabilistic algorithm), then for an instance $H^k(M)$ of $H : K \times D \to R$ for $k \in K$, we have the three main security notions that we would like to hold,

1.(Collision resistance): It is computationally hard to find distinct points $x_1, x_2 \in D$ with $x_1 \neq x_2$, such that $H^k(x_1) = H^k(x_2)$. In the absence of analytical weakness, the birthday paradox [110] means that the effort to compromise collision resistance expected to be no less than $2^{\frac{|D|}{2}}$ operations.

2.(Pre-image resistance): Given $y \in Image(H^k)$, it is computationally hard to find $x \in D$, such that $H^k(x) = y$. In the absence of analytical weakness, the effort required to compromise this property is expected to be $2^{|D|}$ operations.

3.(Second pre-image resistance) Given $x_1 \in D$ and $y \in Image(H^k)$, such that $H^k(x_1) = y$ it is computationally hard to find $x_2 \in D$, such that $x_2 \neq x_1$ and $H^k(x_2) = y$. In the absence of analytical weakness, the effort required to compromise this property is expected to be $2^{|D|}$ operations.

In the rest of this section, we treat hash functions as keyless and we study the main design paradigm for hash functions, named Merkle-Damgård transform [59].

2.2.1 How to Build a Hash Function

In this section we present an important methodology called the Merkle-Damgård transform (cf. Figure 2.8, [59]), that is widely used for constructing collision-resistant hash functions in practise. This methodology allows the conversion from any fixed-length

hash function to a hash function which can handle inputs of arbitrary length and on the same time preserves the collision resistance property of the former.

Informally, the methodology goes as follows. Firstly, we construct a one-way compression function f, which is used as an underlying component in the general architecture. A one-way compression function is a function that transforms a fixed-length input into a fixed-length output. Given the inputs, it is computationally easy to compute the image, however given an output it is computationally hard to invert.

A common way to build one-way compression function is from block-ciphers and several methods were suggested for such constructions [103]. There are two main reasons why we use block ciphers for constructing hash functions. Firstly, if we intent to implement both a block cipher and a hash function in an application, then in this way we save both space and implementation by reusing some components that we must anyway implement. Secondly, if we already have a deployed block cipher then it would be handy to construct a hash function out of this block cipher and then we might be able to leverage trust in the block cipher to make claims about the security of the hash function.

The Merkle-Damgård Construction Paradigm works as follows. Given a fixed-length compression function f for two inputs of length λ and with output of length λ, then we construct a variable length hash function H as follows:

- H : on string $x \in \{0,1\}^*$ of length L, do the following

  1. Set $B := \lceil \frac{L}{\lambda} \rceil$ (number of blocks in x). Pad x with zeroes so its length is a multiple of λ. Consider the sequence of λ-bit blocks $x_1, .., x_B$. Set $x_{B+1} := L$, where L is encoded using exactly λ bits.
  2. Set $z_0 = IV := 0^\lambda$ (This is called the initialization vector and is arbitrary and can be replaced by any constant)
  3. For $i = 1, ..., B+1$, compute $z_i := f(z_{i-1}, x_i)$
  4. Output z_{B+1}

If the underlying compression function is collision-free, then it is proved that the Merkle-Damgård construction is collision-free [80]. This property makes the job of designing practical collision-resistant hash functions much easier. In the next section,

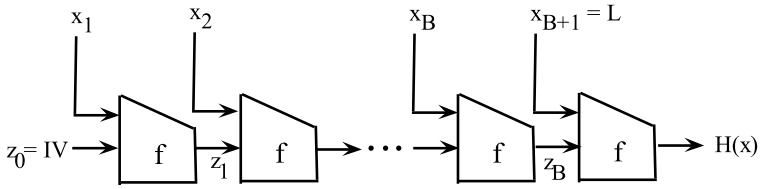

Figure 2.8: Merkle-Damgård Construction Paradigm - General design of a hash function which follows Merkle-Damgård paradigm improved with padding and length

we study the GOST hash function [2], a hash function which deviates from this widely-used and accepted design paradigm. A collision on the compression function of GOST hash function is also presented.

2.2.2 GOST Hash Function

The GOST hash function, defined in the Russian government standard GOST R 34.11-94 is a cryptographic hash function, which processes message blocks of size 256 bits and outputs 256-bit hash values. If the number of bits of the entire message is not a multiple of 256, then the message is padded by appending as many zeroes to it as are required to bring the length of the message up to a multiple of 256 bits [2].

The high level structure of the GOST hash function is more complex than the other common hash functions, such as MD5 and SHA-1, which follow the Merkle-Damgård design principles. In addition, GOST has an extra checksum, which is computed over all input message blocks and is given as input to the last compression function. In Figure 2.9 we recall the structure of the GOST hash function. Note that f stands for the compression function $\mathbb{F}_2^{256+256} \to \mathbb{F}_2^{256}$.

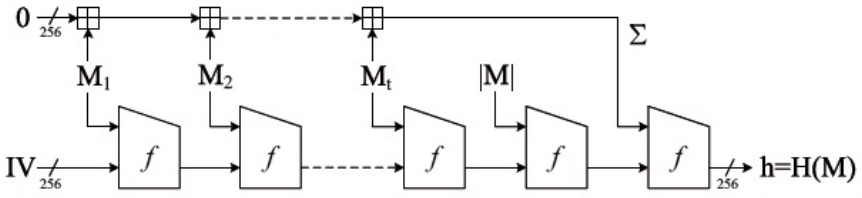

Figure 2.9: GOST Hash Function - High-level description of GOST hash function

The hash value $h = H(M)$ is computed recursively by the following equations:

$$H_0 = IV \tag{2.14}$$

$$H_i = f(H_{i-1}, M_i) \tag{2.15}$$

$$H_{t+1} = f(H_t, |M|) \tag{2.16}$$

$$h = f(H_{t+1}, \Sigma), \tag{2.17}$$

where Σ is the sum of all the message blocks M_i computed modulo 2^{256}. IV is the fixed initial vector given to the compression function and $|M|$ is the size in bits of the entire message.

The compression function f of the GOST hash function consists of three basic sub-algorithms; the *Key Generation*, the *State Update Function* and the *Output Transformation* (Figure 2.10).

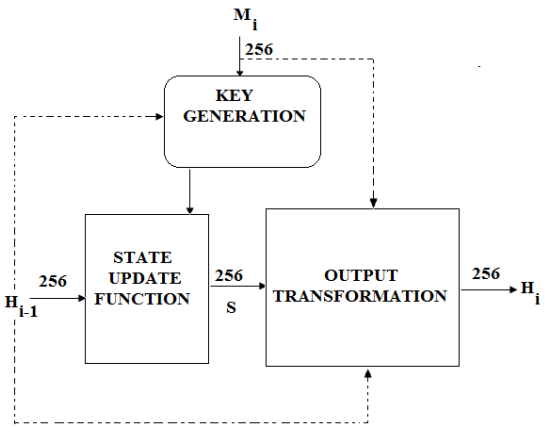

Figure 2.10: GOST Hash Function Main Components - The three different components of the compression function

2.2.2.1 State Update Transformation (SUT)

The *State Update Transformation* (Figure 2.11), is the "four encryptions in parallel" component of the underlying compression function f. The 256-bit intermediate hash value H_{i-1} is written as the concatenation of four 64-bit words in the form $h_3||h_2||h_1||h_0$

and each 64-bit word is encrypted under the GOST block cipher (E). This results in the 256-bit value $S = s_3||s_2||s_1||s_0$, where $s_i \in \{0,1\}^{64}$ for $0 \leq i \leq 3$ as described below,

$$s_i = E(k_i, h_i). \tag{2.18}$$

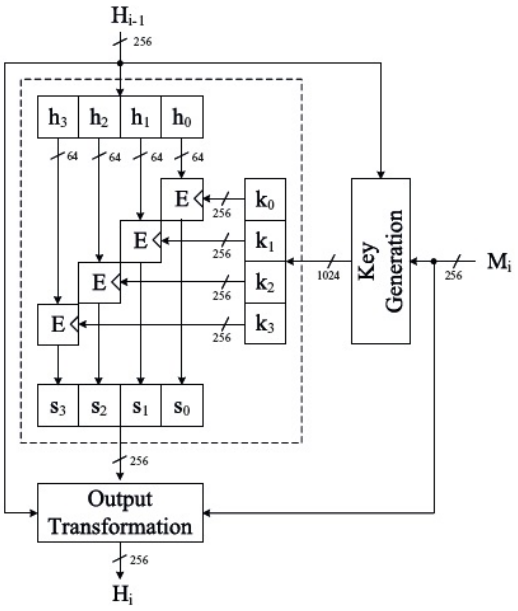

Figure 2.11: GOST's Compression Function - The compression function of GOST [95]

Here $E(K, P)$ stands for the encryption of the given 64-bit plaintext P with a 256-bit key K. The complexity of the evaluation of the compression function depends strongly on the cost of evaluating the "four parallel encryptions". As we will see in our analysis later, all the other components involved in computing the hash values are very inexpensive to implement since they are all linear.

2.2.2.2 Key Generation (KG)

The *Key Generation* sub-procedure of GOST combines the intermediate hash value H_{i-1} and the message block M_i, using simple linear transformations to output a 1024-bit key $K = k_3||k_2||k_1||k_0$.

The 256-bit subcomponents k_0, k_1, k_2, k_3 are computed using the following formulas, where A and P are linear transformations and α is a constant given by

$$1^8||0^8||1^{16}||0^{24}||1^{16}||0^8||(0^8||1^8)^2||1^8||0^8||(0^8||1^8)^4||(1^8||0^8)^4.$$

$$k_0 = P(H_{i-1} \oplus M_i) \tag{2.19}$$

$$k_1 = P(A(H_{i-1}) \oplus A^2(M_i)) \tag{2.20}$$

$$k_2 = P(A^2(H_{i-1}) \oplus \alpha \oplus A^4(M_i)) \tag{2.21}$$

$$k_3 = P(A(A^2(H_{i-1}) \oplus \alpha) \oplus A^6(M_i)) \tag{2.22}$$

For the definition of the linear transformation A and P, we refer to [2], since they are not needed in the scope of this thesis.

2.2.2.3 Output Transformation (OT)

The *Output Transformation* is the final transformation, which is applied in order to produce the value H_i. The final value H_i is computed using the linear and invertible transformation $\psi : \{0,1\}^{256} \to \{0,1\}^{256}$ given by,

$$H_i = \psi^{61}(H_{i-1} \oplus \psi(M_i \oplus \psi^{12}(S))) \tag{2.23}$$

The explicit representation of the transformation ψ is the following:

$$\psi(X) = (x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_{12} \oplus x_{15})||x_{15}||x_{14}||...||x_1, \tag{2.24}$$

where the vector $X \in \{0,1\}^{256}$ is written as $x_{15}||x_{14}||....||x_0$ with each $x_i \in \{0,1\}^{16}$.

As we discussed, GOST except for its iterative structure which follows precisely the Merkle-Damgård construction, employs an extra modular addition over all the message inputs and thus it differs from traditional hash functions. Wagner proved that a generalized form of the birthday paradox can be applied and then collision attack can also be extended with this sum [119]. Thus, this sum does not really enhance the security of the hash function.

Another important observation, is that in the compression function of GOST hash function we have four parallel encryptions and the final output is used to mix the

four results of the encryptions. However, since there are many linear dependencies this degrades the security of the underlying compression function. We study collision-resistance property of this compression function in the next section.

Remark 4. *Usually we have single-block-length compression functions and the hash function outputs the same number of bits as processed by the underlying block cipher and double-length hash functions which output twice the number of bits processed by the underlying block cipher. The main motivation behind double-length constructions is inspired by the fact that given two hash functions h_1, h_2, if either h_1 or h_2 is a collision resistant hash function, then $h(x) = h_1(x)||h_2(x)$ is a collision resistant hash function and thus if h_1 and h_2 are applied independently then one could hope finding a collision for $h(x)$ requires twice the effort to find a collision for one of them [86].*

2.2.3 Attacks on the Compression Function of GOST

Regarding the security analysis of GOST hash function with respect to collision, pre-image and 2nd pre-image attacks, most important attempts are by Mendel *et al* [95]. They present attacks against full hash function as extensions of attacks on the underlying compression function. The attacks they have presented lie in the area of *specific* or *structural* attacks, as they exploit the efficiency of constructing fixed points in the GOST block cipher, which is the major non-linear component inside the compression function. At first stage, they construct a collision attack on the underlying compression function of complexity 2^{96} evaluations of the compression function, which is extended to a collision attack against full hash function with complexity 2^{105} evaluations of the compression function. The extension is achieved using *multi-collisions* [77] and a generalization of the birthday paradox [119].

In this section, we study the underlying compression function. We study how to obtain collisions for the compression function using black-box algebraic techniques. A black-box technique was also presented by Mendel *et al* [94], however our technique is fundamentally different. Unfortunately, an extension of such an attack to the full hash is not always guaranteed and despite our efforts we have not achieved it so far.

Following the description of GOST Hash function, the high-level description of the compression function f can be schematically represented as in Figure 2.12.

The most important thing to note which we exploit in our attacks is that the transformation L is a linear transformation on the 512-dimensional space $M_i \times H_{i-1}$ and it

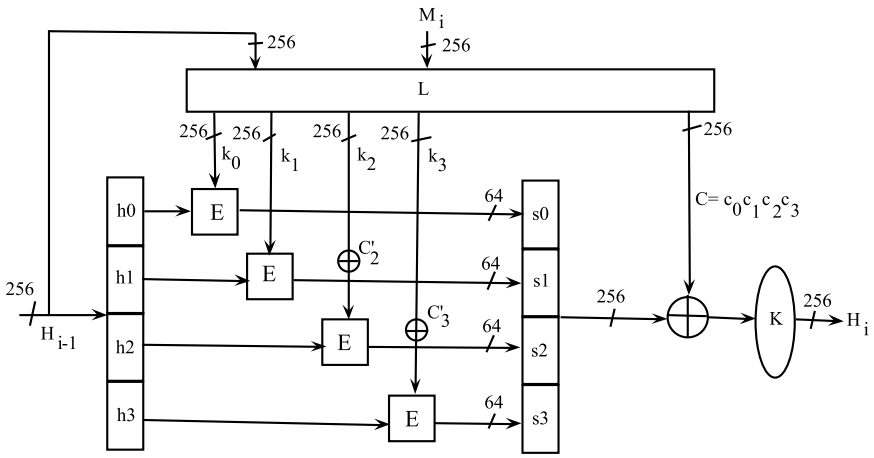

Figure 2.12: GOST Hash Function: Compression Function - A schematic representation of the compression function f in GOST Hash Function

outputs 1024+256 bits. In addition, we have at the end a bijective linear transformation K on the 256 bits resulting from the XOR of the two 256-bit vectors $S = s_3||s_2||s_1||s_0$ and C as depicted in Figure 2.12. The two 256-bit vectors C'_2, C'_3 are just affine constants.

The main idea of our attacks follows the paradigm of *Constrained Inputs Constrainted Outputs* (CICO) which is a method of solving an equation $f(x) = y$ for $x \in X$ and $y \in Y$, for certain subspaces X and Y. This term was invented by the designers of Keccak SHA-3 [14].

The key idea in such attacks is the construction of a restriction of a given function on a proper subspace of the input space $M_i \times H_{i-1}$, which induces a smaller linear-subspace on the 256-dimensional output space. For example, if we select a 192-dimensional linear subspace of the 512-dimensional input space, such that the output space is still large enough, then we have managed to construct a proper restriction of the compression function f, say $f\,|_{res}$. Hence, any collision for $f\,|_{res}$ is also a collision for f. This is a form of *Affine Reduction Property*. Specifically, for the case of GOST hash function, an affine reduction property can be found as follows (cf. Definition 14).

Definition 14. *(Affine Reduction Property)*

1. *Consider 384 linear equations on the 512-dimensional input space $M_i \times H_{i-1}$. This corresponds to selecting a proper 128-dimensional proper subspace V in $M_i \times H_{i-1}$.*

2. *Consider the image of* $f\mid_{res}$ *on the reduced space* V *and call it* $W \subset H_i$. *The main aim is that the selection of the inputs results in additional linear equations on the output space. If 64 such linear equations are obtained, then we have a restricted version of* f *as follows:*

$$f\mid_{res}: V \to W, \dim V = 128, \dim W = 192 (= 256 - 64), \tag{2.25}$$

where each time the input is a member of a reduced linear space of 2^{128} *inputs.*

Remark 5. *In order for collisions to exist for* $f\mid_{res}$, *we need that the inequality* $dimV > \frac{dimW}{2}$ *is satisfied. If this inequality does not hold, then* $f\mid_{res}$ *might have no collisions, as it might be injective.*

In the next subsection we study two ways to achieve this affine reduction property and leading to collision attacks on the compression function.

2.2.3.1 Black-Box Collision Attacks

Algorithm 1 describes the attack on the compression by Mendel *et al.* [95]. This attack is also a black-box attack, however it is not clear that it can be extended to the full hash [95].

Algorithm 1 Mendel *et al*'s Attack against f

1. Fix k_0 for the first instance of the GOST cipher E (cf. Figure 2.12).
2. Fix the input H_{i-1} of the first instance of the cipher GOST, denoted by h_0 (cf. Figure 2.12).
3. Fix $c_0 = s_0$ (cf. Figure 2.12).

Analysis of Algorithm 1: Step 1 corresponds to 256 linear equations in $M_i \times H_{i-1}$, since $k_0 = P(H_{i-1} \oplus M_i)$, where P is a linear transformation. Steps 2 and 3 induce another 64+64 linear equations on the input space. Once k_0 and h_0 are fixed, then $s_0 = E(k_0, h_0)$ is also fixed. Let x be the result of xoring vectors c and vector s. Based on the selection of our parameters we have that $x_0 = 0$ and this leads to 64 linear equations on the output space H_i. Hence, we have constructed a restriction of the compression function f which follows the affine reduction property.

Next, we describe a different black-box attack which results again in the construction of a restriction of the function f, which allows collisions to be found.

Algorithm 2 Our Black-Box Algebraic Collision Attack against f

1. Fix the keys k_0 and k_1 for the first two instances of the cipher GOST.
2. Fix the inputs h_0 and h_1 for the first two instances of the cipher GOST.
3. Fix $c_0 = c_1$ (cf. Figure 2.12).

Analysis of Algorithm 2: Step 1 corresponds to 256 linear equations on the space $M_i \times H_{i-1}$. Steps 2 and 3 induce another 64+64 linear equations. Let x be the result of xoring the vectors c and s. Based on the selection of our parameters we have that $x_0 = x_1$ and this leads to 64 linear equations on the output space H_i. Hence, we have constructed again a restriction of the compression function f, which follows the affine reduction property.

In both attacks we constructed a restriction of f, such that its output space has 2^{192} elements. By applying the birthday paradox we can find a collision with complexity 2^{96} evaluations of the compression function. Since our input space is greater than 2^{96}, then this method is sufficient for finding a colliding input pair. However, extension of this technique does not allow for constructing a collision for the full hash.

3

Probabilistic Methods in Cryptanalysis

"*History has taught us: never underestimate the amount of money, time, and effort someone will expend to thwart a security system. It's always better to assume the worst. Assume your adversaries are better than they are. Assume science and technology will soon be able to do things they cannot yet. Give yourself a margin for error. Give yourself more security than you need today. When the unexpected happens, you'll be glad you did.*" **Bruce Schneier**

Cryptanalytic techniques are very important and continuously used to evaluate the security of cryptographic primitives. Such techniques are used to derive some bounds with respect to the security of a cryptographic primitive under different settings and scenarios such as,

1. The goal of the attacker. For example, an adversary may want to break the full scheme by recovering the secret key K or he may want to recover the plaintexts of some ciphertexts.

2. Allocated resources, like money and computing power

3. The amount of information available to the adversary, like the number of available plaintext-ciphertext pairs (P, C) and the type of access that an adversary has to the system, for instance wiretapping.

In this chapter, we provide a brief introduction to the area of cryptanalysis. Initially, we discuss a classification of the attacks based on the motivation of the attacker, as well

on the amount and type of available data. Then, we discuss *brute-force* cryptanalysis, which is a form of a generic attack.

In the rest of this chapter, we discuss more advanced cryptanalytic techniques. Informally, cryptanalysis can be seen as two separate worlds; the world of *approximation cryptanalysis* and the world of *exact algebraic cryptanalysis*. In approximation cryptanalysis, we search for special events occurring perhaps due to unexpected structure (like propagation of specific input-output differences), which occur with relatively high probability and we exploit them in order to derive some key bits or any other information. On the other hand, in exact algebraic cryptanalysis, we exploit the algebraic description of the primitive, like in algebraic attacks which we describe in Chapter 4. In this section, we are mainly interested in cryptanalysis of block ciphers and we discuss LC and DC, which are the most powerful cryptanalytic techniques. In addition, we discuss some enhancements of DC, which is a major theme for this thesis.

3.1 Classification of Attacks

So far we have investigated the role of a cryptographer. Answering the following questions, we get important insights regarding the capabilities of a cryptanalyst.

1. *What are the aims of an attacker*?
2. *What are the attacker's available resources*?
3. *What type of access does an attacker have to the system*?

The most extreme scenario is that the cryptanalyst would like to recover all bits of the secret key. Knudsen classifies the aim of an attacker in the following way [85]:

1. **Total break**: The attacker recovers the secret key k.
2. **Global deduction**: The attacker finds an algorithm A, which is equivalent to either $Enc_K(.)$ or $Dec_K(.)$, without the need to know the secret key k (can be seen as an equivalent key).
3. **Local deduction**: Generating the message (or ciphertext) corresponding to a previously unseen ciphertext (or message).

4. **Information deduction**: Obtaining some non-negligible quantity or relation regarding the key bits, which is true in general.

5. **Distinguishing Algorithm**: Distinguishing a given cipher from a random permutation. This notion is applied only to block ciphers. In most cases, we construct a distinguisher for distinguishing a reduced-round version of the cipher from a random permutation. In practice, it is frequently possible to extend a distinguishing algorithm to a key recovery attack against the full block cipher, however it is not trivial at all.

Regarding the access to the system, Schneier classifies the attacks in the following way [110]:

1. **Ciphertext-only attack**: The adversary is assumed to have access to observe the ciphertext(s) corresponding to several plaintext(s), which are encrypted under the same cryptosystem and the same secret key. For such an attack to succeed, we need to have enough redundancy in the plaintext.

2. **Known-plaintext attack**: The adversary has access to see some plaintexts together with the corresponding ciphertexts.

3. **Chosen-plaintext attack**: A type of known-plaintext attack but with the flexibility that the adversary has the ability to encrypt plaintexts of his choice.

4. **Adaptive-chosen-plaintext attack**: A special case of chosen-plaintext attack, where an adversary can additionally modify his choice based on the results of previous encryptions.

5. **Chosen-ciphertext attack**: The attacker can select different ciphertexts to be decrypted.

A cipher which is vulnerable to ciphertext-only attacks is very weak. Known-plaintext and chosen-plaintext attacks are realistic scenarios, as in most applications it is feasible to make conclusions about the structure of a plaintext for a given ciphertext. For example, encrypted source code and executable code are very vulnerable, since some words such as $\{define, struct, if, else, for\}$ are repeated many times. However, some countermeasures can be employed in this direction, such that the encryption of

plaintexts which have some specific structure is avoided. A common countermeasure is to compress before we encrypt. The compressed file has a uniform distribution of characters. Moreover, compression results in shorter plaintexts and thus the time needed to encrypt and decrypt is reduced and in addition the reduced redundancy in the plaintext can potentially hinder certain cryptanalytic attacks.

Another scenario is that the adversary can obtain the encryptions of a plaintext under two or more keys which are linked via some mathematical relationship known to the attacker. For example the encryption of the plaintext can be done under two keys which share the same 20 last bits. These attacks are called *related-key* attacks [15].

The success of a cryptanalytic attack is measured based on the required resources needed. In addition to the amount and type of available data, the following resources are fundamental to be considered,

1. **TIME**: The time needed to launch an attack. Potentially it is the most important requirement considered.

2. **MEMORY**: The amount of memory or storage needed during an attack. An example is the meet-in-the-middle attack, where intermediate states resulting from the encryption of plaintexts for a limited number of rounds need to be stored.

3. **DATA**: We have already discussed that the amount and type of data is very important.

The data types and available resources are very important to be considered, when evaluating the security of a cryptographic primitive in isolation. However, if the primitive is used in real-life applications, then the implementation of it must be studied also. For example, a primitive may seem to be secure against such mathematical attacks but some flaws in its implementation might make it insecure. There are attacks which exploit the leakage of physical information or power consumption or time required during encryption, such as *side-channel* and *timing* attacks [110].

3.2 Brute-force Attack

Brute-force attack or *exhaustive key search*, is the most general attack that can be applied to any encryption algorithm. It is applied in cases where no weaknesses of the scheme are known and they do not exploit any structural properties. It is often used in a known-plaintext or ciphertext-only attack scenario.

A brute-force attack works in the following way. Assume that a plaintext P is encrypted to a ciphertext C by an encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $k \in \mathcal{K}$ is the secret key. Then, an adversary given a (P, C) pair, encrypts P under all possible keys $k_i \in \mathcal{K}$, until he finds a key k_j such that $\mathcal{E}_{k_j}(P) = C$. If the block size equals to key size, then on average one pair is enough. In the worst case, the adversary needs to exhaustively search the entire key space. Thus, the length of the key determines the practical feasibility of performing a brute-force attack.

Exhaustive key search is always possible in theory but computationally infeasible in practice. A cipher with key length $|K|$ bits can be broken in a worst-case time $2^{|K|}$ encryptions and on average in time of $2^{|K|-1}$. In symmetric cryptography $|K| = 128$ is considered good enough for commercial applications, while for military-grade security we need at least $|K| = 256$.

However, as we have already mentioned in the introductory section this level of security can never be reached due to inevitable key bits loss. A MITM of time complexity $2^{|K|}(1 - \epsilon)$, where $\epsilon > 0$, is always achievable for practical ciphers [74].

Lastly, brute force attack is possible in a ciphertext-only setting. This could be successful provided that there is enough redundancy in the plaintext (e.g. the plaintext is in English alphabet). Under this assumption, the attacker decrypts the ciphertext under all key guesses, until a meaningful plaintext is obtained. We assume that there exists an efficiently computable procedure which allows us to decide if the plaintext is valid.

3.3 Linear Cryptanalysis (LC)

Linear cryptanalysis (LC) is a *known plaintext attack* on block ciphers discovered by Corfdir and Gilbert [9]. Firstly, it was successfully applied to the FEAL cipher in 1993 by Matsui [93] and a year later to DES [92]. Despite the fact that the published attack on DES was not very practical (as it requires 2^{43} known plaintexts), it led to the development of further academic cryptanalysis. Since then, LC is considered as one of the most important cryptanalytic techniques and evidence of security against it is expected in all new block cipher designs, as well as stream ciphers.

LC is based on finding affine approximations to the action of a cipher, which hold with relatively high probability. The main idea is to find affine relations between bits of the plaintext, the ciphertext and the key (which hold with relatively high probability) such that we can deduce information about some key bits.

Let $(K, \mathcal{E}, \mathcal{D})$ be an encryption scheme, which encrypts a given plaintext $P \in \{0,1\}^n$ to a ciphertext $C \in \{0,1\}^n$, using a key $k \in \{0,1\}^K$. The main aim is to find binary vectors (or linear masks) $\alpha, \beta \in \{0,1\}^n$ and $\gamma \in \{0,1\}^K$, such that linear approximations of the form

$$\alpha.P \oplus \beta.C = \gamma.K, \tag{3.1}$$

hold with relatively high probability. The RHS of Equation 3.1 depends only on key bits. Assuming that the RHS is constant and equal to some value b, the main aim is to find linear masks (α, β) for which the LHS equals to some value b more frequently. Thus, we aim to maximize the following probability,

$$p_{\alpha,\beta} = Pr(\alpha.P \oplus \beta.C = b).$$

In an ideal cipher, we expect such relation to hold with probability $\frac{1}{2}$ for both $b = 0$ and $b = 1$. Thus, we are interested in finding such approximations which hold with probability $p_{\alpha,\beta}$, such that the correlation or imbalance of linear approximation given by,

$$c_{\alpha,\beta} = 2p_{\alpha,\beta} - 1 \in [-1, 1],$$

is either relatively high or low. Then, for all values of the key bits on the RHS of Equation 3.1, we count how many times the approximation is true over all known (P, C) pairs. By construction, we expect the choice of the set of key bits which result to the greatest absolute difference from half of the number of pairs which satisfy this equation to be the most likely set.

The most important task involved in LC is the discovery of such linear masks. All linear functions of a cipher can be trivially described by linear equations. The tricky part is to find linear approximations for the non-linear components, which hold with a comparatively good probability. S-boxes and modular additions are examples of non-linear functions.

In general, in case of an iterated cipher we work in the following way,

1. Consider the linear equations that describe linear components which hold with probability 1.

2. For any non-linear component of the form $S : \{0,1\}^n \rightarrow \{0,1\}^m$, find vectors $\alpha \in \{0,1\}^n, \beta \in \{0,1\}^m$, such that for all inputs X and corresponding output Y, the equation $\alpha X \oplus \beta Y = b$ holds with sufficiently high probability.

3. Combine them to get a linear approximation of the round function.

4. Concatenate to get linear approximations over more rounds.

For each non-linear component, we compute its corresponding *linear approximation table*, which is a table that contains all possibly input and output masks (α, β), together with the the associated bias given by,

$$\epsilon = Pr(\alpha.X \oplus \beta.Y = 0) - \tfrac{1}{2}.$$

After obtaining linear approximations of the round function, we join them to get a linear approximation for several rounds. If this linear approximation covers s rounds of the cipher, it is a called a s-round characteristic (cf. Definition 15).

Definition 15. *(Linear Characteristic)*

An s-round linear characteristic of probability $p = \frac{1}{2}+\epsilon$, is an $(s+1)$-tuple $(\alpha_0, ..., \alpha_s)$, where α_i is the input mask to the i-th round and $\epsilon > 0$ the bias.

3.3.1 Key Recovery using LC

In this section, we discuss how we obtain some information regarding the secret key using LC. Assuming a linear equation of the form $\alpha.P \oplus \beta.C = \gamma.K$, of bias $|\epsilon|$, we can recover one key bit information in the following way (cf. Algorithm 3).

Algorithm 3 Recovering one-bit of secret key via LC

1. Encrypt N distinct plaintexts $P_1, ..., P_N$ and get the corresponding ciphertexts $C_1, .., C_N$
2. Initialize counters T_0, T_1
3. Increment T_i when $\alpha.P_j \oplus \beta.C_j = i$
4. Identify $T = \max_{i=0,1} T_i$ (if the bias is positive)
5. Return i

The above attack returns one bit of key information. The RHS of the above equation involves only key bits, while the LHS involves only plaintext and ciphertext bits. Thus, we expect the RHS of the above equation to be equal to a fixed value (either 0 or 1) for approximately $p.N$ pairs. By construction, we expect the counter with the maximum value to suggest the correct equation involving only key bits.

However, using partial decryption of the last round, we can obtain more than just one linear equation. We apply this to a r-round iterated block cipher. Suppose we have constructed a $(r-1)$-round linear characteristic. This corresponds to a linear approximation of the form,

$$\alpha_0.m \oplus \alpha_{r-1}x_{r-1} = \alpha_0.k_0 \oplus ... \oplus \alpha_{r-1}k_{r-1}, \tag{3.2}$$

where x_{r-1} is the input to the last round and of bias $|\epsilon|$. Algorithm 4 can recover more key bits in the following way.

Algorithm 4 Linear Cryptanalysis using partial decryption of last round

1. Create a list of key guesses k^G for the last round key (or the partial last round key) and consider counters T_0^G and T_1^G
2. For all N known (P, C) pairs and $k \in k^G$, compute the partial decryption to the last round of x_{r-1} under k.
3. If $\alpha_0.m \oplus \alpha_{r-1}.x_{r-1} = i$ increase T_i^G, for $i = 0, 1$
4. Identify the correct T_j^G with the maximum value and take k^G as the correct round key. (The maximum is taken when ϵ is positive, otherwise the minimum)

For a wrong key guess we expect the counter to be around $\frac{N}{2}$ because in a random permutation we expect such linear equation to hold with probability exactly $\frac{1}{2}$. This is called the *wrong-key randomization* assumption [86].

Remark 6. *It is very important to note that in Algorithm 4 we do not need to guess the full last round key. We only need to guess a few key bits which allow us to obtain by decrypting the ciphertext, all bits of the last round input appearing in the linear characteristic.*

To determine the complexity of a linear cryptanalytic attack, it is sufficient to estimate the probability of a linear characteristic. If such a linear characteristic holds with bias ϵ, then approximately $\frac{c}{\epsilon^2}$ of known (P, C) pairs are needed, where c is a small constant [86]. A linear characteristic is the XOR of several random variables and thus the corresponding probability can be computed by the Piling-Up Lemma (cf. Lemma 2).

Lemma 2. *(Piling-up Lemma)*
Let X_i for $1 \leq i \leq n$ be n independent binary random variables such that

$$P(X_i = 0) = p_i \ \forall 1 \leq i \leq n$$

Then,

$$P(X_1 \oplus ... \oplus X_n = 0) = \frac{1}{2} + 2^{n-1} \prod_{1 \leq i \leq n} (p_i - \frac{1}{2}).$$

Proof. The proof can be found in standard textbooks about symmetric cryptanalysis such as [86]. More details are out of the scope of this thesis. □

The Piling-up lemma can be used under the assumption that the random variables or the linear approximations are independent. This brings the notion of Markov cipher with respect to LC but this is out of our scope and more details are found in [86].

3.4 Differential Cryptanalysis (DC)

Differential cryptanalysis is a *chosen plaintext attack*. Its discovery was attributed to Eli Biham and Adi Shamir in the later 1980's, who were the first to publish a differential attack against DES [16, 17].

However, around 1994, Don Coppersmith as a member of the original IBM DES team, confirmed that the technique of DC was known to IBM, as early as 1974. He mentioned that, one of the criteria used in the design of DES was the resistance against this attack [28]. IBM decided after discussion with NSA to keep confidential the design criteria used for DES, as such a publication would reveal this technique which could be used against many other ciphers and cryptographic primitives.

In DC, the main task is to study the propagation of differences from round to round inside the cipher, and find specific differences (cf. Definition 16), which propagate with relatively high probability. Such pairs of input-output differences can be used to recover some bits of the secret key. Generally, it exposes the non-uniform distribution of the output differences given one or several input differences.

Definition 16. *(Difference)*
Let $(G, \otimes)$ be a finite abelian group with respect to the operator $\otimes$ and $x_1, x_2 \in G$ be two elements of the group. The difference between x_1, x_2 w.r.t operator $\otimes$ is defined as

$$\Delta x = \Delta(x_1, x_2) = x_1 \otimes x_2^{-1},$$

where x_2^{-1} is the inverse of x_2 with respect to $\otimes$.

Usually, the operator $\otimes$ considered is the exclusive-or operator $\oplus$. This is due to the fact that in many ciphers the key application in the round function is a simple XOR. An element x with respect to XOR is self-inverse. Thus, $(x_1 \oplus k) \oplus (x_2 \oplus k) = x_1 \oplus x_2 = \Delta x$, which means that the key addition preserves the difference. Generally, the selection of the operator depends on the way the round subkeys are introduced. DC is easier to be performed in cases where the output difference depends only on the input and output differences and does not depend on the round subkeys. Thus, the main aim is to choose a notion of difference that allows us to ignore the action of the key for at least part of the analysis.

In the rest of this section we analyze how DC can be used to obtain key bits for iterated block ciphers. Given an iterated cipher, we study the propagation of differences

though different number of rounds. Then, individual differences are joined to form a differential characteristic for a larger number of rounds (cf. Definition 17). Constructing the best possible differential characteristic by combining several one-round characteristics is a non-trivial optimization task.

Definition 17. *(Differential Characteristic, Figure 3.1)*

An s-round characteristic is an $(s+1)$-tuple of differences $(\alpha_0, ..., \alpha_s)$, where α_i is the anticipated difference Δc^i after i rounds of encryption. The initial input difference $\Delta m = \Delta c^0$, is denoted by α_0.

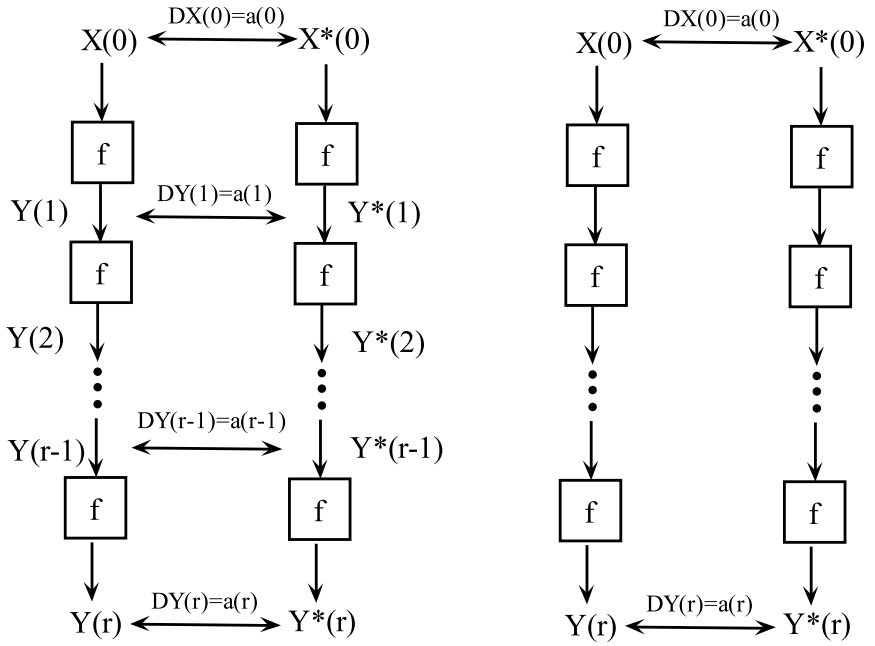

Figure 3.1: A Differential Characteristic and a Differential over r rounds - The diagram on the left illustrates the propagation of differences $\alpha(0), \alpha(1), ..., \alpha(r)$ through different rounds, which is called differential characteristic. The diagram on the right illustrates a differential, where only input-output differences are considered, while middle differences are ignored.

3.4.1 Computing the Probability of a Differential Characteristic

In differential attacks, the first task is to find series of input and output differences over several rounds, which appear with relatively high probability. For each pair of input-output difference, we need to determine the probability of propagation for each

round individually. For the linear components, we can predict the propagation of the difference with probability one. However, in non-linear components, such as S-boxes, a probabilistic analysis is needed. This is a very similar task as in LC.

We call an S-box *active* if its input difference is non-zero, while we call it *inactive* or *passive* if the input difference is zero. Clearly, a zero input difference gives a zero output difference for an inactive S-box with probability 1. In the substitution layer of a cipher, S-boxes are applied in parallel to different chunks of data and thus they are independent and hence corresponding probabilities are multiplied. Another, non-trivial optimization task for the attacker is to carefully select which S-boxes are taken as active in each round such that the overall differential characteristic has a relatively good probability of propagation. Many ad-hoc heuristics can be discovered by studying the structure of the round function of a cipher which might suggest how to select which differences are interesting to study.

In the rest of this section, we study how we can compute the probability of a differential characteristic for an iterated block cipher. Given an $(s+1)$-round characteristic $(\alpha_0 ... \alpha_s)$, the probability of propagation over all keys and plaintexts is given by,

$$P_{\mathcal{K},\mathcal{P}}(\Delta c^s = \alpha_s, \Delta c^{s-1} = \alpha_{s-1}, ..., \Delta c^1 = \alpha_1 | \Delta c^0 = \alpha_0)$$

Thus, we need to compute it on average over all keys and plaintexts. This is difficult to determine for a certain class of ciphers since the model of computation does depend on the cipher. For example, in some ciphers it may be infeasible to compute it since it may depends on the key and the plaintext in a very complex way, while on some other ciphers this dependency may not be so complex and thus we may be able to enumerate all possible differential attacks. However, for the class of Markov ciphers (cf. Definition 18), this can be computed by simply computing transitional probabilities for each round and then multiplying them. The notion of a Markov cipher simplifies a lot the model of computation.

Definition 18. *(Markov cipher, [88])*

An iterated cipher round function $Y = f(X, Z)$ is a Markov cipher, if there is a group operation $\otimes$ for defining differences such that, for all choices of α $(\alpha \neq e)$ and $(\beta \neq e)$,

$$P(\Delta Y = \beta | \Delta X = \alpha, X = \gamma),$$

is independent of γ when the subkey Z is uniformly random.

For a Markov cipher, the probability of an one-round characteristic taken over all keys and plaintexts is independent of the plaintext and thus it can be computed over the key space only.

Moreover, for an iterated r-round Markov cipher with r independent round keys chosen uniformly at random, the sequence of differences $\Delta c^0, .., \Delta c^r$ forms a homogeneous Markov chain (Definition 19).

Definition 19. *(Markov Chain, [88])*
A sequence of r random variables $X_0, X_1, .., X_r$, is called a Markov chain if

$$P(X_{i+1} = \beta_{i+1} | X_i = \beta_i, .., X_0 = \beta_0) = P(X_{i+1} = \beta_{i+1} | X_i = \beta_i),$$

for all $0 \leq i \leq r$.

Such a Markov chain is *homogeneous*, if

$$P(X_{i+1} = \beta | X_i = \alpha) = P(X_i = \beta | X_{i-1} = \alpha)$$

Thus, the probability of an s-round characteristic for a Markov cipher with independent round keys can be computed as follows [86].

$$P(\Delta c^s = \alpha_s, ..., \Delta c^1 = \alpha_1 | \Delta c^0 = \alpha_0) = \prod_{1 \leq i \leq s} P(\Delta c^i = \alpha_i, \Delta c^{i-1} = \alpha_{i-1}) \tag{3.3}$$

3.4.2 Differentials vs. Differential Characteristics

An adversary does not have so much freedom to determine if the input difference follows a given differential characteristic in each step. However, he can choose the input difference and may be able to check the corresponding output difference after s rounds. An s-round characteristic is constructed by concatenating s one-round differentials.

In practice, it is very time consuming to find a really good differential characteristic over a sufficient number of rounds. The collection of all s-round characteristics with input α_0 and output difference α_s is called a differential (Definition 20).

Definition 20. *(Differential, [86])*
An s-round differential is a pair of differences (α_0, α_s), also denoted as $\alpha_0 \rightarrow \alpha_s$, where α_0 is the chosen input difference and α_s the expected output difference Δc^s

Given an s-round differential (α_0, α_s), the probability of such differential on average over key space and all plaintexts is given by,

$$P_{K,\mathcal{P}}(\Delta c^s = \alpha_s | \Delta c^0 = \alpha_0) = \Sigma_{\alpha_1} ... \Sigma_{a_{s-1}} P_{K,\mathcal{P}}(\Delta c^s = \alpha_s, ..., \Delta c^1 = \alpha_1 | \Delta c^0 = \alpha_0)$$

In an attack the key is fixed and only the plaintext can vary. Thus, in practice we may need to compute it over a fixed key which is not known to the attacker. Computing the following probability is enough to mount many cryptographic attacks:

$$P_{\mathcal{P}}(\Delta c^i = \alpha_i | \Delta c^{i-1} = \alpha_0, K = k).$$

However, the key is unknown, and thus we cannot compute this probability unless we consider the assumption of stochastic equivalence (Assumption 1).

Assumption 1. *(Hypothesis of stochastic equivalence)*
Consider an r-round iterated cipher, then for all highly probable differentials, $s \leq r$, (α, β),

$$P_{\mathcal{P}}(\Delta c^s = \beta | \Delta c^0 = \alpha, K = k) = P_{\mathcal{P},\mathcal{K}}(\Delta c^s = \beta | \Delta c^0 = \alpha),$$

holds for a substantial fraction of the key space $\mathcal{K}$.

Remark 7. *In practice, we expect that the probability of a differential does not really depend on the key or the plaintext. It is assumed that such probability is close to the expected probability on average over all plaintexts and key.*

3.4.3 Key Recovery Attacks

In this section, we describe how to derive some key bits using differential attacks. Consider a differential (α, β) over $r-1$ rounds, which holds with probability p for a r-round iterated block cipher. By partial decryption of the last round, we can recover some bits of the last key faster than brute-force.

Firstly, we encrypt N pairs of plaintexts (P, P'), such that $\Delta P = \alpha$ and get the corresponding ciphertext pairs (C, C'). Given these pairs, we guess some bits of the last round key and we partially decrypt the last round. Then, we check if the the difference after $r-1$ rounds is obtained. If this difference is obtained we say that (P, P') suggests a candidate k_G. We expect approximately $p.N$ pairs to result in pairs with difference β in the round before the last round. Such pairs are called right pairs (cf. Definition 21).

Definition 21. *(Right Pair)*
A pair (P, P') *with* $\Delta P = \alpha$ *and associated ciphertexts* (C, C') *is called a* right pair *with respect to the* $(r-1)$*-round differential* (α, β) *if* $\Delta c^{r-1} = \beta$*. Otherwise, it is called a* wrong pair.

In order to launch a successful differential attack, we need at least one right pair. First, in an attack we want to identify a right pair. However, we have also wrong pairs that do not follow our constructed characteristic and this is referred to as noise, while right pairs are the signal. Thus, wrong pairs should be filtered in a very early stage of our attack if it is possible. Often wrong pairs can be eliminated by considering the associated ciphertexts. This process is called *filtering*. Note that there are no general rules how to perform the filtering step and it depends on the cipher.

Algorithm 5 describes an attack on an r-round iterated block cipher using an $(r-1)$ differential characteristic. This attack can be used to obtain some bits of the last round key using a differential characteristic of the form $(\alpha_0, ..., \alpha_{r-1})$, which holds with probability p.

Algorithm 5 Differential Attack against r-round iterated cipher

1. Let T_j a counter for (parts of) possible last round key guesses k_j
2. For $i = 1, ..N$ do
(a) Choose P_i at random and compute $P_i' = P_i \oplus \alpha_0$. Obtain the corresponding ciphertexts (C_i, C_i').
(b) Use filtering. If (P_i, P_i') is a wrong pair, discard it and continue with the next iteration. Otherwise do the following.
(c) For each key guess k_j, partly decrypt the last round and get $(c_i^{(r-1)}, c_i^{'(r-1)})$. Increase T_j, if $c_i^{(r-1)} \oplus c_i^{'(r-1)} = \alpha_{r-1}$
3. Find l, such that $T_l = \max_i(T_i)$
4. Return k_l as the guess for the correct key

Remark 8. *In general, we can have some early rejection on some key bits by computing certain bits of the ciphertexts. This is due to the structure of ciphers and their key schedule, which make certain bits depend only on a few key bits.*

In most cases after applying the method of DC, one pair might suggest several key candidates $\{k_1, k_2, ..., k_l\}$. On the contrary, a wrong pair is expected to suggest a set of candidates which do not include the correct key. The attack is successful, if the

correct key value is suggested significantly more often than the other candidates. This is expected for a differential of probability p if approximately $\frac{c}{p}$ plaintexts are selected uniformly at random, where c a small constant depending on the cipher [86]. In the rest of this section, we discuss some advanced forms of DC, such as truncated and impossible differentials.

3.5 Advanced Differential Cryptanalysis

3.5.1 Truncated Differentials

Truncated Differential Cryptanalysis is a generalization of differential cryptanalysis developed by Lars Knudsen [84]. Usually, in DC we study the propagation of single differences between two plaintexts, while in truncated DC we consider differences that are partially determined (i.e we are interested only in some parts of the difference). This technique has been successfully applied to many block ciphers such as SAFER, IDEA, Skipjack, Twofish and many others. We define the truncation $TRUNC(a)$ of a n-bit string a as in Definition 22.

Definition 22. *(Truncation, [84])*
Let $a = a_0a_1...a_{n-1}$ be an n-bit string, then its truncation is the n-bit string b given by $b_0b_1..b_{n-1} = TRUNC(a_0a_1..a_{n-1})$, where either $b_i = a_i$ or $b_i = *$, for all $0 \leq i \leq n-1$ and $*$ is an unknown value

The notion of truncated differentials (cf. Definition 23) extends naturally to differences.

Definition 23. *(Truncated Differentials, [84])*
Let (α, β) be an i-round differential, then if α' and β' are truncations of α and β respectively, then (α', β') is an i-round truncated differential.

Remark 9. *Note that we need to exclude the zero difference from our set.*

Example 1. *The truncated differential on 8 bytes of the form $0000000000 * 00000$ (in hexadecimal representation), where $* = x_1x_2x_3x_4$, is a set of differences of size $16 - 1$ (excluding the zero difference).*

Given an s-round characteristic $\Delta_0 \to \Delta_1 \to ... \to \Delta_s$, then $\Delta'_0 \to \Delta'_1 \to ... \to \Delta'_s$ is a truncated characteristic, if $\Delta'_i = TRUNC(\Delta_i)$ for $0 \leq i \leq s$. A truncated characteristic predicts only part of the difference in a pair of texts after each round of encryption.

A truncated differential is a collection of truncated characteristics. Truncated differentials proved to be a very useful cryptanalytic tool against many block ciphers which at first glance seem secure against basic differential cryptanalysis.

In the rest of this subsection we apply the technique of truncated differential cryptanalysis to the 5-round CIPHERFOUR block cipher shown in Figure 3.2. CIPHERFOUR is an SPN with S-box given by $\{64C5072E1F3d8A9B\}$ and permutation function given by $\{04812159132610143711 15\}$,

We study Knudsen's example as described in [86] against truncated DC. We observe that an input bits difference 0010 to the second S-box leads to the following output differences after one round

$$(0000000000100000) \rightarrow \begin{cases} (0000000000100000) \\ (0000000000000010) \\ (0010000000100000) \\ (0010000000000010) \end{cases}$$

Note that * means 1-bit difference, including zero difference. However, we need to exclude the case where the difference is identical to zero. Then, we study the propagation of each of these differences after an additional round. We obtain the following the differences,

$$(0000000000100000) \rightarrow (00 * 0000000 * 000 * 0)$$

$$(0000000000000010) \rightarrow (000 * 0000000 * 000*)$$

$$(0010000000100000) \rightarrow (*0 * 00000 * 0 * 0 * 0 * 0)$$

$$(0010000000000010) \rightarrow (*00 * 0000 * 00 * *00*)$$

All these differences can be denoted by a truncated differential of the form $(*0 * *0000 * 0 * * * 0 * *)$. If we continue for another round we get

$$(*0 * *0000 * 0 * * * 0 * *) \rightarrow (*0 * * * 0 * * * 0 * * * 0 * *)$$

which results in the 3-round truncated differential,

$$(0000000000100000) \rightarrow (*0 * * * 0 * * * 0 * * * 0 * *),$$

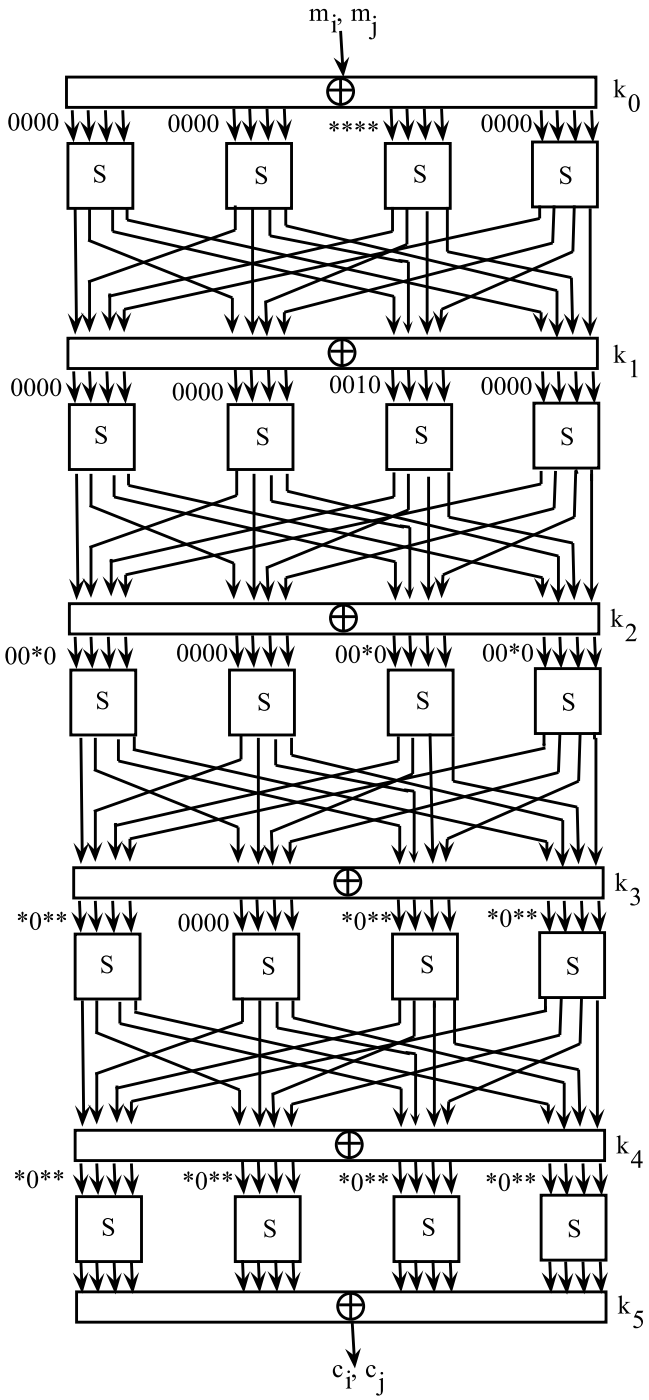

Figure 3.2: DC: Truncated Differentials - The truncated differential attack on five-round CIPHERFOUR proposed by Lars Knudsen, where * implies difference on 4 bits [86]

which holds with probability 1.

Using this truncated differential we can recover bits from the secret keys of the first and last round of the 5-round CIPHERFOUR. If the difference 0000000000100000 occurs at the beginning of the second round then we have the truncated differential

$$(0000000000100000) \rightarrow (*0***0***0***0**)$$

with probability 1.

We launch a differential attack using this truncated differential using a structure of 16 distinct messages of the form $m_i = t_0||t_1||i||t_2$, where t_0, t_1, t_2 are randomly selected constants and $i = 0, .., 15$ [86]. Any two different messages lead to difference of the following form,

$$t_0 \oplus t_0||t_1 \oplus t_1||i \oplus j||t_2 \oplus t_2 = 00000000 * * * *0000.$$

For any value of the four key bits affecting the third S-box, we can find 16 pairs of messages (m_i, m_j) such that the difference in the outputs from the first round S-boxes is (0000000000100000). Once this difference is obtained, then the truncated differential property holds with probability 1. Thus, assuming the four bits of k_0 are correct, our message pairs will give a four-round truncated differential property with probability 1. Then, the partially-encrypted pairs that result will have a difference 0 in the positions as indicated in Figure 3.2 and this property can be used to identify the key bits of k_5.

For the purposes of our attack we suppose that for each possible value of k_0 we use s message pairs (m_i, m_j) that yield the required difference after one round. Then, these pairs can be used to recover four bits of k_0 and all key bits of k_5. Thus, the attack has the potential to find in total 20 bits of the secret key. It is possible that the pairs obtained from the structure of 16 messages are insufficient to uniquely determine the secret key and if this happens one can simply generate another structure by using different constant values for t_0, t_1 and/or t_2. More details can be found in [86].

3.5.2 Impossible Differentials

Impossible differential cryptanalysis is a form of cryptanalysis that studies propagation of differentials which are impossible, i.e they have probability 0 to propagate at some intermediate state of the cipher. Such events can be used to deduce some key bits and such an example is discussed below.

Consider a 6-round Feistel network (Figure 3.3), where the round function f is a bijection for any fixed key. Suppose we have a pair of input messages $X = x_L||x_R$ and $X' = x'_L||x'_R$, such that $x_R = x'_R$ and $x_L \oplus x'_L = a$, where $a \neq 0$.

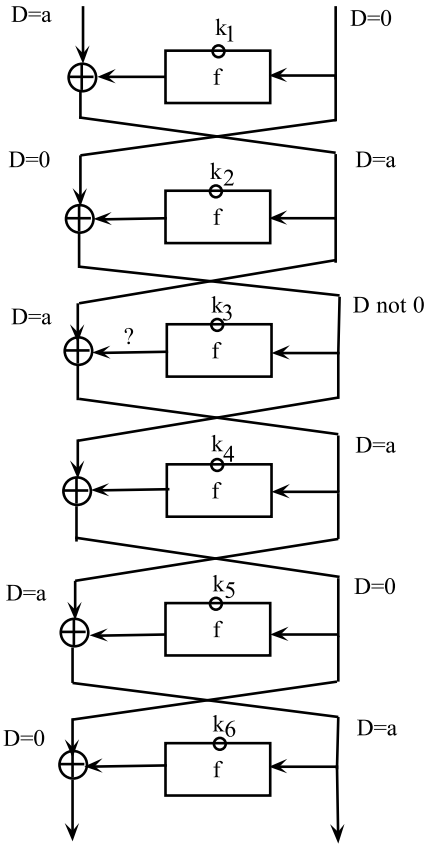

Figure 3.3: DC: Impossible Differentials - A 6-round Feistel network without final swap where the round function f is bijective for any fixed key. The five-round differential $(a, 0) \rightarrow (0, a)$ $(a \neq 0)$ can never occur due to a contradiction in the third round

Assume that the output difference after 5-rounds is $0||a$. Then, by construction the input difference at both the second and fourth round are of the form $0||a$. The input difference in round 2 is $0||a$ so the input difference in round 3 is $a||b$ where $b \neq 0$. The input difference in round 4 is $0||a$ and since f is bijective it implies the input difference in round 3 is $a||0$, which gives a contradiction in round 3.

This implies that $a||0 \rightarrow 0||a$ is an impossible 5-round differential as it holds with probability 0. This fact can be exploited in order to derive the last round key k_6 in the following way [86].

1. Consider plaintexts of the form $m_i = m_i^L||m_i^R$, where $m_i^L = i$ for $i = 0, ..., 2^{\frac{n}{2}-1}$ and $m_i^R = c$, where c is a random and fixed $\frac{n}{2}$-bit value.
2. Let $c_i = c_i^L||c_i^R$ be the corresponding encryptions
3. Find all pairs (m_i, m_j) such that $m_i^L \oplus m_j^L = c_i^R \oplus c_j^R = \alpha_{i,j}$, where $\alpha_{i,j} \neq 0$. ($2^{\frac{n}{2}-1}$ such pairs are expected)
4. Decrypt all pairs for one round for all possible values of k_6.
5. If the input difference in round 6 is $0||\alpha_{i,j}$, then reject this value of k_6

If the obtained input difference in round 6 is $0||\alpha_{i,j}$, then this value of k_6 must be wrong as it gives an impossible differential for five rounds. For wrong values of k_6, we expect this with probability $2^{-\frac{n}{2}}$ for each pair. Thus, for $2^{\frac{n}{2}-1}$ pairs around half of the candidates for k_6 are discarded. For each different value of c that is tried, the key space is halved and eventually only a few values for k_6 will remain. More details regarding this attack are out of the scope of this thesis and can be found in [86].

4

Software Algebraic Cryptanalysis

[Breaking a good cipher should require] as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type" (Claude Elwood Shannon, 1949, [113])

In general, the security of a given block cipher will grow exponentially with the number of rounds and so does the number of required (P, C) pairs which are needed in a linear or differential attack. However, in most cases only a few (P, C) pairs are available and as a result many standard cryptanalytic techniques are not expected to succeed. Thus, different techniques are needed when only very limited information is available. This is the point exactly at which algebraic cryptanalysis comes to our attention.

Claude Shannon once advised that the security of a cipher should be related to the difficulty of solving the underlying system of equations which describes it [113]. This is the core concept behind algebraic attacks. An algebraic attack can be a form of *known plaintext attack* and consists of the following two basic steps:

1. **Modelling Step**: Describe the cipher as a multivariate system of polynomial equations over any well-chosen field in terms of the secret key K, the plaintext P and the ciphertext C.

   $\{f_1(K, P, C) = 0, f_2(K, P, C) = 0, ..., f_r(K, P, C) = 0 \Longleftrightarrow E(K, P) = C\}$

2. **Solving Stage**: Solve the underlying multivariate system of equations and obtain the secret key. In order to reduce the complexity of solving the system, we

substitute to the system one or several (P, C) pairs. The more pairs, the more equations we obtain involving the key bits.

One exploits the fact that many cryptographic primitives can be described by a sparse multivariate non-linear system of equations over a binary field or any other algebraic system. Efficient implementations of ultra-lightweight ciphers suggest low gate equivalent count which means that the system of equations describing it is very *sparse*. Generally, solving a system of multivariate non-linear Boolean equations is an NP-complete problem [68]. Even solving Multivariate Quadratic (MQ) systems is proved to be NP-hard [45].

Given an iterated block cipher, we start by obtaining algebraic representations of each individual component for each round. Then we join them together to get an algebraic description for the whole system. Linear components are trivial to describe. Examples are bit-wise permutation layers and key additions. Describing non-linear components is the hardest task as there is no straightforward and efficient method to obtain such representations.

With respect to the solving stage, several techniques have been developed. A first attempt was to use techniques from algebraic geometry and especially Gröbner bases algorithms [66]. However, most of the times they do not lead to solutions in practice due to their extremely high memory requirements.

Thereinafter, some heuristic techniques were developed as the method of *linearization*, where all the non-linear terms are replaced by an independent variable and the resulting linear system can be solved using Gaussian elimination in some cases [112]. However, it requires that there are enough linearly independent equations and the initial system is highly *over-determined*. Then, the XL algorithm was developed to make the system over-determined, by addition of new equations to the current system. Such new equations are generated by multiplying the initial set of equations with all monomials up to certain degree [45].

Additionally, a very powerful algorithm named *ElimLin* was developed by Nicolas Courtois for solving such systems [55], specifically in the context of algebraic attacks of ciphers. Elimlin exploits the linear equations in the linear span of all polynomial equations. It is a mixture of linear algebra and substitution and we refer to it explicitly in the next section.

4.1 ElimLin Algorithm and Multiplicative Complexity (MC)

The ElimLin algorithm was designed by Nicolas T. Courtois and appeared firstly in [39]. It is an iterated 2-step simple algorithm used for solving multivariate systems of polynomial equations over any field and it is based on Gaussian Elimination on sparse systems.

Given a system of multivariate polynomial equations $\{f_1, f_2, ..., f_m\} \in K[x_1, x_2, .., x_n]$, ElimLin firstly computes affine equations of the form $l = \sum_{0 \leq i \leq n} a_i x_i$, $a_i \in K$, which lie in the linear span $L =< \sum_{1 \leq j \leq m} b_j f_j(x) >$. This is done by performing Gaussian Elimination on the matrix which represents the m polynomial equations and includes the coefficients of all monomials. Then, given an affine equation l, we re-write a variable x_i that appears in this equation as a linear combination (+ an affine constant a_0) of the other variables involved and then eliminate x_i from all other equations in the initial system. This results in a system with less variables. We keep iterating the same procedure until the system does not output any new linear equations, hoping that at this point the system is simple enough to be solved. ElimLin is formally described in Algorithm 6 given below.

Algorithm 6 ElimLin Algorithm

Input: $S^0 = \{f_1, f_2, ..., f_m\} \in \mathbb{F}_2[x_1, x_2, .., x_n]$
Output: An updated system of equations S^T and a system of linear equations S_L
1. Set $S_L \leftarrow \emptyset$ and $S^T \leftarrow S^0$ and $k \leftarrow 1$
2. **Repeat**
For some ordering of equations and monomials perform $Gauss(S^T)$ to eliminate non-linear monomials
Set $S_{L'} \leftarrow$ Linear Equations from $Gauss(S^T)$
Set $S^T \leftarrow Gauss(S^T) \backslash S_{L'}$
Let $l \in S_{L'}$, l non-trivial (if unsolvable then *terminate*)
Let x_{i_k} a monomial in l
Substitute x_{i_k} in S^T and $S_{L'}$ using l
Insert l in S_L
$k \leftarrow k + 1$

Suppose that the total number of monomials is N and the rank of the matrix representation of the system has rank r. Then, using the echelon form of the matrix

after Gaussian Elimination we observe that a sufficient condition for a linear equation to exist is that

$$r > N - 1 - n, \tag{4.1}$$

and this is called the *sufficient rank condition*. It is trivial to see that all the linear equations are found in the first iteration of the algorithm, since the rank of the matrix is invariant and all monomials are assumed to be linearly independent.

ElimLin algorithm and Multiplicative Complexity, which is a major topic in this thesis, seems to be closely related. The basic idea in ElimLin is to exploit hidden linear equations that lie in the linear span of the polynomials. The reduction methods with respect to MC aim to minimize the number of AND gates used in the system by allowing unlimited number of XOR gates. Thus, by considering the straight line program of a circuit is like aiming to compute the same circuit using as few as possible steps which include multiplications. Our method of reducing MC (which we study extensively in a later section) aims at linearizing the system as much as possible.

In the next sections of this chapter, we describe an automated software methodology based on SAT solvers used for algebraic cryptanalysis. The method we describe consists of two main steps; the *conversion* and the *solving* steps where a dedicated software is used in the second step.

4.2 Representations and Conversion Techniques

As we have mentioned above, an algebraic attack is a 2-step methodology. Firstly, we need to obtain the representation of the cipher over a well chosen field and then we try to solve this system to obtain the secret key.

The most tricky part is the modelling-step. We need to find a compact representation over a field which is convenient for expressing the majority of the components of the cipher. For example, in AES we have a mixture of operations from $\mathbb{F}_2$ and $\mathbb{F}_{2^8}$.

The following definitions describe several representations that can be used to describe any given system of Boolean equations over $\mathbb{F}_2$. For example, CNF (Definition 24) is a very important form of logical description of a set of a equations. We can translate any given set of algebraic equations into this language and then try to solve this problem using state-of-art algorithms such as SAT solvers [71].

Definition 24. *(Conjunctive Normal Form)*

A Boolean function f is said to be in conjunctive normal form, if it is a conjunction of clauses, where each clause is a disjunction of literals, i.e f can be expressed in the form

$$\wedge_{I\subseteq M}(\vee_{i\in I}x_i), M = \{1,..,n\} \tag{4.2}$$

Definition 25. *(Disjunctive Normal Form)*

A Boolean function f is said to be in disjunctive normal form, if it is a disjunction of clauses, where each clause is a conjunction of literals, i.e f can be expressed in the form

$$\vee_{I\subseteq M}(\wedge_{i\in I}x_i), M = \{1,..,n\} \tag{4.3}$$

Definition 26. *(Algebraic Normal Form)*

A Boolean function f is said to be in algebraic normal form, if it is an XOR of clauses, where each clause is a conjunction of literals,

$$\oplus_{I\subseteq M}a_I \wedge (\bigwedge_{i\in I}x_i), M = \{1,..,n\}, a_I \in \{T,F\} \tag{4.4}$$

The existence of the ANF of each Boolean function is explained according to the *Universal Mapping Theorem.*

Theorem 3. *(Universal Mapping Theorem)*

For any q, any map from a finite field $GF(q)$ to itself can be written as a polynomial system of equations over $GF(q)$.

Theorem 3 gives us an explicit method for algebraic encoding of components of ciphers and therefore also for the whole ciphers.

4.3 SAT-solvers in Cryptanalysis

The problems of solving multivariate systems of quadratic equations over the Galois Field with two elements $\mathbb{F}_2$ and that of finding a satisfying assignment for a logical expression are both NP-hard. The fact that all NP-complete problems are polynomially equivalent, combined with the extensive research on algorithms for solving CNF-SAT problems motivated the authors in [11, 10] to study algorithms for converting the first problem into the second one.

Thus, if any SAT solver software could be able to determine a logic assignment for a CNF-SAT problem this could be translated into a solution to the corresponding system of equations in polynomial time. This leads to an automated algorithmic procedure for solving such systems of equations. The methodology can be summarized as follows (more details in [11, 10]),

- We assume that the given cryptographic primitive can be described as a logical circuit
- Write it as a degree 2, 3 or 4 system,

$$f_1(x_1, ..., x_n) = y_1, f_2(x_1, ..., x_n) = y_2, ..., f_m(x_1, ..., x_n) = y_m$$

  It is possible to see that this can always be done if we add variables in the previous step.

- Convert it to a CNF-SAT using polynomial time conversion techniques
- Solve this problem using SAT solver software [71]

In general, CNF expressions describe instances of SAT problems, thus we need to obtain the CNF of this multivariate system of quadratic equations. This conversion proceeds by three major steps. Firstly, some preprocessing needs to be done in order to make the system amenable to this conversion, then the system of polynomials will be converted to a linear system and a set of CNF clauses that describe each monomial equivalent to a variable in the system. Lastly, the linear system is converted to a set of clauses. This methodology is described precisely in [11].

Regarding SAT solvers, we use them as black-box algorithms and all technicalities behind them are not in the scope of this thesis. SAT solvers are sophisticated algorithms which allow us to solve NP-complete problems by mapping these problems to CNF. At first glance, this seems to be inefficient since a lot of structure of the original problem is lost during conversion but this is offset by the performance of SAT solvers. In this thesis, we make extensive use of SAT solvers such as MiniSAT and CryptoMiniSAT [96].

Courtois was the first to successfully break ciphers using SAT solvers in the area of symmetric cryptanalysis. He suggested two main approaches in SAT cryptanalysis for

breaking a cipher with a SAT solver [40]. The first approach is called the SAT method and according to this method an attacker after finding a proper algebraic description of the cipher, he fixes X bits of the secret key. Provided that the assumption on X bits is correct, it takes T time to output SAT and a solution. Thus, we need in total $2^X.T$ time. This is very similar to *guess-then-determine* strategy and leads to a combinatorial-optimization problem as follows,

Optimization problem: Find the number of bits X such that $f(X) = 2^X.T$ is optimal, where T is the time taken to output SAT.

There are two main approaches in cryptanalysis of blocks ciphers based on SAT solvers. We have the SAT Method and the UNSAT Method [40].

1. **The SAT Method:** Guess X bits and run a SAT solver. Then, if the assumption on the X bits is correct, it takes time T to output SAT and a solution. Abort all other computations at time T. The total time complexity is about $2^X.T$

2. **The UNSAT Method:** Guess X bits and run a SAT solver which, if the assumption is incorrect, it finds a contradiction in time T with large probability $1 - P$, say 99%.

Then, with a small probability $P > 0$, we can guess more key bits and thus either find additional contradictions or find solution. If P is small enough, then the complexity of these additional steps can be less than the $2^X.T$ spent in the initial UNSAT step. However, if P is not as small as we would like to be, therefore we may have a mix of both methods, where the final complexity will be a sum of two terms none of which can be neglected. This is called a **Mixed UNSAT/SAT Attack** [40].

4.4 Combined Algebraic-Differential Attacks

Albrecht *et al.* suggested a simple way to expand the multivariate system of equations that describe a given block cipher by adding linear equations between inputs and outputs arising from highly likely differentials for some reduced number of rounds [3, 4, 5].

Given two pairs $(P', C'), (P'', C'')$ under the same key K, we can write two systems of equations F', F'', which share the key and key schedule variables but do not share

most of the state variables. We can expand this system by considering a differential characteristic $\Delta = (\delta_0, \delta_1, ..., \delta_r)$ for a number of rounds, where $\delta_{i-1} \rightarrow \delta_i$ is a one-round differential, which propagates with probability p_i.

Then, considering the j-th bit of the inputs denoted by $X'_{i,j}, X''_{i,j}$ to the i-th round and assuming that $X'_{i+1,j}, X''_{i+1,j}$ are the corresponding output bits we have the additional expressions,

$$X'_{i,j} \oplus X''_{i,j} = \Delta X_{i,j},$$

where $\Delta X_{i,j}$ are known values predicted by the characteristic and valid with some non-negligible probability for bits of active S-boxes. In addition, we have the following relations for the non-active S-boxes given by,

$$X'_{i,j} \oplus X''_{i,j} = 0 \tag{4.5}$$

Denote as S_L the of set all these linear constraints, we can expand the initial system $F' \cup F''$ which holds with probability 1 to a larger system $F' \cup F'' \cup S_L$ which holds with probability $p = \prod_i p_i$, assuming statistical independence of one-round differences. If we attempt to solve this system for $\frac{1}{p}$ pairs, we expect at least one consistent solution, which should yield the encryption key. The augmented system is expected to be easier to solve as we add only new linear constraints without adding new variables. In practice, one can reduce the number of variables since all the $X''_{i,j}$ variables can be replaced either by $X'_{i,j}$ or $X'_{i+1,j}$

In a very similar way, we can extend this combined algebraic-differential attack, if truncated differentials exist in the block cipher which hold with sufficiently high probability. Truncated differentials are expected to hold with much higher probability than single differences and thus the time complexity of recovering the secret key is expected to be significantly reduced.

Let X'_i, X''_i the inputs to round i of the block cipher and X'_{i+1}, X''_{i+1} the corresponding outputs. Given truncated differentials, we know that if $X'_i \oplus X''_i \in \Delta X$, then $X'_{i+1} \oplus X''_{i+1} \in \Delta Y$, where $\Delta X, \Delta Y$ are sets of differences. Then, we have the following sets of equations that could be added to initial system of equations

$$X'_{i,j} \oplus X''_{i,j} = I_{(\Delta X)_j}, X'_{i+1,j} \oplus X''_{i+1,j} = I_{(\Delta Y)_j}, \tag{4.6}$$

where $I_{(\Delta X)_j}$ is either 1 or 0. Otherwise no equation is added if the difference is free (i.e $\Delta_j = *$). Equations are added only for inactive bits.

The existence of such differentials implies that the solution of the system $F' \cup F''$ lies in a reduced version of this system where input-output bits are constrained by some linear relations given by the mask of differentials. Intuitively, we believe that if such truncated differentials exist, then the performance of many ad-hoc heuristic algorithms for solving systems of equations like ElimLin is improved. It is also possible to see that SAT based algorithms, as well as Gröbner Basis based techniques are improved by adding such simple linear equations. We have managed to break 10 (out of 44) rounds of the SIMON cipher with 128-bit key and 64-bit block [12] using this technique which combines truncated differentials with algebraic techniques. It is very interesting that we broke 10 rounds without the need to guess any key bits in advance. More details can be found in [53] and are out of the scope of this thesis.

5

Boolean Functions and Measures of Non-Linearity

In cryptography, the are two main philosophies for the design of block ciphers. One philosophy, which is the most widely adopted, studies the cryptographic properties of the underlying non-linear components, such as S-boxes, in isolation. The other philosophy is based on the diffusion criteria of the round function and the security of the full block cipher relies on the adoption of many number of rounds [106].

The first philosophy is widely used and as a result of this, very strict criteria that these S-boxes need to satisfy were formulated and followed by the designers starting from around 1994 [99]. More precisely with *Multipermutations* [118] and the *Wide Trail Strategy* (WTS) [106], we can have strict lower bounds on how many S-boxes in the next round are affected by a difference on few bits.

Following the first design paradigm, the appropriate selection for the S-boxes is a very important task. It is believed that the cipher can be secure against classical attacks by a good choice of a S-box which fulfills the desired criteria. The designers must choose the S-boxes that provide little advantage to the attacker and design the diffusive components of the cipher in such a way such that the number of S-boxes that a cryptanalyst would need to deal with in some attack is maximal. Thus, the designers need to solve the underlying optimization problem and find the optimal S-box which offers the desired level of security.

Current cryptographic standards or prominent block ciphers make use of functions which are simple to compute and efficient with respect to some criteria such as software

and hardware implementation. At the same time, they must be sufficiently "distant" from linear functions. Intuitively, if the function has a good approximation by a linear function, then it is going to be vulnerable against a range of cryptographic attacks. This is exactly the point where the notion of *non-linearity* comes.

Non-linearity is one of the most fundamental requirements for a given cryptographic function and many attempts have been made in order to define an appropriate metric for it. In a very recent paper, Boyar *et al* studied different well-known measures for the non-linearity of Boolean functions such as: *non-linearity*, *algebraic degree*, *annihilator immunity* and *multiplicative complexity* [20]. All these measures do reflect the nonlinearity of the function but are incomparable, in a sense that a function may offer a high level of non-linearity with respect to one metric but very low with respect to another.

In an earlier paper, Courtois *et al* have already introduced the notion of MC and showed (heuristically) that functions with low MC are less resistant against algebraic attacks [42, 44]. The notion of MC is a core point for this thesis and it is proved to be another measure of non-linearity for cryptographic primitives. In modern cryptographic implementations, S-boxes are the main components used to enhance the non-linearity of the primitive. S-boxes of small dimensions (such as 4-bit to 4-bit) are preferred in lightweight implementations.

In this section, we make an introduction to the area of Boolean functions and introduce all known measures of nonlinearity. In particular, we discuss the four notions of non-linearity as recently introduced by Boyar *et al* and highlight the relation between MC and non-linearity [20, 42, 44]. At the end of this section, we discuss a complete classification of all optimal 4-bit to 4-bit S-boxes with respect to cryptography-related metrics as proposed by Leander *et al* [89].

5.1 Boolean Functions

A Boolean function is a function of the form $f : \mathbb{F}_2^n \to \mathbb{F}_2$, where n is called the *arity* of the function. We denote by $B_n = \{f | f : \mathbb{F}_2^n \to \mathbb{F}_2\}$ the set of all n-dimensional Boolean functions. Clearly, B_n is a vector space over $\mathbb{F}_2$ of dimension 2^n.

The cardinality of the set is 2^{2^n} and its size increases doubly exponentially as n increases. For example, if $n = 6$, then we have that the size is 2^{64}, which is feasible to study. However, for $n = 8$, we have that the size equals to 2^{256}, which is approximately

the total number of the atoms in the universe. Any Boolean function can be represented (uniquely) via the following representation forms (cf. Definition 27).

Definition 27. *(Boolean Functions Representations)*

A given Boolean function of the form $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *can be represented by:*

1) ***Truth Table:*** *The output vector of the form* $(f(0), f(1), .., f(2^n - 1)) \in \mathbb{F}_2^n$

2) ***Polarity Truth Table:*** *The truth table of the function* $\widehat{f}(x) = 1 - 2f(x)$ *or* $(-1)^{f(x)}$*. A vector of* 2^n *elements, where each element belongs to the set* $\{-1, 1\}$

3) ***Algebraic Normal Form (ANF):*** *The ANF of an n-dimensional Boolean function* f *is written as:*

$f(x) = a_0 \oplus a_1 x_1 \oplus ... \oplus a_n x_n \oplus a_{12} x_1 x_2 \oplus ... \oplus a_{(n-1)n} x_{n-1} x_n \oplus ... \oplus a_{12..n} x_1 ... x_n$ *,*
where $a_i \in \{0, 1\}$

Using Theorem 4, the computation of the ANF of a given Boolean function can be computed using the Möbius transform as defined below.

Theorem 4. *Each Boolean function* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *has a unique polynomial representation in*

$$\mathbb{F}_2[x_1, ..., x_n]/(x_1^2 + x_1, ..., x_n^2 + x_n)$$

According to Theorem 4, any function f can be written as,

$$f(x) = \sum_{u \in \mathbb{F}_2^n} \lambda_u . x^u,$$

where $x^u = x_1^{u_1} x_2^{u_2} ... x_n^{u_n}$. It is easy to see that the coefficients are given by

$$\lambda_u = \sum_{x \leq u} f(x), \tag{5.1}$$

where $x \leq u$ means $u_i = 0 \Rightarrow x_i = 0$. This is also known as the Möbius transform. From the ANF of a function we can compute its algebraic degree $deg(f)$ (cf. Definition 28).

Definition 28. *The algebraic degree* $deg(f)$ *of a function is defined as the number of variables in the highest order term with non-zero coefficient in the ANF representation.*

Intuitively, the higher the algebraic degree, the higher the resistance of the function against algebraic attacks. The optimal value for algebraic degree is n and some examples are presented below.

Example 2. $S_1(x_1, ..., x_4) = x_1x_3x_4 + x_1 + x_2 + 1$ *has algebraic degree 3*

Example 3. $S_2(x_1, ..., x_4) = x_1x_2x_3x_4 + x_1x_3 + x_4 + 1$ *has algebraic degree 4*

Example 4. $MAJ(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$ *has algebraic degree 2*

Example 5. $CTC2_{(1,0,0)}(x_1, x_2, x_3) = 1 + x_0 + x_1 + x_0x_1 + x_0x_2 + x_1x_2$ *has algebraic degree 2*

As the set of all n-dimensional Boolean functions is a vector space, it is reasonable to define a *measure* on this space. A measure on each element, is given by its *Hamming weight*,

$$HW(f) = \sum_{x=0}^{2^n-1} f(x),$$

where the sum is taken over the integers.

The distance between two Boolean function f, g is given by the Hamming distance

$$HD(f, g) = \sum_{x=0}^{2^n-1} (f(x) \oplus g(x)).$$

The degree of similarity between two functions f, g is given by the *correlation coefficient* $cc(f, g)$ (cf. Definition 29), which is a real number between -1 and 1.

Definition 29. *(Correlation Coefficient)*
Let f, g two Boolean functions of the form $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$. Then the correlation coefficient is given by

$$cc(f, g) = 2P(f(x) = g(x)) - 1 = 1 - \frac{HD(f,g)}{2^{n-1}}$$

Definition 29 implies that if the evaluations of the two functions differ in all points $x \in \mathbb{F}_2^n$, then $cc(f, g) = -1$, i.e $f(x) = 1 - g(x)$. If they are identical, we have $cc(f, g) = 1$. The correlation coefficient is a method of evaluating how one function contributes towards the approximation of another. In general, two functions are correlated if and only if $cc(f, g) \neq 0$, otherwise they are said to be uncorrelated.

5.1.1 Affine Boolean Functions

The subspace of B_n which contains all n-dimensional functions of algebraic degree 1 (i.e of all *affine functions*) is called the *affine subspace*. An affine function is any function of the form

$$l_\alpha = \alpha_0 \oplus \alpha_1 x_1 \oplus \alpha_2 x_2 ... \oplus \alpha_{n-1} x_{n-1},$$

where $\alpha = (\alpha_{n-1}, ..., \alpha_0) \in \{0,1\}^n$. If $\alpha_0 = 0$ then the function is *linear*.

Under the action of affine transformations, two functions f, g are said to belong to the same affine equivalence class if and only if there exist $c \in \mathbb{F}_2$, $a, b \in \mathbb{F}_2^n$ and a square non-singular $n-$dimensional matrix A such that $g(x) = f(Ax \oplus a) \oplus b \cdot x \oplus c$, where $\cdot$ denotes the inner product.

Many cryptanalytic attacks exploit the existence of linear relations between inputs and outputs which hold with high probability. Hence, it is necessary to define a measure of linearity of a given function. (cf. Definition 30)

Definition 30. *(Linearity of a function)*
The linearity of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined by

$$L(f) = \max_{a \in \mathbb{F}_{2^n}} |f^W(a)|,$$

where $f^W(a)$ is the Walsh Coefficient of f at a given by

$$\sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)+a.x}$$

The maximal value of $L(f)$ is 2^n and is obtained iff f is an affine or linear function. As we will see in a later section, the notion of nonlinearity of a Boolean function is extensible to the notion of nonlinearity of multi-dimensional Boolean functions (or S-boxes). Table 5.1 presents the linearity of several Boolean functions.

Table 5.1: The linearity of several Boolean functions

Boolean Function f	$L(f)$
$f(x_1, x_2) = x_1 + x_2$	4
$f(x_1, x_2) = x_1x_2 + x_1$	2
$MAJ(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$	4
CTC2 $S_{(1,0,0)}(x_1, x_2, x_3) = 1 + x_0 + x_1 + x_0x_1 + x_0x_2 + x_1x_2$	4

5.2 Measures of Nonlinearity

The non-linearity of a Boolean function is a notion which is invariant under affine transformations and can be computed using the Walsh coefficient and it is defined as the Hamming distance to the closest affine function. The linearity defined in Definition 30 gives us an insight on the linear sections of the given Boolean function and it is very important to be as low as possible since the more the linearity the higher the chances to make a conclusion about the input given the output by using especially linear algebra tools.

In addition, the algebraic degree is another notion for measuring the security of a primitive against algebraic attacks. It was shown that algebraic degree can give insights regarding the non-linearity of the function by comparing the two notions with respect to other measures such as algebraic thickness and non-normality [27]. However, this is out of the scope of this thesis.

Recently, Boyar *et al* discussed in their paper four measures of non-linearity [20]. In addition to the usual notion of non-linearity and of algebraic degree, which we have already discussed, they study the important and well-known notions of *annihilator immunity* (cf. Definition 31) and *multiplicative complexity* (cf. Definition 32).

Definition 31. *(Annihilator Immunity)*

Let f a Boolean function on n inputs. Then, the annihilator immunity is given by,

$$AI(f) = \min_g(deg(g)),$$

such that either $fg = 0$ or $(f + 1)g = 0$. The function g is called an annihilator.

The annihilator immunity is closely related to the algebraic degree of the function. Attempts have been made to find bounds for this measure [31].

Theorem 5. *(Courtois-Meier 2003, [31])*

Let f a Boolean function on n inputs. Then the annihilator immunity satisfies the following inequality,

$$0 \leq AI(f) \leq \lceil \tfrac{n}{2} \rceil$$

Functions that attain such bounds are known [58].

Definition 32. *(Multiplicative Complexity)*

Multiplicative Complexity of a function is the smallest number of AND gates necessary and sufficient to compute the function using a circuit over the basis (XOR, AND, NOT).

These notions worth studying since they are related to the non-linearity of a cryptographic primitive and they all need to be taken into account individually for the evaluation of the security of a given cryptographic primitive. This is due to the fact that all these measures are *incomparable*, as shown recently by Boyar *et al* [20]. This implies that for each pair of measures μ_1, μ_2, there exist functions f_1, f_2 with $\mu_1(f_1) < \mu_1(f_2)$ but $\mu_2(f_2) < \mu_2(f_1)$. The incomparability is explained in the same paper by providing explicit examples based on constructions using the majority function and symmetric polynomials [20].

The most important result of the same paper, which has direct connection with this thesis, is the connection between MC and non-linearity. Boyar *et al* proved that if a function has low non-linearity, this gives a bound on its MC. Conversely, they prove that if a Boolean function on n inputs has $MC < \frac{n}{2}$, then it has non-linearity at most $2^{n-1} - 2^{n-MC-1}$ [20]. Thus, MC can be seen as a new notion of measuring the non-linearity of a given cryptographic primitive and as an extension it can be seen as a measure of security against known attacks. In addition, they prove that MC is related to the one-wayness of a hash function and they provide a lower bound on the MC of *collision-free* functions. Thus, the notion of MC is a new fundamental notion which needs to be considered in the security analysis of a given primitive and Chapter 6 is devoted to introducing methods for optimizing small components used in block ciphers such as S-boxes, as well as arbitrary algebraic computations over the general non-commutative rings.

5.3 Substitution Boxes

Substitution Boxes are vectorial Boolean functions of the form $S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$. We denote as $B_{n,m} = \{S|S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m\}$, the set of all Boolean functions which take as input n bits and output m bits. The cardinality of this set is $(2)^{m2^n}$, which makes the exhaustive study of all Boolean function a highly intractable problem.

Each S-box can be written as $S = (S_1(x), S_2(x), .., S_m(x))$, where $S_i(x)$ are the coordinate (or component) functions of S (cf. Definition 33). The dimension of an S-box is strongly related to the properties of the S-box.

- If $n < m$ and $S(x_1) \neq S(x_2)$ $\forall x_1, x_2 \in \mathbb{F}_2^m$, then the S-box is *injective*
- If $n > m$ and $\forall y \in \mathbb{F}_2^m$ $\exists x \in \mathbb{F}_2^n$, then we say that the S-box is *surjective*
- If $n = m$ and both assumptions above hold then we say that S-box is *bijective*

Definition 33. *A component function $S_\beta(x)$ ($\beta \in \mathbb{F}_2^m, \beta \neq 0$) for an S-box S is given by*

$$S_\beta(x) = \sum_{1 \leq i \leq m} \beta_i \cdot S_i(x)$$

5.3.1 Cryptographic Properties of S-boxes

S-boxes play a fundamental role for the security of nearly all modern block ciphers. In many block ciphers, S-boxes are the only non-linear part of the block cipher. Each S-box can be implemented as a look-up table with 2^n words of m bits each. In general there are two ways of generating cryptographically good S-boxes,

1. **Select a large S-box randomly from the set of all S-boxes**. This results in inefficient software and especially hardware implementations due to its enormous circuit representation. Another way to implement S-box efficiently is by the so called bit-slice methods.
2. **Generate S-boxes of small dimensions which fulfil certain properties**, like high resistance against linear and differential cryptanalysis and each of its components $S_\beta(x)$ has high algebraic degree and it has an efficient implementation.

However, it is not trivial to find such S-boxes which are optimal with respect to these properties since the systematic exploration of the space of all permutations from n bits to m bits is a computationally hard problem. We can restrict the number of S-boxes to be studied by considering their equivalence classes under a given relation. Following this idea, the study is restricted to the study of the properties of the representative of each class.

Another very important notion in cryptography is that of *balancedness* (cf. Definition 34).

Definition 34. *(Balance) An N variable boolean function $f(x)$ is said to be balanced if $|\{x|f(x) = 0\}| = |\{x|f(x) = 1\}|$*

Clearly for a balanced function we have $HW(f) = 2^{n-1}$ and a function which is not balanced is called unbalanced.

Another very important measure which quantifies resistance of a given S-box against LC is the notion of *Linearity* (cf. Definition 35) which is a simple extension of non-linearity of Boolean functions.

Definition 35. *(Linearity)*
Given an S-box $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ the linearity denoted by

$$L(S) = \max_{a \in \mathbb{F}^n, b \in \mathbb{F}^m - \{0\}} |S_b^W(a)|,$$

is the maximum linearity value obtained among all the linear combinations of the output components. $S_b^W(a)$ is the Walsh coefficient of Boolean function S_b as defined in Definition 30.

Another important concept is the notion of *Avalanche effect* introduced firstly by Feistel [67]. Avalanche effect is the spreading of the input bits throughout a process so that will result in a likely uniform distribution in the number of 0 and 1 bits in the output. Firstly, we need to define a measure of the effect on the output of a function when there is a change in the input in a given direction a (cf. Definition 36).

Definition 36. *(Discrete Derivative) The discrete derivative of a function f in the direction of the vector $a \in \mathbb{F}_2^n$ and it is given by*

$$d_a f(x) = f(x) \oplus f(x \oplus a)$$

A function f is said to satisfy the Strict Avalanche Criterion (SAC) if it satisfies the following property in Definition 37

Definition 37. *(Strict Avalanche Criterion (SAC))*
Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Then, f satisfies strict avalanche criterion if $\forall a \in \mathbb{F}_2^n$ such that $HW(a) = 1$, $d_a f(x)$ is balanced.

If a function f satisfies the strict avalanche criterion it means that the derivative of the function over all inputs and in all directions a with Hamming weight 1, is balanced. Thus, the output difference for all single bit changes to the input must be uniformly distributed.

S-boxes are the main non-linear components in many prominent block ciphers, thus properties such as strict avalanche criterion for S-boxes individually are well studied. For an S-box S, denote by $\Delta_{S,a}(b) := \{x \in \mathbb{F}_2^n | d_a S(x) = b\}$. By computing $|\Delta_{S,a}(b)|$, we find the number of input pairs $(x, x + a)$ that result in an output difference b. A strong S-box against differential cryptanalysis means that the distribution of output differences is approximately uniformly distributed.

Thus, the value $Diff(S) := max_{a \neq 0, b \in \mathbb{F}^n} |\Delta_{S,a}(b)|$, which is related to the maximal probability that any fixed non-zero input difference causes any fixed output difference after applying the S-box, is a good indicator of how strong the S-box is against differential cryptanalysis. The resistance of a given S-box can be formally measured using the branch number as defined in Definition 38. The larger the branch number is, the stronger the avalanche effect should be.

Definition 38. *(Branch number)*
Given an S-box $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ its branch number is denoted by

$$BN(S) = min_{a,b \neq a \in \mathbb{F}_2^n}(HW(a \oplus b) + HW(S(a) \oplus S(b)))$$

5.3.2 Classification of Small (4×4) S-boxes

Designers of block ciphers usually use small S-boxes with good cryptographic properties due to their efficient implementation in hardware. Common choices of (n, m) for an S-box are $n = m = 8$ and $n = m = 4$. Table 5.2 presents the dimensions of input and output for S-boxes used in several prominent block ciphers.

Generally, 4-bit to 4-bit S-boxes are used in many important ciphers such as GOST and PRESENT. An essential requirement is that the S-boxes selected are optimal

Table 5.2: S-boxes used in several prominent block ciphers

Cipher	n	m
AES	8	8
DES	6	4
GOST	4	4
PRESENT	4	4
CTC2	3	3

against linear and differential cryptanalysis and in addition have some desirable algebraic properties. Definition 39 lists the properties that a 4-bit to 4-bit S-box needs to fulfill in order to be cryptographically good. Proof of this can be found in [89].

Definition 39. *Let $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be a bijective S-box. Then S is an optimal S-box w.r.t linear and differential cryptanalysis if:*

1. $L(S) = 8$
2. $Diff(S) = 4$

$Diff(S)$ is a measure of resistance against DC as it is related to the maximal probability that any fixed non-zero input difference causes any fixed output difference after applying the S-box. By optimality we mean that S has as low as possible linearity $L(S)$ and $Diff(S)$. However, finding optimal 4-bit to 4-bit S-boxes is a very hard computational task, as there are roughly $16! \simeq 2^{44}$ such permutations. Considering the action of a set of transformations on the set of S-boxes which do not alter these properties partitions the space into distinct equivalence classes. Thus, the study of each representative is sufficient.

We say that two S-boxes S, S' are affine equivalent and denoted by $S \sim S'$, if there exist bijective linear mappings A, B and constants $a, b \in \mathbb{F}_2^4$, such that

$$S'(x) = B(S(A(x) + a)) + b$$

Under this equivalence relation the study can be reduced to the study of only $11! \simeq 2^{25}$ permutations which are then tested if they satisfy the definition of optimality [89]. This reduction can be done by observing that for a bijection $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$, we can always find bases $B, B' \in \mathbb{F}_2^n$ such that $S(B) = B'$. Using this equivalence relation,

Leander discovered that there exist only 16 such equivalence classes, (cf. Table A.1, Appendix A.1).

The notion of Affine Equivalence (AE) (cf. Figure 5.1) does not alter any of the cryptographic properties and especially the nonlinearity of the cipher and especially for the n-bit to n-bit optimal S-boxes this is proved in Theorem 6.

Theorem 6. *(Affine Equivalence (AE), [89])*

Let $A, B \in GL(n, \mathbb{F}_2)$ be two invertible $n \times n$ matrices and $a, b, \in \mathbb{F}_2^n$. Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be an optimal S-box. Then the S-box S' given by

$$S'(x) = B(S(A(x) + a)) + b,$$

is an optimal S-box as well.

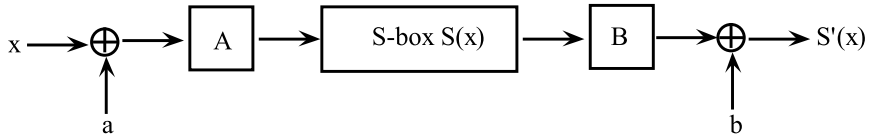

Figure 5.1: Affine Equivalence Relation between two S-boxes - A, B boxes denote multiplication by matrices in $GL(4, \mathbb{F}_2)$.

However, for members of an AE class there is no guarantee that a single bit difference in the input will not cause a single bit output difference. This implies that only a single S-box is activated in the next round of a substitution permutation network and it is equivalent to branch number equal to 2. Considering equivalence classes with respect to linear equivalence does not ensure that avalanche effect is the same for all elements within the same equivalence class. A classification of 4-bit to 4-bit S-boxes with respect to permutation equivalence (PE) (Definition 40) was studied by Saarinen [109].

Definition 40. *(Permutation-XOR Equivalence , [109])*

Let P_i, P_o be two bit permutation matrices and c_i, c_o two vectors. Then the S-box S' given by $S'(x) = P_o S(P_i(x \oplus c_i)) \oplus c_o$ belongs to the permutation-xor equivalence class as S.

In [109] there is a more general study of S-boxes with respect the Permutation-XOR equivalence which guarantees also the same avalanche effect within the same classes. After exhaustive search on the set of 16! S-boxes using some short-cuts, 142,090,700

different PE classes were found. Members of a PE class (not permutation-xor) are usually equivalent with respect to their bit-slicing implementations meaning that they have equivalent circuit complexity. This is very interesting from a cryptanalytic point of view as it implies that multiplicative complexity is still the same.

Remark 10. *In the next chapter, we study the bit-slice implementation of the PRESENT S-box. Using a SAT-solver based technique, we obtain the best current bit-slice implementation.*

In Chapter 6, we study methods for computing the MC of a given cryptographic primitive and we successfully compute it for small enough circuits such as 4-bit to 4-bit S-boxes. The method we employ is based on conversion and SAT solver techniques. However, we also use classification of 4-bit to 4-bit S-boxes for speeding up our algorithm.

Part II

On the Matrix Multiplication and Circuit Complexity

6

Optimizations of Algebraic Computations

Finding shortcuts to the problems of Matrix Multiplication (MM) and Circuit Complexity (CC) are among the most important NP-hard problems in the area of computational complexity [117, 29, 26, 25, 44]. Many attempts have been made for reducing the number of multiplications needed for computing the product of two matrices, as well as for constructing optimal circuits under almost any meaningful metric, such as gate count, depth, energy consumption etc.

Despite the fact that a lot of research has been carried out in these areas, both problems are still intractable and existing techniques are subject to more improvements. A small improvement in the existing algorithms for solving such problems has the potential to result in huge improvements in many other algorithms and applications.

Most of the existing algorithms for solving such problems, are based on well-chosen *ad-hoc heuristics*. However, there is no formal proof that such techniques yield optimal solutions and many of these heuristics have *exponential* time complexity. As a result of this, they can only be applied to optimize very small instances of the problems. Surprisingly, obtaining optimal representations for smaller instances may lead to significant improvements in general. For example, this happens in MM problems, where the multiplication of two matrices can be decomposed into multiplication of smaller sub-matrices.

Recently, Boyar and Peralta presented new techniques for obtaining more efficient implementations of arbitrary digital circuits with repsect to Boolean complexity based

on the notion of Multiplicative Complexity (MC) [23]. They suggest a 2-step method, where initially the circuit is optimized with respect to its number of non-linear (AND) gates and then it is optimized with respect to linear (XOR) gates independently. The key idea behind this heuristic is that if we optimize a circuit with respect to AND gates, then we just need to work to optimize the circuit with respect to linear gates locally and this technique will sometimes give very good results. The problem of optimization the total number of additions needed for computing linear forms is well-studied and there are several good techniques that can be applied [21, 69].

There is no formal proof (and it seems unlikely to be true in general) that optimization with respect to AND gates yields circuits with optimal Boolean complexity. However, this heuristic technique yields circuits implementations which are sometimes the best known compared to available implementations [23, 69].

In this chapter, we propose a fully automated 2-step methodology based on SAT-solvers for optimizing digital circuits with respect to their MC, but also with respect to other metrics meaningful in cryptography and cryptanalysis. In our method, firstly we describe algebraically the problem by a multivariate system of equations or by another formal method of encoding. Then, we convert it to a CNF-SAT problem using the Courtois-Bard-Jefferson tool [11] and finally we attempt to solve this problem using a SAT solver [42].

Importantly, using this methodology we have been able to obtain circuit representations of optimal circuit complexity for sufficiently small circuits, with respect to several metrics we have tried. Using our technique we have been able to optimize the S-boxes of widely used ciphers such as GOST [63] and PRESENT [18] with respect to MC and prove that it cannot be reduced anymore. Optimization with respect to MC is very important since it is one of the major countermeasures against Side Channel Attacks (SCA) on smart cards [104]. We also optimize the 3-bit to 3-bit S-box of CTC cipher [32] with respect to several metrics in order to illustrate that our technique is capable of discovering optimal circuit representations under different metrics.

Moreover, using Brent's equations [26] as a method of formal encoding for the MM problem and following similar methodology as in circuit's optimization, we have been able to obtain new *bilinear* non-commutative algorithms of optimal bilinear complexity for the computation of the product matrix of two sufficiently small matrices. Such

solutions to smaller instances can be used recursively to obtain solutions for the more general case using the *Divide-and-Conquer* paradigm [61].

Remark 11. *It is worth noticing that SAT solvers are subject almost every year to improvements since there is a very active academic community in this area. We can hope this will give us the opportunity to discover much more interesting bilinear non-commutative algorithms especially for the MM problem in the future and manage to discover optimal circuit representations for bigger circuits.*

6.1 Divide-and-Conquer Paradigm

In computer algebra, a main strategy for designing more efficient algorithms for solving computationally hard problems is by solving smaller (fixed-size) problems and then applying the solution obtained recursively. This paradigm is known as *Divide-and-Conquer* [61]. Many attempts of solving the MM problem have been made based on this strategy [26]. The paradigm of Divide-and-Conquer works as follows [61]:

1. Partition the given problem into subproblems, that are themselves smaller instances of the same type of problem
2. Solve these smaller instances
3. Combine the solutions obtained for the smaller instances
4. Apply recursively at the lower levels

The complexity of solving the main problem is determined by the complexity of completing the above steps. As an introductory example, we will study how this technique can be applied to the *Integer Multiplication* problem and how it yields a new algorithm, which is more efficient than the naive one. Later, we study how this paradigm can be applied to the MM problem.

6.1.1 Integer Multiplication

Suppose x, y are two n-bit integers, where n is a power of 2. (The general case is treated in a quite similar way). In the first place, we split each of these integers into their $\frac{n}{2}$-bit left and right halves, resulting in the following representations,

$$x = 2^{n/2}x_L + x_R \tag{6.1}$$

$$y = 2^{n/2}y_L + y_R \tag{6.2}$$

The product xy can be written as follows,

$$xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R \tag{6.3}$$

Thus, computing the product xy requires the same effort as the effort needed for computing the RHS of Equation 6.3. The operations involved in the RHS are additions, multiplications by powers of 2 and multiplications of $\frac{n}{2}$-bit integers. Additions take linear time, as well as multiplications by powers of 2 since they are just left-shifts. Hence, the operations of highest cost involved in the above expression are the multiplications of the four $n/2$-bit integers,

$$x_L y_L, x_L y_R, x_R y_L, x_R y_R.$$

Using this reduction, the problem of computing the multiplication of two n-bit integers is reduced to the problem of computing four multiplications of $\frac{n}{2}$-bit integers combined with 3 additions and left-shifts.

Let $T(n)$ denote the overall running time for computing the product of two n-bit integers using this method. The following recurrence relation holds

$$T(n) = 4T(n/2) + \mathcal{O}(n). \tag{6.4}$$

Working out this recurrence relation the complexity is still $\mathcal{O}(n^2)$ [61]. Thus, no improvement so far. However, Carl Friedrich Gauss (1777-1855) proved that the product of two complex numbers

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

can be computed with three multiplications instead of four. In particular we have,

$$bc + ad = (a + b)(c + d) - (ac + bd).$$

Applying Gauss' trick to Equation 6.3 we obtain,

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R,$$

Hence, 3 out of 4 products are sufficient to compute the initial expression. The recurrence relation describing the overall time complexity is now given by,

$$T(n) = 3T(n/2) + \mathcal{O}(n). \tag{6.5}$$

Thus, at every level of the recursion, we have a constant factor improvement. We can compute the running time of the algorithm by studying the algorithm's pattern of recursive calls. This is a form of a tree structure, similar to the shape of the tree shown in Figure 6.1. In the associated tree structure of this particular problem, every problem is split in three subproblems of halved size. Starting with the problem of multiplying n-bit integers, we expect the subproblems to have size 1 at the $log_2(n)$-th level of the tree. Precisely at this point, the problem becomes straightforward to solve and hence the recursion ends. Hence, the height of the tree is $log_2(n)$.

At depth k of this tree, there are exactly 3^k subproblems of size $\frac{n}{2^k}$. From depth k to identify problems at size $k + 1$, the total time spent is $3^k \times \mathcal{O}(\frac{n}{2^k})$, which equals to $(\frac{3}{2})^k \times \mathcal{O}(n)$. At top level, this quantity equals to $\mathcal{O}(n)$, while at $log_2(n)$-th level, equals to $\mathcal{O}(n^{log_2(3)})$. At each level, the time spent increases geometrically by a factor $\frac{3}{2}$ and by summing over all levels, the overall running time complexity turns out to be upper bounded by $\mathcal{O}(n^{log_2 3}) \approx \mathcal{O}(n^{1.59})$, which is a substantial improvement over the naive algorithm [61].

6.1.2 The Divide-and-Conquer Master Theorem

Formally, the Divide-and-Conquer paradigm can be seen as follows: Given a problem of size n, we split this problem into a subproblems of size n/b. Then, we solve these smaller instances of the same problem and finally we combine all these partial answers in $\mathcal{O}(n^d)$ time, for some $a, b, d > 0$ (cf. Figure 6.1).

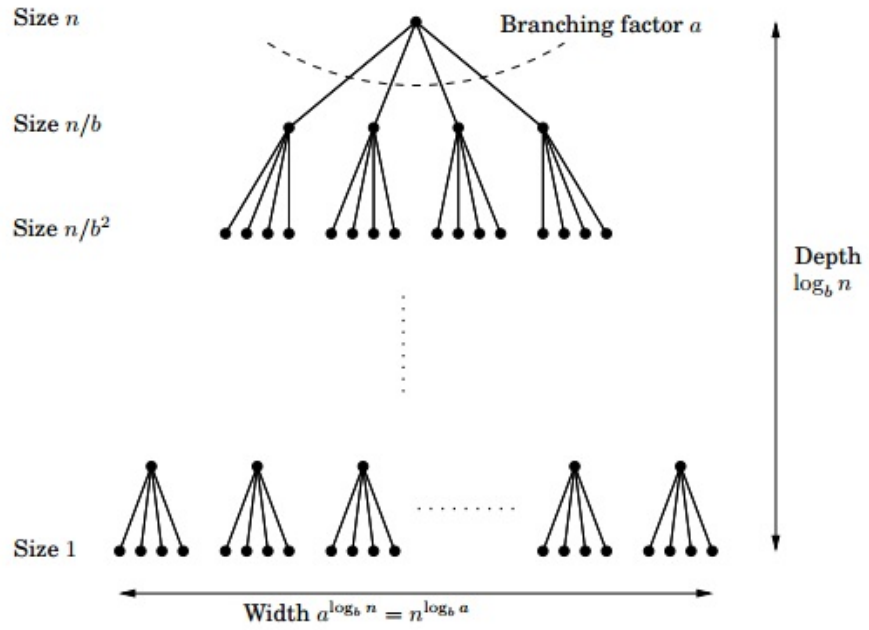

Figure 6.1: Divide and Conquer Paradigm - Recursive calls to an algorithm of size n using a branching factor a [61]

Figure 6.1 suggests that the overall running time complexity $T(n)$ for obtaining a solution to the initial problem can be obtained by adding the running time for obtaining solutions to the smaller instances. In particular we have that

$$T(n) = aT(\lceil n/b \rceil) + \mathcal{O}(n^d) \tag{6.6}$$

where a is called the *branching factor*.

The very generic structure of these algorithms allows us to compute precisely the running time $T(n)$ according to the exponent d (Theorem 7).

Theorem 7. *(Divide and Conquer Master Theorem, [61])*

Let $T(n)$ the complexity function for solving a given problem on n inputs. Suppose that this problem can be partitioned into a sub-problems on n/b inputs and the partial answers to these smaller instances can be combined together in $\mathcal{O}(n^d)$ time, for some $a > 0, b > 1, d \geq 0$ Then,

$$T(n) = aT(\lceil \frac{n}{b} \rceil) + \mathcal{O}(n^d) \tag{6.7}$$

The following cases hold

$$T(n) = \begin{cases} \mathcal{O}(n^{log_b(a)}), d < log_b(a) \\ \mathcal{O}(n^d log(n)), d = log_b(a) \\ \mathcal{O}(n^d), d > log_b(a) \end{cases}$$

Proof. More details about this proof can be found in standard textbooks about computational complexity and algorithms such as in [61]

□

This paradigm is very important and widely used in the area of computational algebra and was applied in MM problem. In the subsequent section, we outline how we can use this technique for solving problems in MM by solving smaller instances and combining them recursively.

6.2 Matrix Multiplication (MM)

The problem of Matrix Multiplication (MM) (Definition 41) is one of the most important problems in the area of mathematics and have been studied by many famous Mathematicians. Until 1968, the only known method for multiplying two matrices was the *Naive Multiplication* algorithm simply arising from the definition of the problem (cf. Definition 41).

Definition 41. *([MM])*

Let A and B two $n \times n$ matrices, $n \in \mathbb{N}$, with entries in a (not necessarily) commutative ring $\mathcal{R}$ having the following form

$$A_{n,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \text{ and } B_{n,n} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{pmatrix},$$

Then, the entries of the product matrix $C = AB$ are given by

$$C_{p,q} = [AB]_{p,q} = \sum_{i=1}^{n} a_{p,i} b_{i,q}, \tag{6.8}$$

where multiplication is defined over $\mathcal{R}$.

Based on the above definition, we observe that the naive multiplication algorithm requires $2n^3 - n^2$ operations (in total) over $\mathcal{R}$ for computing the product matrix C. More precisely, $n^3 - n^2$ additions and n^3 multiplications are required.

However, the expansion of computer technology in the late 20th century brought back to the surface the problem of (more) efficient matrix multiplication. Nowadays, algorithms for MM are widely used and a speed-up in MM will automatically result in a speed up in many other important algorithms. Below we mention some areas which would immediately benefit from *fast linear algebra* [65]. Note that this list is not exhaustive.

Applications of MM:

1. Commercial software, such as MATLAB, MATHEMATICA and GAUSS
2. Economic modeling
3. Weather prediction
4. Signal processing
5. Gauss Elimination algorithm for solving a system of linear polynomial equations
6. Algorithms for solving non-linear polynomial equations
7. Recognizing if a word of length n belongs to a contextfree language
8. Transitive closure of a graph or a relation on a finite set
9. Statistical analysis of large data sets
10. Integer factorization
11. Cryptanalysis
12. Computer graphics, e.g. games

Definition 41 can be extended to multiplication of non-square matrices. The problem of multiplying a $m \times p$ matrix A by a $p \times n$ matrix is equivalent to computing the mn bilinear forms $C_{i,l} = \sum_{j=1}^{p} a_{i,j} b_{j,l}$ for $1 \leq i \leq m$ and $1 \leq l \leq n$ (cf. Definition 42).

Definition 42. *(Bilinear Non-commutative Algorithm)*

Let $\mathcal{R}$ a non-commutative ring and consider the two sets of variables $A = \{a_1, a_2, ..., a_r\}$ and $B = \{b_1, b_2, ..., b_s\}$. Then, a bilinear algorithm *over A, B and ring $\mathcal{R}$ is an algorithm which computes products of the form*

$$(\sum_{i=1}^{r} \alpha_i^t a_i)(\sum_{j=1}^{s} \beta_j^t b_j)$$

in the variables in the sets A, B using coefficients $\alpha_i^t, \beta_j^t \in \mathcal{R}$, where $t = 1, 2, ..., d$ and d is the number of required products we need to compute.

The number d of multiplications is called the *bilinear complexity* of the algorithm. A bilinear algorithm over a ring $\mathcal{R}$ calculates a set of bilinear forms. One way of solving MM problems is to search for bilinear algorithms over the set of the entries of the matrices A, B which have minimal bilinear complexity. Note that in case of MM, the algorithm is actually trilinear since in first place we use a bilinear algorithm to compute some products and then we combine these products linearly to compute the entries of the product matrix $C = AB$. In the next section, we provide a historical overview regarding the bilinear complexity of the MM problem and the progress up to date.

6.2.1 On the Exact Bilinear Complexity of MM

As we have already mentioned, the time complexity of multiplying two $n \times n$ matrices using the naive multiplication algorithm is $\mathcal{O}(n^3)$. Let $M_{\mathcal{R}}(n)$ the total number of operations required for the multiplication of two $n \times n$ matrices over the ring $\mathcal{R}$. The exponent of MM is defined as follows (Definition 43).

Definition 43. *The exponent of MM over a general non-commutative ring $\mathcal{R}$ is defined as*

$$\omega(\mathcal{R}) := \inf\{\tau \in \mathbb{R} | M_{\mathcal{R}}(n) = \mathcal{O}(n^\tau)\}$$

An upper bound for $\omega(\mathcal{R})$ is 3 (by naive multiplication algorithm). The output of the multiplication of two $n \times n$ matrices is an $n \times n$ matrix and thus it contains n^2 entries. This implies at least n^2 operations are needed to compute it. Thus, $\omega(\mathcal{R}) \in [2, 3]$.

Since 1968, many mathematicians tried to improve the upper bound of $\omega(\mathcal{R})$ and some researchers have conjectured that MM complexity could be nearly quadratic, for example like $\mathcal{O}(n^2(log(n))^\alpha)$. In Figure 6.2 we show the progress up to date.

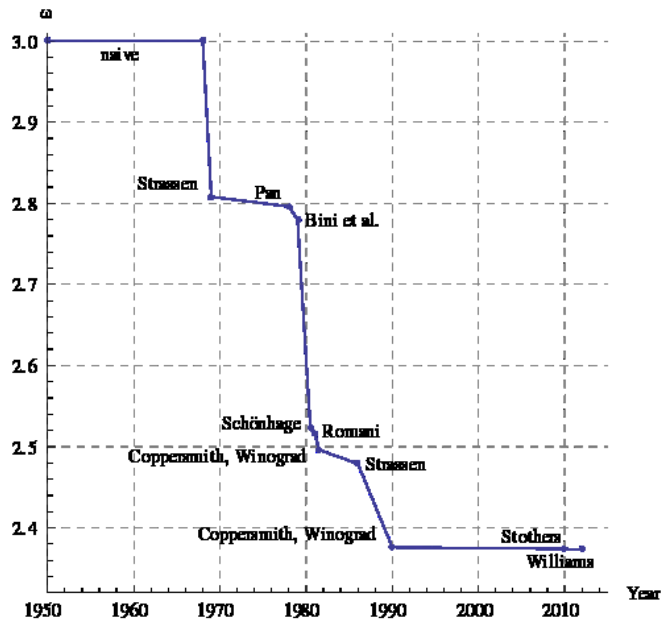

Figure 6.2: On the Complexity of MM - Bound ω in MM over time [1]

The first attempt to improve the exponent $\omega(\mathcal{R})$ was by Winograd [121]. He computed the inner product of two n-dimensional vectors using fewer multiplications but more additions. As MM can be seen as a computation of many different inner products, this technique can slightly improve the number of operations needed. This technique resulted in a slight improvement in the exponent ω. More details are found in [121].

Within the same year, Strassen discovered a non-commutative bilinear algorithm for multiplying two 2×2 matrices using 7 multiplications instead of 8 (cf. Algorithm 7).

Algorithm 7 Strassen's Non-Commutative Bilinear Algorithm using 7 Multiplications

Given two 2×2 matrices A, B over a ring $\mathcal{R}$, with entries $a_{i,j}, b_{i,j} \in \mathcal{R}$ $1 \leq i, j \leq 2$, then the entries $c_{i,j}$ of the product matrix $C = AB$ can be computed by the following formulas,

$P_1 = (a_{1,1} + a_{2,2})(b_{1,1} + b_{2,2})$
$P_2 = (a_{2,1} + a_{2,2})b_{1,1}$
$P_3 = a_{1,1}(b_{1,2} + b_{2,2})$
$P_4 = a_{2,2}(-b_{1,1} + b_{2,1})$
$P_5 = (a_{1,1} + a_{1,2})b_{2,2}$
$P_6 = (-a_{1,1} + a_{2,1})(b_{1,1} + b_{1,2})$
$P_7 = (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2})$,
$c_{1,1} = P_1 + P_4 - P_5 + P_7$
$c_{1,2} = P_2 + P_4$
$c_{2,1} = P_3 + P_5$
$c_{2,2} = P_1 + P_3 - P_2 + P_6$

Algorithm 7 can be extended recursively to an algorithm for multiplying two $n \times n$ matrices with $\omega \leq log_2(7)$ using the Divide-and-Conquer paradigm (cf. Lemma 3).

Lemma 3. *Let A, B two $n \times n$ matrices over a ring $\mathcal{R}$. Then, the application of Strassen's algorithm gives an improved algorithm for multiplying A and B in time*

$$\mathcal{O}(n^{log_2 7}) \approx \mathcal{O}(n^{2.81})$$

Proof. Write the matrices A, B in the following way,

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \text{ and } B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix},$$

where A_i, B_i for $1 \leq i \leq 4$ are $\frac{n}{2} \times \frac{n}{2}$ square matrices over $\mathcal{R}$.

Then, the product matrix AB is given by,

$$AB = \begin{pmatrix} A_1B_1 + A_2B_3 & A_1B_2 + A_2B_4 \\ A_3B_1 + A_4B_3 & A_3B_2 + A_4B_4 \end{pmatrix}$$

Therefore, using the Divide-and-Conquer algorithm, we can compute the product of size n by computing eight products of size $\frac{n}{2}$. In particular, we need to compute the following products.

$$\{A_1B_1, A_2B_3, A_1B_2, A_2B_4, A_3B_1, A_4B_3, A_3B_2, A_4B_4\},$$

Then, the total running time for computing the product AB is given by,

$$T(n) = 8T(\frac{n}{2}) + \mathcal{O}(n^2). \tag{6.9}$$

Strassen's algorithm hold in general over any non-commutative ring and thus using Algorithm 7 we can rewrite the product AB in the following way,

$$AB = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}.$$

The new running time is improved and given by

$$T(n) = 7T(\tfrac{n}{2}) + \mathcal{O}(n^2),$$

which by Theorem 7 works out to be $\mathcal{O}(n^{log_2 7}) \approx \mathcal{O}(n^{2.81})$ □

Since then, many substantial improvements have been conducted in this area. Coppersmith and Winograd in 1987 improved the bound to $\omega \leq 2.3755$ [29]. Stothers improved the bound to $\omega \leq 2.3737$ [117] and the final successful attempt was in 2011 by Williams, who improved the bound to $\omega \leq 2.3727$ [120]. Amazingly enough, many scientists conjecture that it could by nearly quadratic like $\mathcal{O}(n^2(log_2(n)^\alpha))$ for some α. In the next section, we present an automated discovery methodology for searching for bilinear algorithms of reduced bilinear complexity based on SAT solvers.

6.2.2 Encoding of MM and the Brent Equations

Richard Brent (in his thesis) developed non-commutative bilinear algorithms for the MM problem [26]. His main objective was to obtain an algebraic description of the problem of multiplying two matrices using a fixed number of multiplications. By an algebraic description, we mean a system of equations with variables the entries of the matrices and the number of multiplications that we would like to use.

Importantly, this implies that if this system has a solution, then there exists a non-commutative bilinear algorithm of desired bilinear complexity, which computes the product matrix. Otherwise, no such algorithm exists. Theorem 8 describes this technique [26].

Theorem 8. *(Brent Equations)*

The entries of a $n \times n$ product matrix $C = AB$ are to be constructed from linear equations of κ products, in which the operands are linear combinations of the elements of A and B respectively.

Specifically, given κ triples of constant $n \times n$ matrices, $((\alpha_{mk}^{(u)}), (\beta_{li}^{(u)}), (\gamma_{js}^{(u)})$, where $1 \leq u \leq \kappa$, if

$$C_{js} = \sum_{u=1}^{\kappa} (\sum_{m,k} A_{km} \alpha_{mk}^{(u)}) (\sum_{l,i} B_{il} \beta_{li}^{(u)}) \gamma_{js}^{(u)}, \tag{6.10}$$

for all $j, s \in [0, n)$, then the $3n^2\kappa$ elements of the 3κ constant matrices satisfy Brent's set of n^6 equations,

$$\textstyle\sum_{u=1}^{\kappa} \gamma_{js}^{(u)} \beta_{li}^{(u)} \alpha_{mk}^{(u)} = \delta_{ls} \delta_{mi} \delta_{jk},$$

for all $i, j, k, l, m, s \in [0, n)$, where $\delta_{xy} = 1$ if $x = y$ and 0 otherwise.

Proof. The proof follows by equating coefficients $A_{km}B_{il}$ in the first equation with those in $C_{js} = \sum_r A_{jr}B_{rs}$ □

The above reformulation implies the existence of a bilinear algorithm of bilinear complexity κ, provided that the associated system of Brent Equations has a solution over the ring $\mathcal{R}$ in some cases.

Importantly, this reformulation of MM problem using Brent Equations provides a tool for formal encoding of the problem and it can be used in an automated procedure for discovery of bilinear algorithms of pre-specified bilinear complexity.

Given a formal algebraic encoding of the problem and considering it over the Galois Field of two elements $\mathbb{F}_2$, we then compute the associated CNF-SAT form [11] and finally we search for a solution over $\mathbb{F}_2$ using SAT solvers. This leads to an automated discovery of new solutions to the MM problem over $\mathbb{F}_2$, which can heuristically be extended to general rings $\mathcal{R}$ [43].

Algorithm 8 Automated Solutions for MM problem via SAT solver

1. Form the Brent Equations of the corresponding MM problem
2. Convert it to CNF using the Courtois-Bard-Jefferson Method [11].
3. Search for solutions valid mod 2.
4. The SAT-solver terminates either with SAT and a solution or with UNSAT meaning there is no solution.
5. Lift the solution to a general ring $\mathcal{R}$ by another constraint satisfaction algorithm.

6.2.3 On the Equivalence of Bilinear Algorithms for MM

Given two solutions obtained by Algorithm 8, how do we know if these two solutions are non-isomorphic or non-equivalent (cf. Definition 44)?. Two solutions are considered isomorphic if one solution can be transformed into the other solution using a group of transformations. Such transformations for the isomorphism of bilinear algorithms used for solving MM problem were studied extensively in [76].

Definition 44. *(Equivalence of Solutions [76])*

Let $A^{(u)}, B^{(u)}, \Gamma^{(u)}$ $(1 \leq u \leq \kappa)$ the matrices as defined in Theorem 8. The following transformations map a given solution to another (isomorphic) solution,

1. *Permuting the κ indexes u*
2. *Cyclic shift of the three set of matrices $A^{(u)}, B^{(u)}, \Gamma^{(u)}$, $\forall 1 \leq u \leq \kappa$*
3. *Change of order followed by transposition. Replace $\{A^{(u)}, B^{(u)}, \Gamma^{(u)}\}$ by $\{(\Gamma^{(u)})^T, (B^{(u)})^T, (A^{(u)})^T\}$, $\forall 1 \leq u \leq \kappa$*
4. *Replace $\{A^{(u)}, B^{(u)}, \Gamma^{(u)}\}$ by $\{a_u A^{(u)}, b_u B^{(u)}, \gamma_u \Gamma^{(u)}\}$, where $a_u, b_u, \gamma_u \in \mathbb{Q}$ such that $a_u b_u \gamma_u = 1$, $\forall 1 \leq u \leq \kappa$*
5. *Replace $\{A^{(u)}, B^{(u)}, \Gamma^{(u)}\}$ by $\{UA^{(u)}V^{-1}, VB^{(u)}W^{-1}, W\Gamma^{(u)}U^{-1}\}$, where U, V, W are arbitrary invertible matrices*

The transformations described above can be used to check if two bilinear algorithms of the same bilinear complexity are isomorphic. As we will see later, we will use these transformations to check if the bilinear algorithm for computing two 3×3 matrices of bilinear complexity 23 which we have obtained by our methodology is isomorphic to Laderman's algorithm [87].

6.3 Circuit Complexity (CC)

The problem of CC is considered as one of the very hard problems in mathematics and computer science. In particular, obtaining an optimal circuit representation under almost any meaningful metric (gate count, depth, energy consumption, etc.) is a hard computational problem. For example, there is no known provably optimal circuit representation with respect to Boolean complexity for a Boolean function of the form $\mathbb{F}_2^8 \to \mathbb{F}_2$ [25, 22].

None of the existing algorithms is able to synthesize small and compact implementations for the greater majority of circuits. In practice, there are no analytical techniques yielding optimal representations and most of them rely on a variety of well-chosen ad-hoc heuristics. Unfortunately, many of these heuristics have exponentially large running time complexity and as a result they can be applied for the optimization of very small circuits only.

However, given a large enough circuit, optimizing its smaller sub-circuits yields a very efficient (but not proven optimal) circuit. This technique can be applied to functions that naturally decompose into repeated use of smaller subcomponents. For example, as we have seen for the MM problem, as well as directly in cryptography. Most of the cryptographic functions are based on design architectures, which can be seen as multiple iterations of a cryptographically weaker function f which contains several linear steps and a few non-linear (e.g. S-boxes). Thus, optimizing this round function yields an improved implementation of the whole cipher. Below we discuss some immediate benefits arising from such optimal circuit representations [42, 39].

Applications of CC:

1. Develop certain "Bitslice" parallel-SIMD software implementations of block ciphers such as in [6].

2. Lower the hardware implementation cost of encryption algorithms in silicon. Such implementations are particularly important in smart cards and RFID, where cryptographic algorithms are the main computationally costly components.

3. Optimizing a circuit with respect to its MC is a countermeasure against Side Channel Attacks (SCA) on smarts cards, such as Differential Power Analysis

(DPA) [104]. This is simply because $\oplus$ gates are easier to protect against side-channel attacks.

4. Cryptanalysis based on SAT-solver techniques benefits immediately from such improved implementations, as the time taken for a SAT solver to find a solution depends on the compactness of the circuit. Low-data complexity attacks such as algebraic attacks might be improved if a more *compact* and *efficient* representation of a cipher is found [42, 45].

5. Direct application in numerical algebra and symbolic computing. Such optimizations can be applied recursively for improving other algorithms such as, Gaussian reduction and MM.

We have developed a 2-step method based on formal coding and SAT solvers for searching for optimal circuit representations with respect to meaningful metrics to cryptology and it is very similar to the previous methodology used for the MM problem. We view the problem as a *constraint satisfaction problem*. We apply this methodology to optimize a given circuit with respect to a given metric, which is dependent on its application. For example the notion of MC (cf. Definition 45) is very important for numerical algorithms and software that employ routines for multiplying matrices of large dimensions.

Definition 45. ***Multiplicative Complexity (MC):*** *The minimum number of AND gates required to compute a given circuit if we allow an unlimited number of NOT and XOR gates.*

Moreover, the notion of *bitslice* complexity (cf. Definition 46) is precisely relevant in bitslice implementations of block ciphers on standard CPUs.

Definition 46. ***Bitslice Gate Complexity (BGC):*** *The minimum number of 2-input gates of type XOR,OR,AND,NOT needed to compute a given circuit.*

In addition, the notion of Gate Complexity (Definition 47) is very useful for silicon optimization. It is the most general notion implemented by designers of digital circuits. This model is relevant in so called bitslice parallel-SIMD implementations of block ciphers [6].

Definition 47. ***Gate Complexity (GC):*** *The minimum number of 2-input gates of type XOR,AND,OR,NAND,NOR,NXOR needed to compute a given circuit.*

Lastly, we provide optimizations with respect to the NAND complexity (Definition 48), which are important for cryptanalysis.

Definition 48. ***NAND Complexity (NC):*** *The minimum number of 2-input NAND gates needed to compute a given circuit.*

6.4 Combinatorial Circuit Optimization

In 2008, Boyar and Peralta developed a 2-step (heuristic) methodology based on the notion of MC. This methodology was applied to the S-box of AES and the smallest circuit known is due to this methodology, which consists of 32 AND gates and 83 XOR/XNOR gates, for a total of 115 gates [24]. This heuristic methodology for designing gate-efficient implementations is as follows,

- *STEP 1:* Compute the Multiplicative Complexity.
- *STEP 2:* Optimize the number of XORs separately.
- *STEP 3(optional):* Perform additional optimizations to decrease the circuit depth, possible gate count, power consumption, etc.

This is a completely heuristic methodology and there is no way of proving that it outputs optimal results. However, it depends on a key observation that circuits with low MC will naturally have large sections which contain only $\oplus$ gates and thus we can further optimize the circuits locally. The first step of this technique considers identifying nonlinear components and optimize the given circuit representation with respect to AND gates. However, finding circuits with minimal MC is still a highly intractable problem [25]. This step depends on the structure of the given circuit and the selection of ad-hoc heuristics is essential. However, this reduction is not trivial to do. For example, the two circuits shown in Figure 6.3, both compute the function

$$MAJ(a,b,c) = \lfloor \frac{1}{2} + \frac{a+b+c-1/2}{3} \rfloor, \tag{6.11}$$

which is true if and only if more than half of its inputs are true. This is the Majority function on three inputs, which we study in a latter subsection. However, it is not easy to algorithmically transform one circuit into the other.

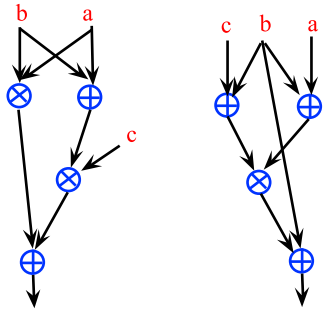

Figure 6.3: $MAJ(a, b, c)$ **Circuit Representation** - Two different circuit representations computing the same function. The circuit on the left has 2 AND and 2 XOR gates, while the other on the right has 3 XOR and only 1 AND gate

We have developed a simple tool for computing the MC of a given circuit (cf. Algorithm 9) [42]. The Brent-like approach which we used in the MM problem could lead to a very large problem to solve and thus this method is very different. In MC optimizations, we can say that each variable is an affine or linear combination of previous variables.

Given a circuit representation, we firstly encode the circuit as a straight-line optimization problem and then we encode it as a system of multivariate equations with variables which can be either 0 or 1. In our encoding we use four distinct sorts of variables. We denote the input bits to the truth table as x_i, the output bits to the truth table as y_i, the input and each output of each gate as z_i and finally we use the variables a_i to express some intermediate states as an affine combination of previous variables. Algorithm 9 describes precisely the procedure how to obtain the system of multivariate equations which express the problem of our interest.

Algorithm 9 can be used to compute the MC of a given circuit. If the integers MIN_{SAT} and MAX_{UNSAT} satisfy $MIN_{SAT} - MAX_{SAT} = 1$, then it means the MC of the circuit is MIN_{SAT}. There are some cases where the SAT solver does not terminate and we output -1 or whether no SAT was obtained in an output and we output ∞. In the cases we study, we deal up with circuits of small size and thus we test small integers α and β and a timeout T of about 1000 seconds.

Algorithm 9 Method of computation of MC of a given circuit

INPUT: Truth Table of a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, an interval $[\alpha, \beta]$ where α, β are positive integers and a timeout bound T (in seconds)

1. For all integers M such that $\alpha \leq M \leq \beta$, Do Steps 2-10

2. We define by Z a working set of internal variables and let $|Z|$ be its cardinality

3. We create a test file called Prototype as follows (Steps 3a-4)

3a. Let $\kappa = M$

3b. Initially $Z = \{x_1, x_2, ..., x_n\}$ which corresponds to the input bits and $|Z| = n$

3c. While $\kappa > 0$, Do

3d. Set $r = 1$

3e. Write z_r as the product of the following two terms $a_{|Z|}^{(r)} t_{|Z|} + ... + a_1^{(r)} t_1 + a_0^{(r)}$ with $a_j^{(r)} \in \mathbb{F}_2$ $(1 \leq j \leq |Z|)$ and $a_{|Z|}^{\prime(r)} t_{|Z|} + ... + a_1^{\prime(r)} t_1 + a_0^{\prime(r)}$ with $a_j^{\prime(r)} \in \mathbb{F}_2$ $(1 \leq j \leq |Z|)$ and $t_i \in Z$ for all $1 \leq i \leq |Z|$

3c. Insert z_r in Z

3d. Set $\kappa = \kappa - 1$ and $r = r + 1$

4. Write affine equations which make each output y_i $(1 \leq i \leq m)$ an affine combination of variables from Z, with coefficients being some other $a_i \in \mathbb{F}_2$.

5. Convert this Prototype system of equations over $\mathbb{F}_2$ to SAT with the Courtois-Bard-Jefferson tool. Now all variables that appear in the Prototype are replaced by unique positive integers via the mapping $\pi : Z' \to \mathbb{Z}$, where Z' is the set of all variables of the form $x_i, y_i, z_i, t_i, a_i^{(j)}, a_i^{\prime(j)}$, and π maps the variables to their corresponding number.

6. Produce 2^n copies of this SAT problem. In each instance, renumber all the $\pi(z_i)$ variables to distinct numbers using new unused integers and keep all the $\pi(a_i)$ the same.

7. For each copy substitute the x_i and y_i with the input and output values respectively. This corresponds to all possible evaluations of the same circuit.

8. Concatenate the 2^n copies of the SAT problem.

9. Attempt to solve using a SAT solver and abort after T seconds

10. Convert back from CNF to circuit representation. Replace in the Prototype all the a_i found by their values 0(false) or 1 (true) using the mapping π.

Output: Two integer values; MIN_{SAT}: the smallest value such that SAT was obtained (∞ if no SAT was obtained) and MAX_{UNSAT}: the biggest value such that UNSAT was obtained (-1 if no UNSAT was obtained)

In the Boyar and Peralta methodology, after the computation of the MC, we optimize the number of XORs separately [21, 69]. Optimizing the number of XORs in a given circuit is an intractable problem and has been proven to be NP-hard [21]. This can be proved over $\mathbb{F}_2$ by reducing it to the *Vertex Cover* problem, which is already proven to be NP-hard [21]. Boyar *et al* proved in the same paper, that the problem does not have ϵ-approximation schemes, unless $P = NP$. That implies there is no method of obtaining a solution close to the optimal solution (up to a small error ϵ) and thus the only chance of getting a better solution is by improving existing approximation techniques based on a variety of heuristics.

Algorithm 9 can be used for computing the MC of a given circuit only. In case, we would like to compute the complexity of a given circuit with respect to a different metric we need a different methodology. Algorithm 10 is designed to address optimizations with respect to a more complex metric. For this problem, we consider six sorts of variables of the following form,

- x: Inputs of the truth table
- y: Outputs of the truth table
- q, q': Inputs of gates
- t: Outputs of gates
- b: Variables which define the function of each gate (e.g. one gate could be AND, OR, XOR and the model is $b(uv) + b'(u + v)$, and when $b = 1, b' = 0$ this will be AND gate)
- a: Variables which will be the unknown connections between different gates

The motivation behind this algorithm is that not every circuit is very algebraic and thus it cannot be efficiently described by sparse multivariate polynomial expressions. This is quite usual in industrial implementations, where circuits of low gate count are preferred for efficient hardware implementations. Precisely for this reason, using Brent-like equations could lead to very huge problem to solve. However, we can describe the initial problem as a substitution box with a truth table and proceed as described in Algorithm 10.

Algorithm 10 General method of computation of exact complexity of a given circuit wrt a given metric

INPUT: Truth Table of a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, an interval $[\alpha, \beta]$ where α, β are positive integers and a timeout bound T (in seconds)

1. For all integers M such that $\alpha \leq M \leq \beta$, Do Steps 2-10

2. We define by Z a working set of internal variables and let $|Z|$ be its cardinality

3. We create a test file called Prototype as follows (Steps 3a-4)

3a. Let $\kappa = M$

3b. Initially $Z = \{x_1, x_2, ..., x_n\}$ which corresponds to the input bits and $|Z| = n$

3c. While $\kappa > 0$, Do

3d. Set $r = 1$

3e. Write $q_r = a_{|Z|}^{(r)} z_{|Z|} + ... + a_1^{(r)} z_1 + a_0^{(r)}$ and $q'_r = a_{|Z|}^{\prime(r)} z_{|Z|} + ... + a_1^{\prime(r)} z_1 + a_0^{\prime(r)}$, where only one $a^{(r)}$ is allowed to be 1 and only one $a^{\prime(r)}$ is allowed to be 1 (This is achieved by adding all the quadratic equations of the form $a^{(i)}.a^{(j)} = 0$ and $a^{(i)}.a^{(j)} = 0$ for all $1 \leq i, j \leq |Z|$ with $i \neq j$ in our system).

3f. Write t_r in the form $b_r(q_r q'_r) + b'_r(q_r + q'_r) + b''_r$ where $b_r, b'_r, b''_r \in \mathbb{F}_2$

3g. Insert t_r in Z

3h. Set $\kappa = \kappa - 1$ and $r = r + 1$

4. Set each output y_i $(1 \leq i \leq m)$ to be equal to one of the previous variables in Z

5. Make several copies of the same problem, renaming all x, y, q, q', t variables but leaving a, b the same.

6. Consider all these copies together and form a large system of multivariate equations

7. Convert this to SAT using Courtois-Jefferson-Bard software [11]. Now all variables that appear in the Prototype are replaced by unique positive integers via the mapping $\pi : Z' \to \mathbb{Z}$, where Z' is the set of all variables of the form $x_i, y_i, q_i, q'_i, a_i^{(j)}, a_i^{\prime(j)}, t_i, b_i, b'_i, b''_i$, and π maps the variables to their number

8. Substitute the known pairs (x, y) and solve for a, b

9. Substitute the values a_i, b_j obtained by SAT solver to the Prototype using the mapping π

Output: Two integer values; MIN_{SAT}: the smallest value such that SAT was obtained (∞ if no SAT was obtained) and MAX_{UNSAT}: the biggest value such that UNSAT was obtained (-1 if no UNSAT was obtained)

A very important task involved in Algorithm 10 is how to describe the connections between gates. The idea is that each variable t is some combination of previous variables of type x, t, q, q' and the variables a are used to encode all the unknown connections between different gates, their inputs and outputs, and the pairs of inputs-outputs (x, y). For example, in MC optimizations we can say that each variable is an affine (or linear) combination of past variables. In other optimizations we can add constraints of type $a_i a_j = 0$, which says that in a certain set of a_i only at most one of these variables is at 1.

6.4.1 Provable Optimality in Complexity Theory

The described methodology based on SAT solvers is a very powerful technique, as it can be used for discovering bilinear algorithms of least bilinear complexity for MM, as well as for constructing optimal circuits with respect to several meaningful metrics.

Given a CNF-SAT problem, a SAT solver either terminates in reasonable time or not. In the first case, it will output either SAT with a satisfying logical assignment to the problem or UNSAT, which means no solution exists. In the second case, the SAT solver does not terminate on the given problem in reasonable time and thus no conclusions can be drawn. However, in all cases we describe in this thesis are related to the first case.

In particular, in MM we can apply this methodology to prove for example if the multiplication of two matrices can be done with **exactly** κ multiplications. In SAT solvers language, this is translated to SAT in κ multiplications and to UNSAT in $\kappa - 1$ or fewer multiplications. Similarly, for obtaining optimal circuit representations for sufficiently small circuits.

Using this methodology we are able to find solutions which are optimal and proven to be impossible to further improve. However, we are not able to provide *provably optimal* results since we lack of a proof of correctness of the SAT solver software and there is the possibility of a bug in a SAT solver. A bug means that maybe a problem which is SAT is claimed to be UNSAT by another solver. However, we can minimize this problem by using different SAT-solvers and checking if the results obtained are indeed consistent.

We have obtained a new bilinear algorithm of bilinear complexity 23 for the problem of multiplying two square 3×3 matrices. Moreover, we have been able to prove that

the product matrix of two matrices $A \in M_{2\times 2}(R)$ and $A \in M_{2\times 3}(R)$ requires exactly 11 multiplications. Lastly, we applied this methodology for optimizing the S-boxes of the ciphers GOST, PRESENT and CTC2 with respect to their MC.

Remark 12. *All results that are presented in the next section were obtained using a PC having an Intel i7 1.73 GHz processor and 4.00 GB RAM.*

6.4.2 Application of Our Method to MM problems

As a proof of concept, we have applied our methodology to multiply two 2×2 matrices using 7 multiplications. We have converted the total of 64 equations over $\mathbb{F}_2$ to a CNF-SAT problem using the Courtois-Bard-Jefferson software and then we solved this problem using CryptoMiniSat in approximately 0.10s. In order to prove that 7 is the exact number of required multiplications, we have encoded the problem for 6 multiplications and we have obtained UNSAT after 524.41s=0.146h. This is a complete proof that the bilinear complexity of this problem is exactly 7 over $\mathbb{F}_2$.

However, the solutions obtained are valid only over $\mathbb{F}_2$. Our aim is to find solutions over any ring $\mathcal{R}$. Given a solution we proceed step by step by lifting the solution to larger and larger rings. We have developed a heuristic methodology for lifting a solution over $\mathbb{F}_2$ to a solution over $\mathbb{Z}/4\mathbb{Z}$ and then progressively proceed to $\mathbb{Z}/4\mathbb{Z}$ and so on. Our Heuristic Lifting Technique works as follows.

Heuristic Lifting to Larger Rings

The map $\mathbb{Z}/2\mathbb{Z} \to \mathbb{Z}/8\mathbb{Z}$ given by $0 \to \{0, 2\}$ and $1 \to \{1, -1\}$ is an embedding of the first field to the latter ring. Given an element α over $\mathbb{F}_2$, we map it firstly to 0 if it is 0 and firstly to 1 and then to -1 if it is equal to 1. Then, we map 0 to 2 and 1 is mapped firstly to 1 and then to -1. We use this heuristic to map a given solution of Brent Equations over $\mathbb{F}_2$ to a solution over $\mathbb{Z}/4\mathbb{Z}$.

Remark 13. *Based on observations, most of the times 0 is mapped to 0. In addition, for the majority of the cases we have tried, lifting to $\mathbb{Z}/4\mathbb{Z}$ was enough since the solution was already true in general.*

Applying the Lifting (heuristic) methodology to the solutions over $\mathbb{F}_2$ for the problem of multiplying two 2×2 matrices using 7 multiplications yields the following result.

Application of Heuristic Technique: Given two 2×2 matrices, we consider the corresponding MM problem with 7 multiplications over $\mathbb{F}_2$. Using MiniSat we obtained

solutions for the variables a, b, c involved to the total of 64 Brent Equations over $\mathbb{F}_2$. Then, we lifted these equations (and the solutions) over $\mathbb{Z}/4\mathbb{Z}$. We have obtained solutions (cf. Appendix A.2), which are also true over a general ring $\mathcal{R}$.

Moreover, we have been able to obtain solutions to other MM problems, such as the multiplication of a 2×2 matrix by a 2×3 matrix using 11 multiplications and the problem of multiplying two 3×3 matrices. The following theorems summarize the results we have obtained.

Theorem 9. *Given two matrices $A \in M_{2\times 2}(\mathcal{R})$ and $B \in M_{2\times 3}(\mathcal{R})$, where R is an arbitrary (not necessarily) commutative ring, the product matrix $C = A.B$ can be computed using exactly 11 multiplications. Similarly multiplying a 3×2 matrix by a 2×2 matrix requires precisely 11 multiplications.*

Proof. An upper bound for solving this problem is by naive matrix multiplication algorithm and it is 12 multiplications in total over a general (not necessarily) commutative ring $\mathcal{R}$.

Firstly, we consider the Brent Equations corresponding to 11 multiplications. Thus, we obtain 144 equations in 176 unknowns (12098 right clauses). Then, we convert it to a CNF-SAT problem, which we solve using CryptoMiniSat in approximately 474.54s=0.132h. We have obtained the following bilinear algorithm.

$P01 := (-a_{11} - a_{12} + a_{21} + a_{22}) * (b_{23})$;
$P02 := (-a_{11} - a_{12} + a_{21}) * (b_{12} - b_{23})$;
$P03 := (a_{11} - a_{21}) * (-b_{13} + b_{23})$;
$P04 := (a_{11}) * (b_{11})$;
$P05 := (a_{22}) * (-b_{21} + b_{23})$;
$P06 := (-a_{11} - a_{12}) * (b_{12})$;
$P07 := (a_{21}) * (b_{12} - b_{13})$;
$P08 := (a_{22}) * (b_{21} - b_{22})$;
$P09 := (a_{12}) * (-b_{12} + b_{22})$;
$P10 := (a_{21}) * (-b_{11} + b_{12})$;
$P11 := (a_{12}) * (b_{21})$;
$c_{11} = (P04 + P11)$;
$c_{12} = (-P06 + P09)$;
$c_{13} = (P02 - P03 - P06 - P07)$;
$c_{21} = (P01 + P02 - P05 - P06 - P10)$;
$c_{22} = (P01 + P02 - P05 - P06 - P08)$;
$c_{23} = (P01 + P02 - P06 - P07)$;

Initially, only 117 out of 144 equations were also true over $\mathbb{Z}/4\mathbb{Z}$. Applying the above heuristic lifting technique, we have lifted all solutions over $\mathbb{Z}/4\mathbb{Z}$. The solution was also true for an arbitrary ring. Then, we have obtained the corresponding Brent Equations for 10 multiplications (144 equations in 160 unknowns). The output of the SAT-solver was UNSAT, implying that there is no solution. Hence, 11 multiplications is the minimum number of required multiplications.

Note: It is easy to see that, if the number of multiplications required to multiply a $m \times p$ matrix A by a $p \times n$ matrix B is κ, then the same number of multiplications is required to multiply $(B^T)(A^T)$ [73]. □

Theorem 10. *Given two square matrices $A, B \in M(R, 3)$, where R an arbitrary non-commutative ring, we can compute the product matrix $C = A.B$ using at most 23 multiplications*

Proof. An upper bound for solving this problem is 27 (naive algorithm). Firstly, we consider the Brent Equations corresponding to 23 multiplications (729 equations in 621 unknowns). Then, we convert it to a SAT problem, which we solve using CryptoMiniSat and we obtain the following solution following precisely the same methodology described in the previous theorem.

$P01 := (a_{23}) * (-b_{12} + b_{13} - b_{32} + b_{33});$
$P02 := (-a_{11} + a_{13} + a_{31} + a_{32}) * (b_{21} + b_{22});$
$P03 := (a_{13} + a_{23} - a_{33}) * (b_{31} + b_{32} - b_{33});$
$P04 := (-a_{11} + a_{13}) * (-b_{21} - b_{22} + b_{31});$
$P05 := (a_{11} - a_{13} + a_{33}) * (b_{31});$
$P06 := (-a_{21} + a_{23} + a_{31}) * (b_{12} - b_{13});$
$P07 := (-a_{31} - a_{32}) * (b_{22});$
$P08 := (a_{31}) * (b_{11} - b_{21});$
$P09 := (-a_{21} - a_{22} + a_{23}) * (b_{33});$
$P10 := (a_{11} + a_{21} - a_{31}) * (b_{11} + b_{12} + b_{33});$
$P11 := (-a_{12} - a_{22} + a_{32}) * (-b_{22} + b_{23});$
$P12 := (a_{33}) * (b_{32});$
$P13 := (a_{22}) * (b_{13} - b_{23});$
$P14 := (a_{21} + a_{22}) * (b_{13} + b_{33});$
$P15 := (a_{11}) * (-b_{11} + b_{21} - b_{31});$
$P16 := (a_{31}) * (b_{12} - b_{22});$
$P17 := (a_{12}) * (-b_{22} + b_{23} - b_{33});$
$P18 := (-a_{11} + a_{12} + a_{13} + a_{22} + a_{31}) * (b_{21} + b_{22} + b_{33});$

$P19 := (-a_{11} + a_{22} + a_{31}) * (b_{13} + b_{21} + b_{33});$
$P20 := (-a_{12} + a_{21} + a_{22} - a_{23} - a_{33}) * (-b_{33});$
$P21 := (-a_{22} - a_{31}) * (b_{13} - b_{22});$
$P22 := (-a_{11} - a_{12} + a_{31} + a_{32}) * (b_{21});$
$P23 := (a_{11} + a_{23}) * (b_{12} - b_{13} - b_{31});$

$c_{11} = P02 + P04 + P07 - P15 - P22;$
$c_{12} = P01 - P02 + P03 + P05 - P07 + P09 + P12$
$+P18 - P19 - P20 - P21 + P22 + P23;$
$c_{13} = -P02 - P07 + P17 + P18 - P19 - P21 + P22;$
$c_{21} = P06 + P08 + P10 - P14 + P15 + P19 - P23;$
$c_{22} = -P01 - P06 + P09 + P14 + P16 + P21;$
$c_{23} = P09 - P13 + P14;$
$c_{31} = P02 + P04 + P05 + P07 + P08;$
$c_{32} = -P07 + P12 + P16;$
$c_{33} = -P07 - P09 + P11 - P13 + P17 + P20 - P21;$ □

A lot of research was carried out in order to multiply two 3×3 matrices using 23 multiplications. Even though we have obtained another solution using 23 multiplications, our achievement is still important as the solution obtained is not isomorphic to the first solution found by Laderman [87]. This suggests that the space of solutions to Laderman's problem is probably larger than expected.

In a previous subsection, we have discussed a complete characterization of transformations, which yield to isomorphic bilinear algorithms for the MM problem (cf. Theorem 11). In Theorem 12 we prove that the solution found is not isomorphic to Laderman's solution.

Theorem 11. *(Invariant for Equivalent Solutions)*
All transformations described in Definition 44 leave the distribution of ranks of $3 \times r$ *matrices unchanged, except that these integers can be permuted.*

Proof. See [76] for further analysis. □

Theorem 12. *The solution obtained above is not isomorphic to Laderman's Solution (Appendix A.2)*

Proof. Laderman's solution has exactly 6 matrices of rank 3 which occur in products $P1, P3, P6, P10, P11, P14$.

As proved in [76] in the space of solutions of this problem at most 1 matrix will have rank 3. However, the above solution has exactly 2 matrices of rank 3 which occurs in products $P18$ and $P20$. Thus the solutions are not isomorphic. □

Despite the fact that our methodology involves complex encoding procedures of the problem, when this encoding is completed successfully the only thing that does matter is the number of CPUs available for running the SAT solver and more importantly the quality of the SAT solver. A lot of research is carried every year on improving SAT solvers' performance.

6.4.3 Application of Our Method to CC problems

6.4.3.1 Bit-Slice Implementation: PRESENT S-box

We apply our methodology for optimizing the PRESENT S-box given by,

$$\{12, 5, 6, 11, 9, 0, 10, 13, 3, 14, 15, 8, 4, 7, 1, 2\} \text{ [18]}.$$

Using our methodology, we have obtained that its MC is 4. Lemma 4 is a proof of this bound based on SAT solvers. A similar result was obtained for AES S-box by similar methodology [24].

Lemma 4. *The Multiplicative Complexity of the PRESENT S-box is 4 (cf. Figure 6.4).*

Proof. Initially, we encoded the problem using 3 AND gates and all the SAT solvers we have tried output UNSAT. This could be converted to a formal proof that the MC is at least 3. For 4 AND gates, our system outputs SAT and a solution.

Further optimization of the linear part (which is also optimal as we also obtained UNSAT for smaller numbers) allowed us to minimize the number of XORs to the strict minimum possible (proved by additional UNSAT results). As a result, we have obtained an implementation of the PRESENT S-box with 25 gates in total: 4 AND, 20 XOR, 1 NOT.

A better result in terms of gate complexity can be achieved by observing that AND gates and OR gates are affine equivalents since we have that $OR(x, y) = AND(x + 1, y + 1) + 1$. Thus, it is likely that if we implement certain AND gates with OR gates, we might be able to reduce even more the overall complexity of the linear parts. We try all possible 24 cases, where some AND gates are implemented with OR gates.

Starting with the right optimization with MC=4 (as several such optimizations may exist), we obtain the following implementation of the PRESENT S-box, which requires only 14 gates total,

```
    T1=X2^X1; T2=X1&T1; T3=X0^T2; Y3=X3^T3; T2=T1&T3; T1^=Y3;
T2^=X1;   T4=X3|T2; Y2=T1^T4; T2^=~X3; Y0=Y2^T2; T2|=T1; Y1=T3^T2;
```

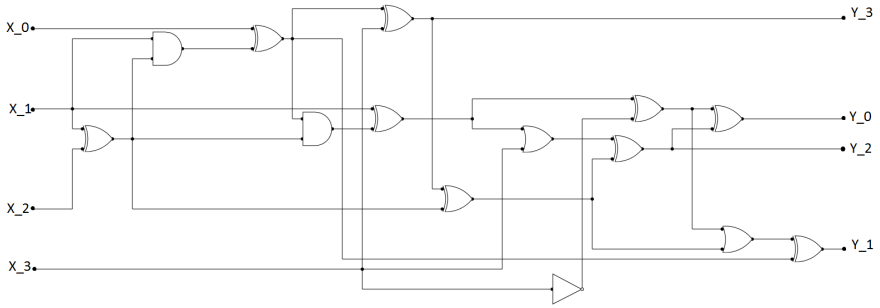

Figure 6.4: PRESENT S-box Best Known Implementation - Our bit-slice type implementation of PRESENT S-box with 14 gates

□

Remark 14. ***Applications:*** *This implementation is used in a recent bit-slice implementation of PRESENT [6]. In addition we postulate that this implementation of the PRESENT S-box is in certain sense optimal for DPA-protected hardware implementations with linear masking, as it minimizes the number of non-linear gates (there are only 4 such gates). One method of protection against DPA is to randomize sensitive data like internal states by XORing them with a randomly generated mask [104].*

6.4.3.2 Multiplicative Complexity (MC): GOST S-boxes

In this section, we apply our automated methodology for constructing optimal circuit representations of the 8 S-boxes S1-S8 of GOST block cipher with respect to their MC.

Lemma 5. *The MC of each of the 8 S-boxes of GOST cipher as specified in OpenSSL is equal to the values given in Table 6.1.*

Table 6.1: MC of the 8 principal GOST S-Boxes

S-box Set	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
TestParamSet	4	5	5	5	5	5	4	5
CryptoProParamSet	4	5	5	4	5	5	4	5
CryptoProParamSetA	5	4	5	4	4	4	5	5
CryptoProParamSetB	5	5	5	5	4	5	5	5
CryptoProParamSetC	5	5	5	5	5	5	5	5
CryptoProParamSetD	5	5	5	5	5	5	5	5
SberbankHashParamset	4	4	4	5	5	4	4	4
ISO	5	5	5	5	5	5	5	5

6.4.3.3 Optimization of CTC2 S-box

We applied same methodology for optimizing the S-box of CTC2 cipher. The S-box of CTC2 cipher is $\{7, 6, 0, 4, 2, 5, 1, 3\}$.

Moreover, this S-box gives 14 fully quadratic equations with 22 terms of the type,

$$\sum \alpha_{ij} x_i x_j + \sum \beta_{ij} y_i y_j + \sum \gamma_{ij} x_i y_j + \sum \delta_i x_i + \sum \epsilon_i y_i + \eta = 0 \tag{6.12}$$

This S-box is a very good example, which can be used to illustrate the provable optimal aspects of our methodology. Using the famous Berkeley's software *Logic Friday* we have obtained a circuit representation with 14 gates in total [105].

Since CTC2 is small enough, we achieved in obtaining optimal circuit representations (and proving that no better can be done) with respect to all circuit complexity metrics we have introduced in the introductory section. The Lemmas below summarize all the results, which can be found also in [42, 44].

Lemma 6. *The MC of CTC S-box is 3 (we allow 3 AND gates and an unlimited number of XOR gates, cf. Figure 6.5)*

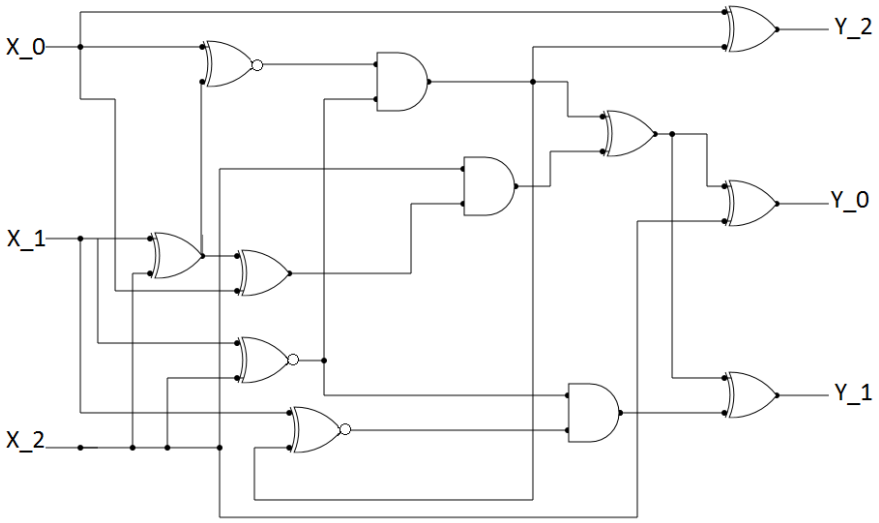

Figure 6.5: MC Optimization: CTC2 - Our provably optimal implementation of CTC2 S-box with MC 3

Lemma 7. *The Bitslice Gate Complexity (BGC) of CTC S-box is 8 (allowed gates are XOR,OR,AND,NOT, cf. Figure 6.6)*

Lemma 8. *The Gate Complexity (GC) of CTC S-box is 6 (allowing XOR,OR,AND,NOT,NAND,NOR,NXOR, cf. Figure 6.7)*

Lemma 9. *The NAND Complexity (NC) of CTC S-box is 12 (only NAND gates and constants, cf. Figure 6.8)*

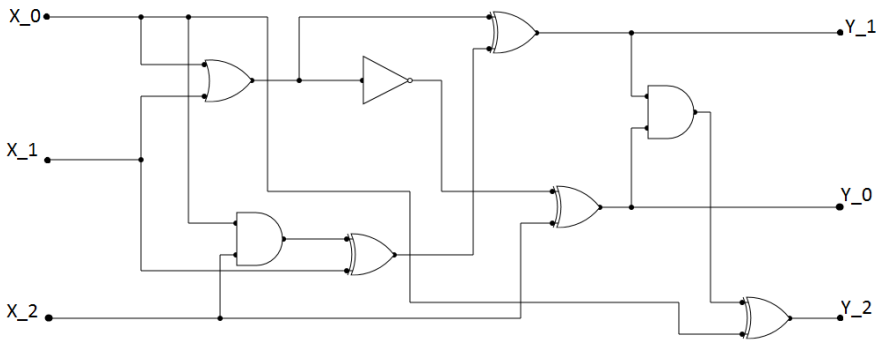

Figure 6.6: Optimal BGC: CTC2 - Our provably optimal implementation of CTC2 S-box with Bitslice Gate Complexity 8

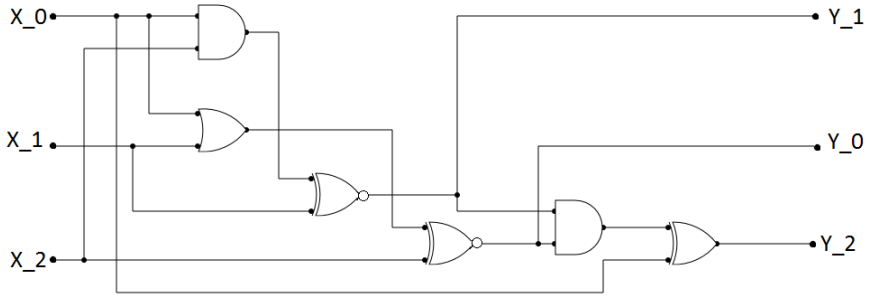

Figure 6.7: Optimal GC: CTC2 - Our provably optimal implementation of CTC2 S-box with Gate Complexity 6.

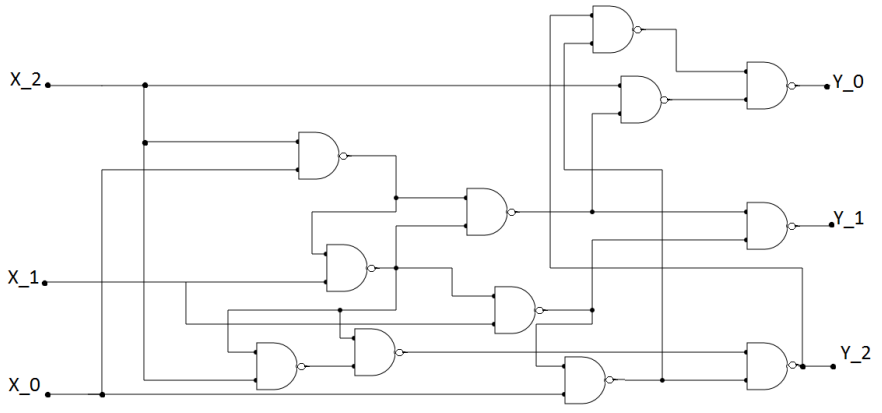

Figure 6.8: Optimal NAND: CTC2 - Our provably optimal implementation of CTC2 S-box with NAND Complexity 12

6.4.3.4 The Majority Function

We also apply our methodology to obtain circuits of optimal MC for the circuit representation of the majority function on small number of inputs. The majority function is of great interest and it is used in many cryptographic applications, such as electronic voting. However, exact bounds on its MC are not known. Using our SAT-solver based methodology we have been able to obtain optimal circuit representations with respect to MC for the majority function with 3, 5 and 7 inputs.

The MC of majority function on 3 inputs was proven to be 1 using our methodology (cf. Figure 6.9). Using same methodology as before, we solved the problem for $MC = 1$ in 5.0s.

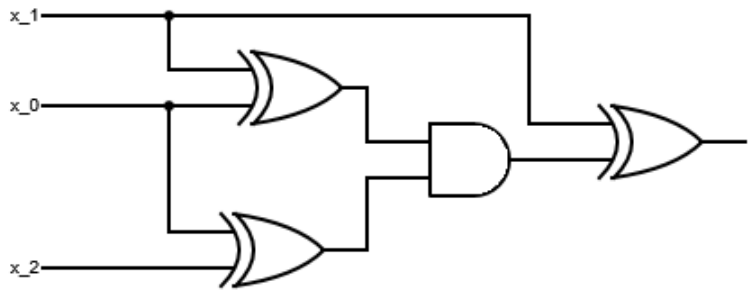

Figure 6.9: Majority function on 3 inputs - Optimal circuit representation of a 3 input Majority function with 1 AND gate

For the case of majority function on 5 inputs, we have obtained that the optimal value for MC is 3. We obtained SAT and the circuit representation shown in Figure 6.10 in 8.1s for $MC = 3$ and UNSAT for $MC < 3$.

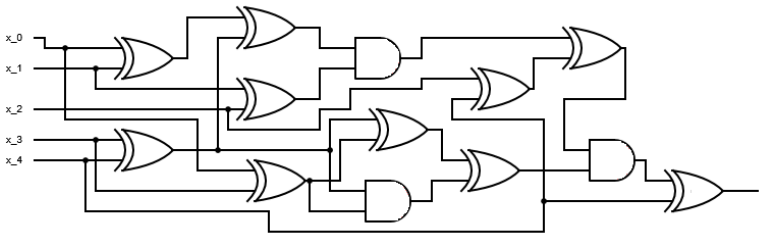

Figure 6.10: Majority function on 5 inputs - Optimal circuit representation of a 5 input Majority function with 3 AND gate

Moreover, we have been able to obtain the optimal representation with respect to

MC for the majority function on up to 7 inputs. The following circuit representation illustrated in Figure 6.11 was obtained with a SAT solver in 16s.

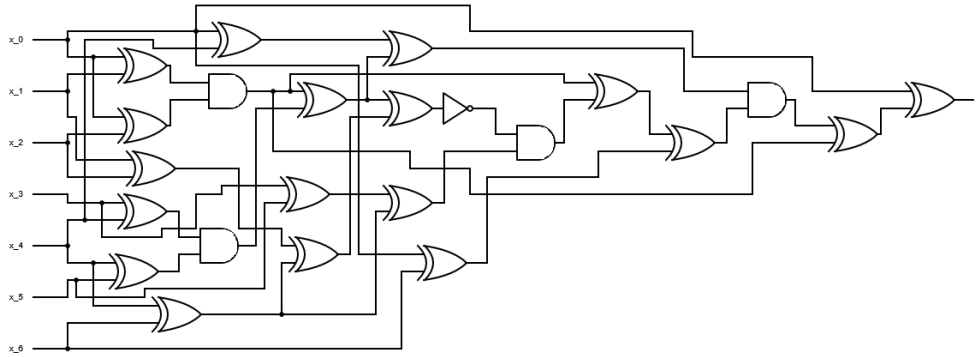

Figure 6.11: Majority function on 7 inputs - Optimal circuit representation of a 7 input Majority function with 4 AND gate

6.5 MC and Cryptanalysis

In general, more efficient circuit representations especially with respect to their MC produce very compact algebraic representations, which are suitable in algebraic cryptanalysis. Using our methodology, we can optimize and obtain optimal representations for sufficiently small circuits. Based on this idea we are able to minimize the number of non-linear gates in a whole cipher possibly to a proven lower bound. We apply our methodology based on optimization of MC to encode GOST cipher, as follows:

1. We write all the equations in their Algebraic Normal Form (ANF).
2. For the S-boxes we use the representations obtained by our method which are optimized with respect to MC. For the modular 2^{32} addition we use the algebraic description studied in Chapter 2.
3. For each input of each each multiplication (AND gate) we add one new variable. All the other gates which are XORs and NORs their corresponding ANFs give linear equations over $\mathbb{F}_2$.
4. The above steps result in a system of quadratic multivariate equations over $\mathbb{F}_2$ which describe the whole cipher.

Next we convert the above system of multivariate quadratic equations over $\mathbb{F}_2$ to a CNF-SAT problem using the Courtois-Bard-Jefferson software [11]. Finally, we attempt to solve this system using the well-known open-source SAT solver MiniSat 2.06 for random choices of key and assuming some plaintext-ciphertext pairs are available. We derived the key using MiniSat 2.06 all random instances we have tried in approximately 10 seconds on a modern CPU.

Remark 15. *Being able to solve for the key for small number of rounds is very important since in many attacks there are always some steps which reduce the problem of attacking the full cipher to a problem of attacking a reduced version of the same cipher. Such reductions can be done for example using fixed points or reflection properties, depending on the structure of the Key Schedule of each algorithm. This idea is summarized in [34] and called "Algebraic Complexity Reduction".*

Based on our optimization techniques the following facts were obtained and found in (Table 1, page 25, [34]) and [37, 38].

Fact 1. *Given 2 plaintext-ciphertext pairs for 4 rounds of GOST, we can recover the 128-bit key in time equivalent to* 2^{24} *GOST encryptions and with very limited memory requirements.*

Similarly, using the guess-then-determine strategy the following result is obtained by Courtois, which works also for random sets of S-boxes [34].

Fact 2. *Given 3 plaintext-ciphertext pairs for 6 rounds of GOST, we can recover the 192-bit key in time equivalent to* 2^{56} *GOST encryptions and with very limited memory requirements.*

The hard problem in such strategies is to determine which bits to guess. Bits which are repeated frequently are preferred as they lead to substantial improvements in the resulting system. For example, in DES, the best choice is to fix the first 20 variables and determine the others [39]. This is because such variables repeat quite uniformly at random inside the algorithm.

As we have mentioned, algebraic attacks are based on solving for the key a large sparse system of multivariate equations using only a few plaintext-ciphertext pairs. At the moment, there is no algebraic attack that can break a full cipher directly. In this section we have (heuristically) observed that MC reductions could speed up algebraic attacks and might be able to increase the number of rounds we can break. However, we do not have strong evidence that this hold in general and it is a speculation arising from our experimentation with algebraic attacks using MC reduction as preprocessing and applied particularly to GOST block cipher.

Part III

Advanced Differential Cryptanalysis

7

Differential Cryptanalysis and Gaussian Distributions

In cryptography, we often study the problem of distinguishing distributions. One distribution that describes the number of certain events which occur at random and another that describes the same variable but due to propagation inside the cipher. In this section, we study this *hypothesis testing problem* applied to DC. The observed variable is the expected number of pairs which have a particular output difference in a set ΔY after some number of rounds, given that they satisfy some input difference in a set ΔX.

This can be realized as a *hypothesis testing problem.* Suppose that a source is used to generate independent random samples in some given finite set with some distribution $\mathcal{P}$, which is either $\mathcal{P} = \mathcal{P}_0$ or $\mathcal{P} = \mathcal{P}_1$. The task of the attacker (or the *distinguisher*) is to determine which one is most likely the one which was used to generate the sample. Thus, we either have a *null hypothesis* $H_0 : \mathcal{P} = \mathcal{P}_0$ or an *alternative hypothesis* $H_1 : \mathcal{P} = \mathcal{P}_1$.
.

7.1 Gaussian Distributions as a Tool in Cryptanalysis

Let $R_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$ denote the round function of an iterated block cipher with round-key k. We assume that each round key k is applied via the group operation $\otimes$ at the input of each round. Additionally, suppose that a block cipher $E_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$ consists of r iterations of the round function R_k, where $K = (k_0, k_1, ..., k_{r-1})$ is the vector consisting of all round keys. Then, the functional decomposition of E_K is given by

$$E_K(x) = R_{k_{r-1}} \circ R_{k_{r-1}} \circ \cdots \circ R_{k_0}(x).$$

For a fixed choice of K, an input difference $\alpha \in A$ (where A is a set of size $|A|$) and an output difference $\beta \in B$ (where B is a set of size $|B|$), the probability $P_{E_K}(A, B)$ of a differential of the form $(\alpha \in A) \rightarrow (\beta \in B)$ is defined by,

$$P_{E_K}(A, B) = \frac{1}{|A|} \sum_{\alpha \in A} P(E_K(x) \oplus E_K(x \oplus \alpha) \in B).$$

for randomly uniformly chosen x.

Given N pairs of inputs (x_i, x_i') such that their difference lies in the set A, we denote as $\mathcal{E}_{E_K}^{(N)}(A, B)$ the number of pairs following the given differential of the form $(\alpha \in A) \rightarrow (\beta \in B)$.

The expected value of $\mathcal{E}_{E_K}^{(N)}(A, B)$ is $N.P_{E_K}(A, B)$ and we are interested in the distribution of $\mathcal{E}_{E_K}^{(N)}(A, B)$. In the rest of this section we use N to be $2^{n-1} \times |A|$, where n is the block size of the cipher of our interest and $|A|$ is the size of the set of input differences. Note that for $|A| > 2$ this corresponds to all possible samples for a given key.

Computing this probability over the right key is not feasible as the key is unknown during the attack. However, as we have already mentioned in Chapter 3, if the output difference in round r only depends on the output difference one round before and not on the individual inputs x_i, x_i', then the cipher is said to have the *Markov property* and the average value of

$$\widetilde{P_{E_K}}(A, B)$$

over all possible keys is given by

$$\widetilde{P}_E(A, B) = \frac{1}{\#K} \sum_K (P_{E_K}(A, B))$$

According to *hypothesis of stochastic equivalence* [88, 7, 86], we have

$$P_{E_K}(A, B) = \widetilde{P}_E(A, B)$$

for almost all keys K and hence the expected number of $\mathcal{E}_{E_K}^{(N)}(A, B)$ is given by $N.\widetilde{P}_E(A, B)$. This simplifies our model a lot, since in an attack we do not know the key but according to this assumption we can compute the expected number of events by computing it on average over all the keys.

For our purpose, we are interested in the distribution of $\mathcal{E}_{E_K}^{(N)}(A,B)$. This was analyzed in [57, 7] and it turns out that considering $\mathcal{E}_{E_K}^{(N)}(A,B)$ as the result of N independent Bernoulli trials with success probability $\widetilde{P}_E(A,B)$ is a reasonable approximation for the distribution of $\mathcal{E}_{E_K}^{(N)}(A,B)$.

Assumption 2. *(cf. Theorem 14 in [57])*
The random variable $\mathcal{E}_{E_K}^{(N)}(A,B)$ follows a Binomial distribution $B(N,\widetilde{P}_E(A,B))$, that is

$$P\left(\mathcal{E}_{E_K}^{(N)}(A,B)=c\right)=\binom{N}{c}\widetilde{P}_E(A,B)^c\left(1-\widetilde{P}_E(A,B)\right)^{N-c}$$

for a fixed value c and over random keys K.

On the other hand, according to the *wrong-key randomization* hypothesis (cf. Assumption 3), for a wrong-key guess we expect the variable $\mathcal{E}_{E_K}^{(N)}(A,B)$ to be distributed as follows.

Assumption 3. *(Wrong-Key Randomization Hypothesis)*
For $K' \neq K$

$$\mathcal{E}_{E_{K'}}^{(N)}(\alpha,\beta)\sim B(N,q),$$

where q a constant probability depending on the size of the input and output sets of differences and K is the right key.

In standard DC, we assume that the counter follows the distribution $B(N,p)$ for the right key, while it follows the distribution $B(N,q)$ for the wrong keys. The aim is to distinguish these two distributions.

Moreover, if the term Np is large enough, then a Binomial distribution given by $B(N,p)$ can be approximated by a Poisson distribution $Pois(\lambda)$, with mean and variance equal to λ, where $\lambda = Np$. A rule of thumb for such approximation is that $N \geq 20$ and $p \leq 2^{-4.3}$ or $N \geq 100$ and $Np \leq 10$ [98]. Thus, Poisson distribution turns out to be a good approximation to the distribution of the number of events of our interest.

Considering n independent observations $X_i = \mathcal{E}_{E_K}^{(N)}(A,B) \sim Pois(\lambda)$, the sample mean

$$\overline{S}_n = \frac{X_1 + X_2 + ... + X_n}{n} \tag{7.1}$$

has (approximately) Gaussian Distribution due to Central Limit Theorem (CLT) (cf. Theorem 13) given by $\mathcal{N}(\lambda,\lambda)$.

Theorem 13. *(Central Limit Theorem)*

Let $\{X_1, .., X_n\}$ *a sequence of independent and identically distributed random variables sampled randomly from distributions of expected values* μ *and finite variances* σ^2*. Let* $\overline{S}_n = \frac{X_1+..+X_n}{n}$*. Then, the following limit holds,*

$$\lim_{n\to\infty} P(\sqrt{n}.(\overline{S}_n - \mu) \leq \sigma.x) = \Phi(x),$$

where $\Phi(x)$ *is the probability that a standard normal variable is less than* x

This theorem is very useful in cryptanalysis as we will see in the next section.

7.2 Distinguishers in Differential Cryptanalysis

Let $\mathcal{R}$ and $\mathcal{W}$ denote two random variables with expected mean and variance $(E(R), V(R))$ and $(E(W), V(W))$ respectively which are described by Gaussian distributions.

In DC, we assume that $\mathcal{R}$ is the expected number of specific events that occur for the cipher that we study, while $\mathcal{W}$ is the expected number of such events in case of a random permutation. We assume that the right key value corresponds to $\mathcal{R}$ and if the key is wrong then the variable of the number of events has distribution according to $\mathcal{W}$. The validity of this assumption is not granted, however it is a very common assumption in cryptography. In the rest of this section, given independent samples either from distribution $\mathcal{W}$ or $\mathcal{R}$, we study the following hypothesis testing problem.

$$H_0 : P = \mathcal{W}$$

$$H_1 : P = \mathcal{R},$$

where $\mathcal{W}$ corresponds to the wrong key and $\mathcal{R}$ to the right key. The probability density function of $\mathcal{W}$ is given by

$$f_{\mathcal{W}}(x) = \frac{1}{\sqrt{2\pi Var(W)}} \exp^{-\frac{1}{2Var(W)}(x-E(W))^2} \tag{7.2}$$

Figure 7.1 represents the probability distributions of the two Gaussian distributions, one corresponding to the wrong key and one corresponding to the right key. The variable x denotes the number of pairs satisfying a given differential. In our statistical hypothesis we set a threshold $E(R)$ and we make conclusions based on the distance of the observed variable from this mean. If x exceeds $E(R)$ then we assume that the

sample data we observe corresponds to the Gaussian distribution which corresponds to the right key. However, there is a probability that we reject H_0, while it is actually true (**Type I error**, red-shaded region Figure 7.1). In our applications this means accepting a wrong key as correct and this probability is given by,

$$P(\mathcal{W} \geq E(R)) = \int_{E(R)}^{\infty} f_{\mathcal{W}}(x)dx = \frac{1}{2}(1 - erf(\frac{E(R)-E(W)}{\sqrt{2Var(W)}})),$$

where $erf(x)$ is the Gaussian error function given by $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp^{-t^2} dt$.

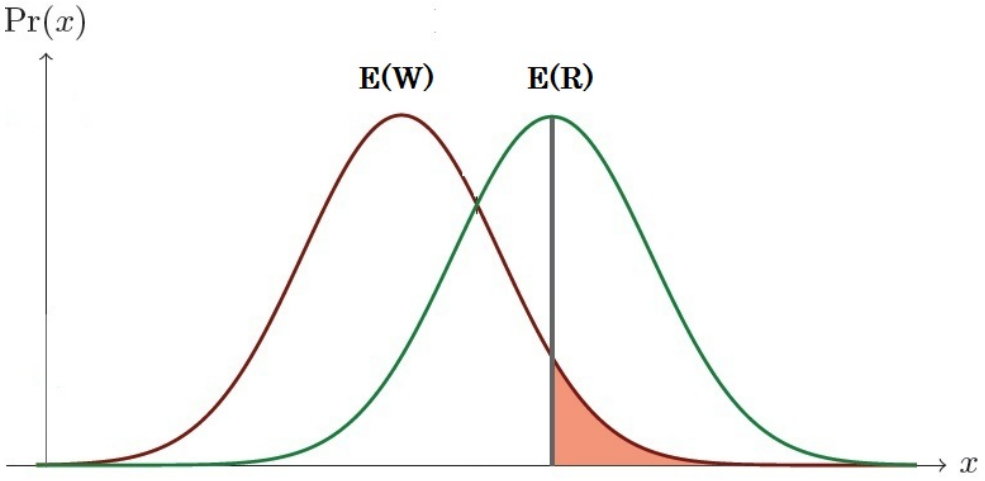

Figure 7.1: Computation of Advantage - Red-Shaded area represents the probability of Type I error

Moreover, there is also the possibility of accepting H_0, while it is false (**Type II error**) and this corresponds to right key rejection. This is fixed to $\frac{1}{2}$ as we see from Figure 7.1. We apply this methodology to mount attacks against the full GOST cipher which we describe in the following chapters.

8

Cryptanalysis of GOST Block Cipher

GOST is a block cipher that encrypts 64-bit blocks using a 256-bit secret key. In spite of considerable cryptanalytic effort over the past 20 years, no key recovery attack against the full cipher (faster than brute-force) was discovered, without any additional key assumptions and relaxations such as *related* keys.

Moreover, its small block size and the very simple key schedule influence the size of the required circuit size and thus make it suitable to be implemented even in very small devices. As a result of these, GOST became an Internet standard implemented in many libraries and it was submitted to ISO 18033-3 to be an international standard.

Immediately after the submission, many attacks were discovered [75, 35]. In the following chapters, we describe advanced differential attacks against many variants of GOST. Our attacks are truncated differential attacks of Depth-First search style based on the construction of a statistical distinguisher. In this chapter, we describe a methodology for the construction of distinguishers based on our notion of general open sets [50]. These sets can be seen as a refinement of Knudsen's truncated differentials and are based on the connections between S-boxes from round to round in GOST. The exploration of this space yields differentials that propagate with sufficiently good probability and can be composed efficiently resulting in the construction of distinguishers with sufficiently large advantage.

8.1 Brute-Force Attack on 256-bit GOST Keys

Brute-force attack is a non-trivial attack when the length of the key exceeds the size of the block since many false positives are expected when trying to recover the key. For example in GOST, given one (P, C) pair, we expect that $2^{256-64} = 2^{192}$ keys (out of the total 2^{256}) will satisfy $E_k(P) = C$. We can apply brute-force attack in GOST using the Depth-First search approach as follows.

Given a pair (P_1, C_1), we start testing keys $k \in K$ if they satisfy $E_k(P_1) = C_1$. During this stage, we discard a key k if it does not satisfy this relation and try a different key, otherwise we keep testing the same key k by requesting a new pair (P_2, C_2). Given this new pair, we check again if k satisfies $E_k(P_2) = C_2$. If the answer is positive we request another distinct pair, otherwise we discard it and go again to the first stage of the attack. The first pair will reduce the space to 2^{192} possible key candidates, the second one to 2^{128} and the third one to 2^{64}. Using the first pair, we expect that we will go through 2^{255} encryptions on average, then with the second pair we will go through 2^{127} encryptions on average and finally with the third one we will go through 2^{63} on average. Finally, a fourth pair is used to determine the key. Figure 8.1 illustrates this technique.

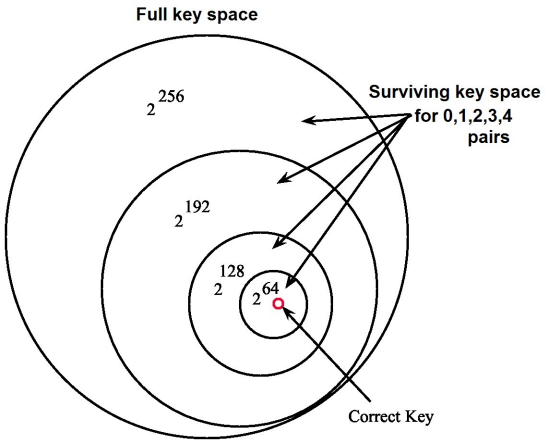

Figure 8.1: Brute Force Attack on GOST - Exhaustive-search against GOST

The expected (average) for the total time complexity in terms of GOST encryptions is given by $2^{255} + 2^{191} + 2^{127} + 2^{63} + 1 \simeq 2^{255}$.

8.2 Existing Attacks on Full GOST

Gabidulin *et al* were the first to conduct a basic assessment of the security of GOST against linear and differential cryptanalysis [114, 115]. As they claim, 5 rounds are sufficient to secure GOST against LC at the security level of 2^{256}, while only 6 are enough even if the S-boxes are replaced by the Identity map. Additionally, they claim that 7 rounds are sufficient for a 128-bit security level against naive DC.

Before the submission to ISO, no attack which was disputing the 256-bit level security was known. In the same year of the submission, many attacks faster than brute force were developed; we now have reflection attacks, attacks based on double reflections, related-key attacks and advanced differential attacks [111, 62, 48, 35, 34, 36].

Ten years earlier, the Japanese researchers Seki and Kaneko developed an attack on 13 rounds of GOST using 2^{51} chosen plaintexts based on truncated differentials [111]. The notion of truncated differentials (partitioning type) allows us to reduce the influence of the round keys on the transitional probabilities and it thus simplifies a lot the analysis. In the same paper, they have proved that naive DC always fails in GOST. This is because propagation of single differences for one round occurs with very low probability for the majority of the keys and as the number of rounds increases we expect this probability to vanish for most keys.

Isobe presented at FSE 2011 the first *single-key attack* against the full 32 rounds by developing a new attack framework called *Reflection-Meet-in-the-Middle* (RMITM) attack [75]. His method combines techniques of the reflection and the Meet-in-the-Middle attack in an optimized way. This attack has time complexity 2^{225} GOST encryptions and requires 2^{32} known plaintexts.

In parallel, many other attacks based on different frameworks were developed. Courtois presented attacks based on the notion of *algebraic complexity reduction*, which allows one to reduce the problem of attacking the full cipher to a problem of attacking a reduced version of the same cipher [34]. This reduction takes into account many algebraic and structural properties of the cipher, such as the weak key schedule and the poor diffusion for limited number of rounds. It makes use of software such as SAT solvers at the final solving stage for solving for they key a system that describes a reduced version of the cipher [35, 34, 47].

In addition, *advanced differential attacks* were developed and successfully applied against the full block cipher. The first differential attack against full 32-rounds of GOST was developed by Courtois and Misztal [46]. The most complex task involved in this attack is the construction of a 20-round distinguisher. By the same year, an improved differential attack with complexity 2^{179} GOST encryptions was presented by Courtois [36].

Furthermore, Courtois studied the *multiple-key* scenario, where (P, C) pairs from randomly selected keys are available. This scenario is very realistic, as in real-life applications we expect encryptions with random keys rather than a fixed key. He proved that one such key can be revealed in approximately 2^{101} encryptions, provided that approximately 2^{32} pairs are available for each key.

Table 8.1 summarizes the state-of-art regarding cryptanalysis of full GOST for both single and multiple key scenarios. The reference point for the time complexity is the number of GOST encryptions required.

Table 8.1: State-of-art in cryptanalysis of GOST

Author	**Type**	**Time**	**Data**	**Scenario**
Isobe [75]	RMITM	2^{224}	2^{64}	single key
Dinur *et al* [62]	2DMITM,Fixed Points	2^{192}	2^{64}	single key
Courtois [34]	2DMITM,Fixed Points	2^{191}	2^{64}	single key
Courtois [36]	Differential	2^{179}	2^{64}	single key
Kara *et al* [79]	2DMITM	2^{130}	2^{64}	mutiple key
Courtois [34]	Algebraic-Differential	2^{101}	2^{32} per key	mutiple key

All the attacks presented so far are based on the most popular implementation of GOST, which uses the set of S-boxes *GostR3411-94-TestParamSet*. There was no attempt so far to find an attack against any other variant of GOST and no attempt to provide a general methodology which would work in all cases. The first who introduced such a method are Courtois and Mourouzis [50]. They introduced the fundamental notion of *general open sets*, which are special forms of sets of differentials dictated by the structure of GOST. This notion allows one to explore efficiently this space and obtain surprisingly good truncated differential properties which can be used in some

cases to mount differential attacks against the full cipher. We introduce this notion in the next section.

8.3 General Open Sets

In this section, we introduce a new type of sets of differences, which we name *general open sets*. They can be seen as a refinement of Knudsen's truncated differentials [84]. The main difficulty in attacks using truncated differentials is the exploration of the exponentially large space of possible sets of differences and how to discover interesting truncated differential properties.

However, if we consider special sets which are dictated by the structure of the encryption algorithm that might allows us to explore this subspace and discover interesting properties. We follow this idea in the case of GOST and we consider some special sets, which we name *general open sets* (cf. Definition 49). These sets are constructed based on the connections between the S-boxes from round to round.

Definition 49. *(General Open Sets, [50])*

We define a General Open Set X as a set of differences on 64 bits with additional constraints as follows. A General Open Set is represented by a string Q of 16 characters on the alphabet $\{0, 7, 8, F\}$ in the following way:

1. Differences in X are "under" Q, by which we mean that for all $x \in X$ $Sup(x) \subseteq Sup(Q)$, where $Sup(x)$ is the set of bits at 1 in x, $Sup(x) \subset \{0, 1, ..., 63\}$.

2. In each of the up to 16 non-zero characters in string Q which may be any of 7,8,F, there is at least one "active" bit at 1 in x for all $x \in X$.

3. In the case of F the most significant bit is always active for each $x \in X$ and for each position in Q which is at F.

The main reason why we have this very special alphabet $\{0, 7, 8, F\}$ is the internal connections of the GOST cipher and in particular the 11-bit rotation to the left after the substitution layer. Informally, we can say that we group together bits which are likely to be flipped together. F is used to make the sets disjoint such that each difference x belongs to only one general open set.

Given a general open set represented by the string Q, we define the closure of this set as in Definition 50.

Definition 50. *(Closure of Differential Sets)*

The closure of a differential set Q is denoted by $[Q]$ and it is the set that contains all the differences that are under the string Q with the only rule to exclude the zero difference on the 64 bits. A set P such that $P = [Q]$ will be called a closed set.

Remark 16. *It is possible to see that closed sets corresponds to truncated differentials as defined by Knudsen and open sets corresponds to a particular way of partitioning truncated differential sets (or closed sets which we have defined in Definition 50)*

Consider the general open set represented by 8070070080700700*. Then, this sets contains in total* $(2^3-1)^4$*. The closure of* Q*, denoted by* $[Q]$*, contains* $2^{14}-1$ *elements. This is because due to Definitions 49 and 50, in general open set* 8 *can be only* 1000 *and* 7 *any difference in the set* $\{0111, 0100, 0010, 0001, 0110, 0101, 0011\}$*, while in case of a closed set* 8 *can be any element in the set* $\{0000, 1000\}$ *and the character* 7 *can be any element in the set* $\{0000, 0111, 0100, 0010, 0001, 0110, 0101, 0011\}$ *provided that the zero difference on 64-bits is excluded.*

For example, in Figure 8.2, we illustrate the connections for the study of truncated differential or closed set [8070070080700700]*. We observe that the 3 least significant bits from* S_3 *are entering* S_6 *and this is denoted by* 7 *in the differential, while the most significant bit from* S_6 *is entering* S_8 *and this is denoted by* 8*.*

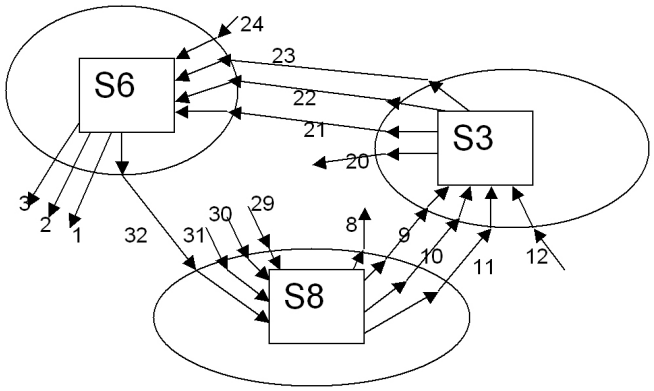

Figure 8.2: S-boxes Connections in GOST cipher - Connections between S-boxes for the study of the truncated differential [8070070080700700]

It is a non-trivial task to define such sets in general since some heuristics suggested by the structure of the algorithm need to be discovered. Note that the same idea can be applied to any cipher. In the next section, we study the diffusion inside GOST aiming to illustrate that in particular for the first 8 rounds the diffusion is really poor.

8.3.1 Propagation of Differentials

In this subsection, we study diffusion in GOST up to 8 rounds. In particular, we study propagation of two different sets of differences. Firstly, we study the propagation of a one-bit difference of the form 8000000000000000 *and then we study the propagation of a closed set with 14 active bits of the form* [8070070080700700]*. The aim is to show that in GOST we have very poor diffusion, especially in the first 8 rounds and that differences lie in some specific sets with a very high probability. In our simulations we used the most popular version of GOST, which uses the set of S-boxes "GostR3411-94-TestParamSet".*

Remark 17. *In case of the special set* [8070070080700700] *(cf. Definition 50), we denote it for abbreviation as* [877877] *and we consider all* $2^6 - 1$ *underlying general open sets which are explicitly enumerated in Figure 8.3. For example,* 870877 *or also denoted as* 87_877 *in one subset of the closed set which is general open set.*

In order to compute the probability of a transition we use the simple Algorithm 11. We assume that the distribution of the number of events of our interest follows (approximately) a Poisson distribution (cf. Chapter 7). We use this distribution as we have experimentally observed that for all cases we have tried,

- *We have a discrete distribution of small integers*
- *In all cases we have tried and are included in this thesis the variance is relatively close to the mean.*

Thus, for a sample of size N *if* x *denotes the number of events that were observed (approximated by Poisson with parameter Poisson mean* Np *where* p *is the true mean), then the approximated Standard Deviation (SD') of the variable* $\frac{x}{N}$*, where* N *is assumed to be constant and* p' *the observed mean, is given by* $\frac{\sqrt{Np'}}{N} = \sqrt{\frac{p'}{N}}$*. This is because the variance equals to the mean in case of a Poisson distribution.*

Denote by p' *the approximated mean, then* $SD' = \sqrt{\frac{p'}{N}}$*. Then, for example with about* 99% *confidence interval, the true mean is within* $\pm 3.\sqrt{\frac{p'}{N}}$ *of the observed mean.*

Let I_1 *be the interval* $[p' - t\sqrt{\frac{p'}{N}}, p' + t\sqrt{\frac{p'}{N}}]$*. In our simulations we would like this interval* I_1 *to be contained in the interval* $I_2 = [p'.2^{-a}, p'.2^a]$*, where* a *is an error we allow in the exponent of the mean as a power of 2.*

We assume that the true mean that we are aiming to approximate by simulations is bigger than some probability value p_0 *(for example like* $2^{-26.0}$*) in order to ensure that*

Algorithm 11 terminates in reasonable time. The inclusion of sets $I_1 \subseteq I_2$ *implies that we need to run about* $N > N_0$ *simulations, where* N_0 *is given by*

$$N_0 = \left(\frac{(2^a)t}{(2^a - 1)}\right)^2 . \frac{1}{p_0} \tag{8.1}$$

in order to compute an approximated mean with the desired precision. For smaller values of probabilities we need to use different values for parameters a, t *such that it is computationally feasible to run beyond this bound. Most of the later results which are less than approximately* $2^{-26.0}$ *are inexact results and were taken by setting* $a = 0.3$ *and* $t = 5$ *in most cases.*

On an Intel i7 1.73 GHz PC with 4.00 GB RAM computer, we can run around 2^{22} *full GOST encryptions per second per CPU. For probabilities above* $2^{-26.0}$ *we set* $t = 3$ *and* $a = 0.1$*, while for smaller probabilities we allow* a *to be around 0.3 or even higher and it thus the results are inexact. In Table 8.2 we present the time taken to compute some probabilities that are presented in this thesis with some precision* $a = 0.1$ *and* $t = 3$*.*

Table 8.2: Time taken to compute the mean of a Poisson process for $a = 1$

Probability	**Number of Rounds**	**Time Taken**
$2^{-13.8}$	4	2 seconds
$2^{-16.5}$	6	15 seconds
$2^{-24.0}$	8	2.3 hours
$2^{-24.0}$	10	2.8 hours
$2^{-25.0}$	8	2.3 hours
$2^{-25.0}$	10	2.8 hours

Algorithm 11 Measuring the limit of a transitional probability up to some precision

Input: Sets of differences A, B, r the number of rounds

1. Initiate counter T and set it at 0
2. Let N be a large integer (around N_0 is the bound derived before)
3. For all $1 \leq i \leq N$, Do

3a. Randomly generate a plaintext P_i and a key K_i

3b. Select at random a single difference $\alpha \in A$

3c. Compute $P_i' = P_i \oplus \alpha$

3d. Compute C_i, C_i', the encryptions of P_i, P_i' after r rounds

3e. If $C_i \oplus C_i' \in B$ increase T by 1

4. Return $-\log_2(T/N)$

8.3.1.1 Case Study A: 8000000000000000

We study the propagation of the 1-bit input difference 8000000000000000 *over the first 8 rounds. We compute by simulations the probability of the output difference to lie in some general open sets within the closed set* [8070070080700700]. *This closed set contains* $2^6 - 1$ *disjoint general open sets, excluding the zero differential.*

Using computer simulations we have computed the probability of a transition with input difference 8000000000000000 *and an output difference in any of the 64-1 disjoint open sets over random plaintexts and keys. The 64-1 boxes in Figure 8.3 represent the 64-1 non-empty disjoint classes and the way they are ordered is not for any particular reason. The width of each box is proportional to the probability (using a logarithmic scale) for the output differential to lie within a specific general open set.*

From Figure 8.3 we observe that the output difference of the resulting encrypted plaintexts after 1 round is 00000000800000000 *with probability 1. This, allows to gain one round in an attack, as we know exactly the previous state. For the first 5 rounds, we observe that the diffusion is very poor since the output difference is still contained within the set* [8070070080700700] *with very high probability.*

However, after 8 rounds the diffusion is improved and the probability distribution of the output differences will tend to a uniform distribution. From Figure 8.4 and Table 8.4, we observe that the entropy of the distribution of the output difference is initially low as expected for small number of rounds and then it increases reaching the value of 12.31 *(approximately) after 7 rounds. However, for 8 rounds and more we expect the entropy to be close to 14 as the number of active bits in the truncated differential we study.*

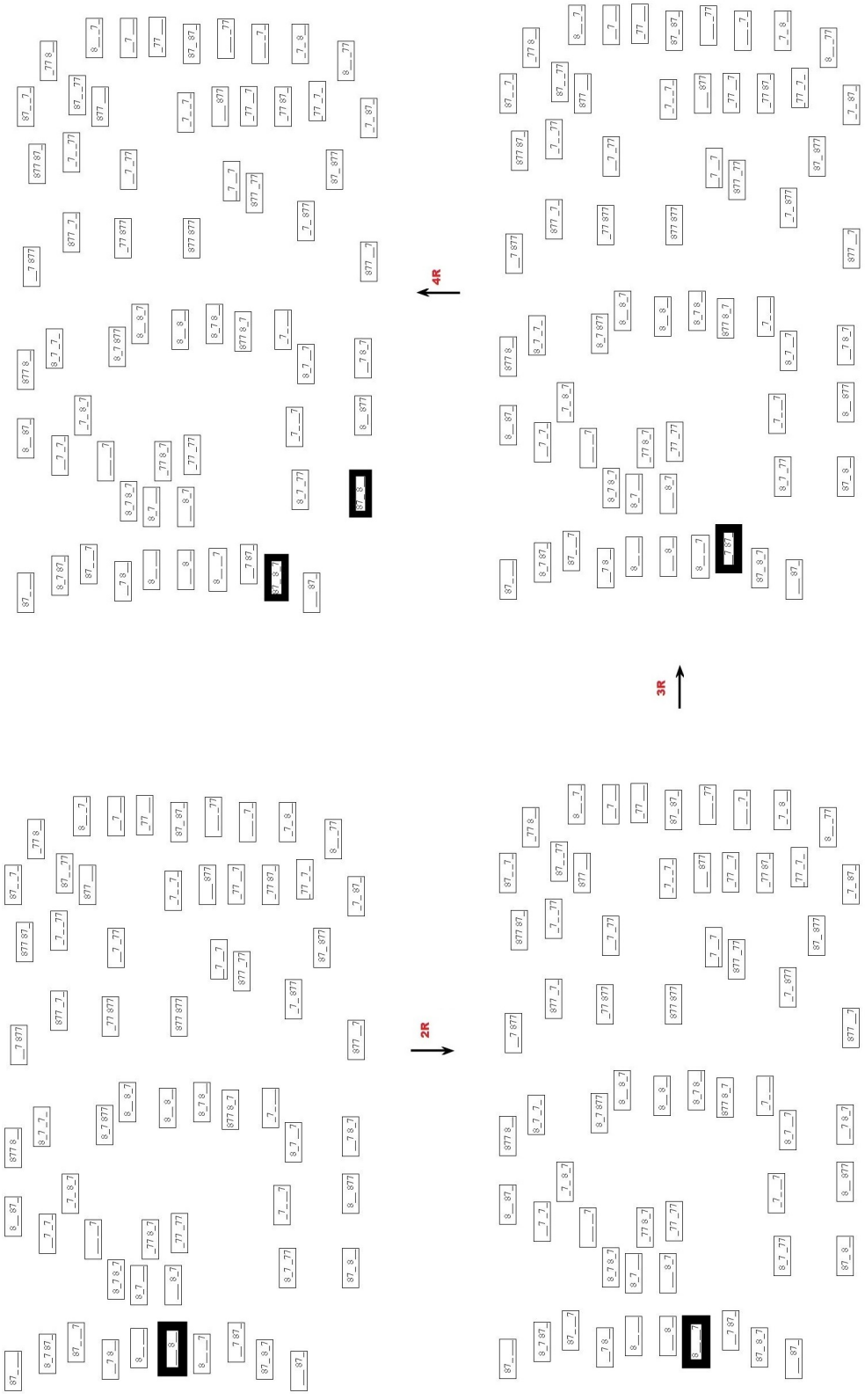

Figure 8.3: Propagation in GOST - Propagation of 8000000000000000 for 1R-4R

Table 8.3: Entropy's estimation after 7 rounds in GOST

Round	Entropy
0	0.0
1	0.0
2	2.81
3	5.61
4	5.72
5	8.19
6	10.92
7	12.31

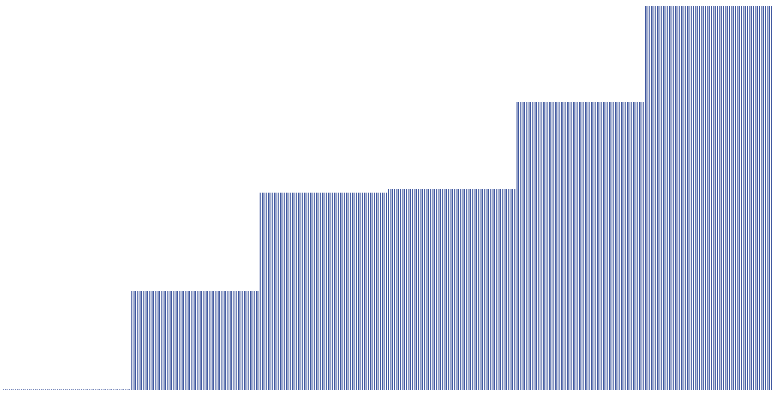

Figure 8.4: Graph of Entropy for GOST - The entropy of the distribution of the output difference with input difference 8000000000000000 against first 7 rounds of GOST [50] (cf. Table 8.4)

8.3.1.2 Case Study B: [8070070080700700]

We study the propagation of the closed set [8070070080700700] *through different number of rounds of GOST. As before, we are interested in output differences which lie within the mask of the truncated differential* [8070070080700700]. *In Figure 8.5, 8.6 and 8.7, we illustrate the propagation of this difference after 1,4 and 8 rounds respectively.*

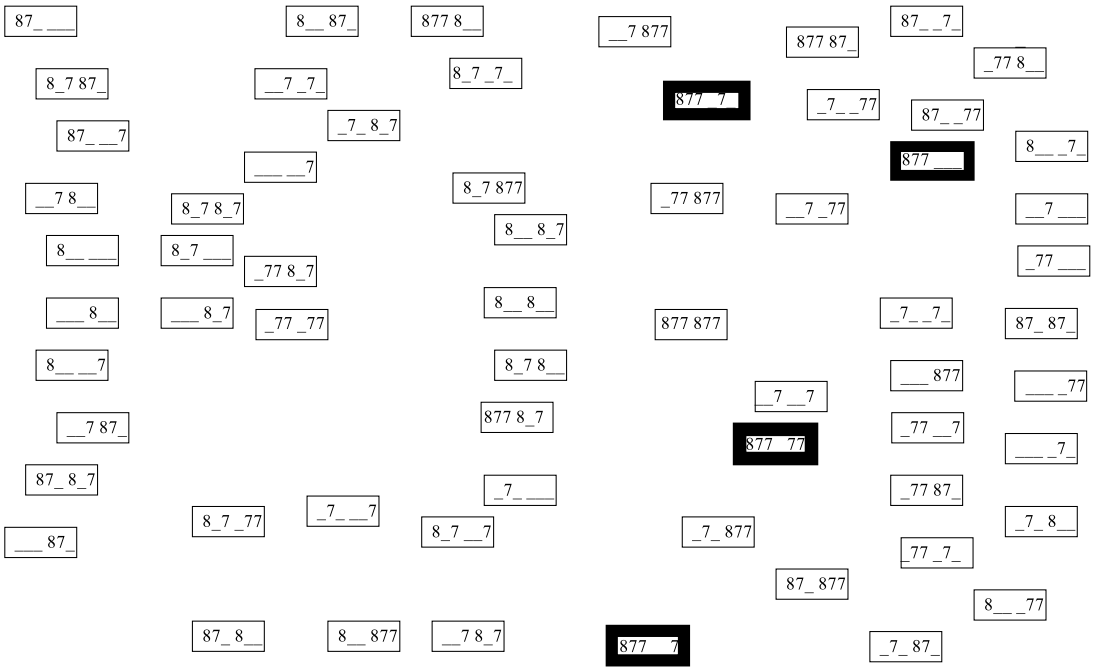

Figure 8.5: Study of Propagation in GOST(1R) - Propagation of [8070070080700700] after 1R of GOST with *GostR3411-94-TestParamSet* set of S-boxes

In this case, diffusion is much stronger even after 4 rounds of GOST as expected. After 8 rounds, we have that all the differences which lie in [8070070080700700] *are uniformly distributed as expected (Figure 8.7).*

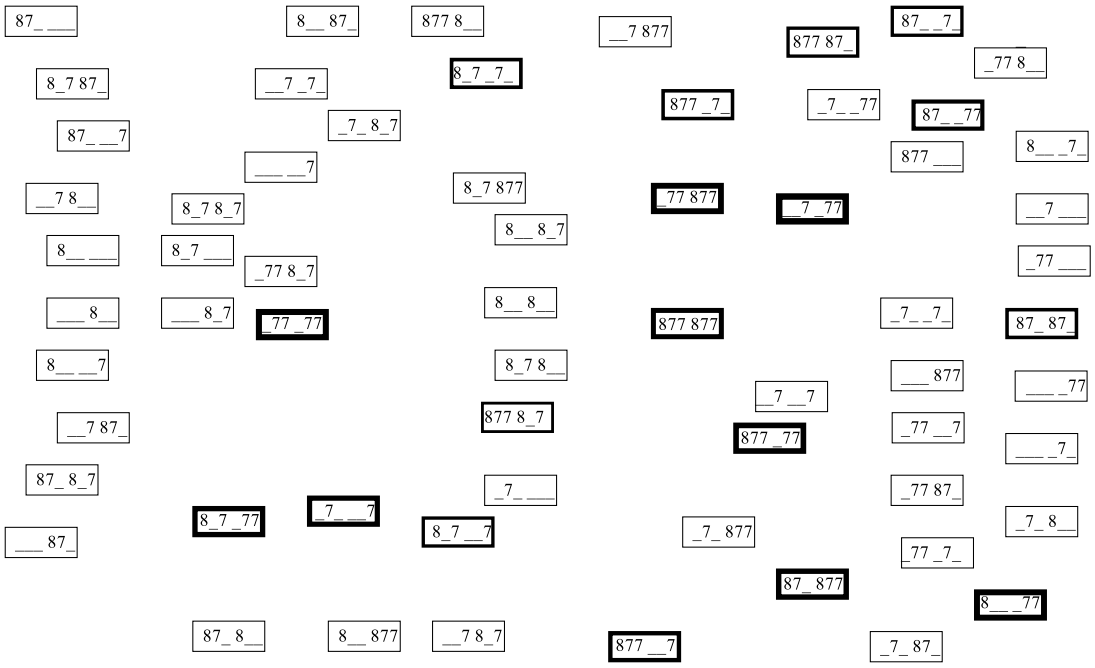

Figure 8.6: Study of Propagation in GOST (4R) - Propagation of [8070070080700700] after 4R of GOST with *GostR3411-94-TestParamSet* set of S-boxes

8.4 Truncated Differentials and Markov Ciphers

The insertion of the round-key via modulo 2^{32} addition, makes GOST deviate from the classical notion of Markov ciphers. The study of propagation of differences in a non-Markov cipher is much more complex and difficult, since there are no adequate mathematical methods accounting for the structure of such ciphers.

Additionally, the hypothesis of stochastic equivalence does not hold in a non-Markov cipher and what we really aim to do is to consider DC in special sets of differences such that Markov cipher theory is directly applied and give sufficiently good results up to a small margin of error. This is very important from a cryptanalytic point of view. As we will see GOST behaves approximately as a Markov cipher if we consider particular sets of differences. To recall, an iterated block cipher with round function $Y = f(X, Z)$ is a Markov cipher if

$$P_Z(\Delta Y = \beta | \Delta X = \alpha, X = x_0) \tag{8.2}$$

is independent of x_0 for all choices $\alpha, \beta \neq e$, where Z is the key space.

In Theorem 15 we prove that GOST is not a Markov cipher. Actually this is due

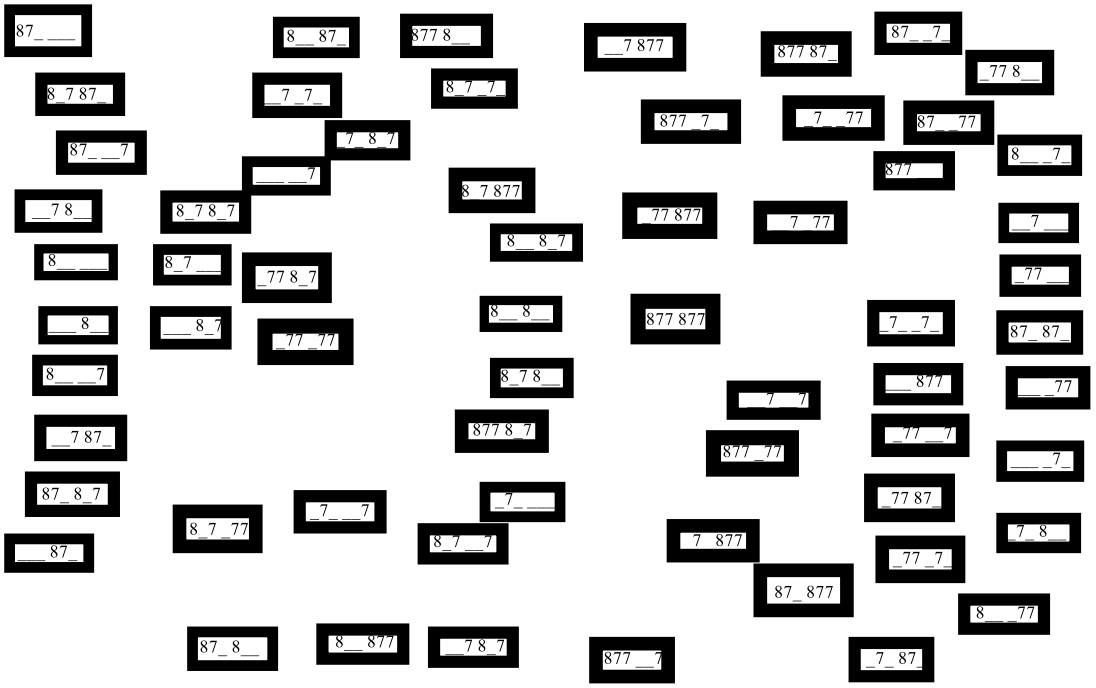

Figure 8.7: Study of Propagation in GOST (8R) - Propagation of [8070070080700700] after 8R of GOST with *GostR3411-94-TestParamSet* set of S-boxes

to the modular 2^{32} addition in the round function, which makes the propagation of differences to be dependent on the values of the sub-key (cf. Theorem 14).

Theorem 14. *GOST is a Markov cipher with respect to bitwise XOR operation $\oplus$ if the modulo 2^{32} addition $\boxplus$ is replaced by the bitwise XOR operation $\oplus$*

Proof. Let $P, C \in \{0,1\}^{64}$ and $K \in \{0,1\}^{256}$ denote the plaintext, the ciphertext and the key space respectively.

Fix $X \in P$ and consider $x, x' \in P$ such that $x \oplus x' = X$. As shown in Figure 8.8 the first operation involved is a bitwise XOR of x and x' with the round 32-bit sub-key k. The difference $X = x \oplus x'$ is mapped to $Z = (x \oplus k) \oplus (x' \oplus k) = x \oplus x'$ and thus is independent of the sub-key k.

Suppose that there exist exactly N pairs $\{(z_i, z_i')\}_{1 \leq i \leq N}$, such that $S(z_i) \oplus S(z_i') = Y$ with $z_i \oplus z_i' = X$, where S the substitution layer as shown in Figure 8.8.

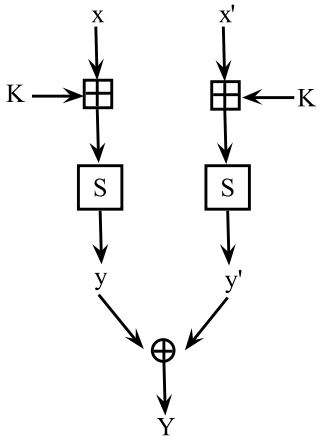

Figure 8.8: Modified GOST - Modified round function in GOST

For a fixed pair (x, x') with $x \oplus x' = X$ we can find a unique key k such that $x \oplus k = z_i$ for each $i \in \{1, .., N\}$ and $x' \oplus k = z_i'$.

By construction $S(z_i) \oplus S(z_i') = Y$. Thus, for a fixed pair of inputs (x, x') with $x \oplus x' = X$, we can find exactly N keys such that the output difference is Y. Thus, modified GOST is a Markov cipher with respect to bitwise XOR. □

Theorem 15. *GOST is not a Markov cipher*

Proof. (Sketch of Proof) The main reason is the key insertion via modulo addition 2^{32}. The number of non-zero components of the difference at the input of the key adder is not equal to the number of non-zero components of the difference at the output of the key adder in the case of modular addition. More details about this proof can be found in [8].

□

In addition, we present below some heuristic evidence by computer simulations that GOST is not a Markov cipher. During our experimentations, we have computed many specific counterexamples to show that GOST is NOT a Markov cipher. We provide counterexamples for the GOST cipher which uses the set of S-boxes "GostR3411-94-TestParamSet".

Using simulations we have obtained the following probability after 1 round of the following transition

$$P(\Delta Y \in [8070070080700700] | \Delta X \in [8070070080700700]) = 2^{-3.73}.$$

However, we have computed the same transition but for fixed plaintexts and we have obtained that

$$P(\Delta Y \in [8070070080700700] | \Delta X \in [8070070080700700], X = X_0)$$

is not exactly equal to $2^{-3.73}$ for different choices of X_0. We present two such examples below,

- *For $X_0 = 000031A90F4A3312$, we get $2^{-3.61}$*
- *For $X_0' = 0000001000000010$, we get $2^{-3.89}$.*

On one hand, we proved that GOST is not a Markov cipher. On the other hand, it seems that the probabilities obtained over different plaintexts in case of truncated differentials, even if they are not exactly equal, are very close. From these observations, it seems that GOST exhibits some Markov structure and it is an ϵ-approximate *Markov cipher (cf. Definition 51). However, this is an observation due to simulations and it is not formally proved.*

Definition 51. *(ϵ- approximate Markov cipher).*

An iterated cipher round function $Y = f(X, Z)$ is an ϵ- approximate Markov cipher if there is a group operation $\otimes$ for defining differences, such that for all choices of α $(\alpha \neq e)$ and $(\beta \neq e)$ and subkey Z chosen uniformly at random, then for all γ we have,

$$\log_2(P_Z(\Delta Y = \beta|\Delta X = \alpha, X = \gamma)) = (1+\epsilon).\log_2(P_{Z,X}(\Delta Y = \beta|\Delta X = \alpha)),$$

for some small constant ϵ

We have computed the probability for truncated differentials after one round of the form

$$p = P(\Delta Y \in [8070070080700700]|\Delta X \in [8070070080700700], X = X_0)$$

for some structure of plaintexts of the form $X_0 = (X_0)_L||(X_0)_R$*, where* $(X_0)_L$ *is fixed and* $(X_0)_R$ *takes all possible* 2^{32} *values. There is some evidence that this value is bounded above and below by* $2^{-3.6}$ *and* $2^{-3.9}$ *respectively.*

In particular, we have that $|p_{max} - p_{min}| \simeq |2^{-3.6} - 2^{-3.9}| \simeq 2^{-6.01}$*, where* p_{min} *and* p_{max} *are the minimum and maximum probabilities obtained by computer simulations. Hence, for this specific set of input-output differences, we expect GOST to behave like a Markov cipher with a margin of error only about approximately* 2^{-7}*. In the next chapter, we are going to introduce a heuristic black-box discovery method for finding transitions in GOST which hold with sufficiently high probability.*

8.5 A Discovery Method for Finding Differential Properties

Algorithm 12 describes a heuristic methodology for the discovery of good sets of differentials in GOST. This algorithm can be used to obtain sufficiently good characteristics, which can be composed efficiently for constructing a distinguisher.

We do not claim that it outputs the best possible transitions that can be found within the space of differentials of our interest. However, the results obtained are sufficiently good for cryptanalytic purposes. All the steps are based on ad-hoc heuristics we have discovered during our analysis. It is an evolutionary black-box algorithm.

Algorithm 12 Heuristic Black-Box Discovery Algorithm for Transitions [52, 51]

Input: Random number X, $1 \leq X \leq 64$ (Number of active bits)

1. Select at random a set of X bits from a 64-bit vector
2. Consider all plaintexts P with corresponding $64 - X$ bits fixed to 0 or 1
3. Repeat the following computation for at least 2 different numbers of rounds (e.g 8 and 12)
4. Check how many of the resulting ciphertexts share the same $64 - X$ bits by encrypting the plaintexts with random keys.
5. Keep a population of some 1000 best sets (in terms of transitional probability) and mix new sets of X bits generated with older sets of X in which we flip a few bits.

In Step 1, we have (heuristically) discovered that suitable selections are 14 or 19 bits. However, as we will see in the end of this section we have some heuristic evidence obtained by computer simulations that 14 bits is the optimal choice. Steps 2 and 3 are based on the method called Structures *by Biham and Shamir and it provides a quadratic speedup in measuring the probability [16].*

Step 3 is a heuristic which improves the transitional probabilities. It is used to ensure that the transitional probability is not only good locally *for some number of rounds, but it is also good enough for more rounds.*

For example, the truncated differential [7800007807070780] *with uniform sampling over all possible input differences, produces an element of the same differential set after 4 or 8 rounds with the following probabilities on average over all possible keys:*

1. *For 4 rounds of GOST with probability* $P_4 = 2^{-13.8}$. *This is NOT as good as* $P_4 = 2^{-13.6}$ *for the previous set* [8070070080700700], *however for 8 rounds it will be otherwise*

2. *After 8 rounds of GOST we obtain probability* $P_8 = 2^{-24.0}$ *on average over all possible keys which is strictly better than* $P_8 = 2^{-25.0}$ *for the previous set* [8070070080700700]

3. *This however does NOT mean that it will be better for 10 rounds. In fact for 10 rounds we obtain less than* 2^{-35} *which is not as good as* $P_{10} = 2^{-31.0}$ *with the previous set* [0x8070070080700700] *(Note that the result is very inexact).*

We see that discovery of interesting iterative invariant attacks on 8 rounds of GOST cannot rely on heuristic combination of $8 = 4 + 4$ *rounds, and that our new result is not very good for 4 rounds yet, now becomes the best ever found for 8 rounds which however does NOT guarantee it is the best for 10 rounds. This justifies our black-box methodology but also shows that it is difficult to find a solution which works for various numbers of rounds.*

Using this simple algorithm, we have obtained the following results (Table 8.4) which appear also in [51, 52]. We observe that for each set of S-boxes we can find truncated differentials which occur with very similar probability. This suggests that possible extensions of attacks against full 32 rounds may be feasible for all versions of GOST using similar ideas as in [36, 48].

Table 8.4: New invariant truncated differential attacks with sets of 19 bits and their propagation probabilities for 8.

S-box Set Name	Best Set S	$P(8R)[S] \to [S]$
TestParamSet	78001078 07070780	$2^{-24.9}$
CryptoProParamSet	08070780 78788030	$2^{-24.4}$
CryptoProParamSetA	78780820 00070707	$2^{-25.2}$
CryptoProParamSetB	80707820 07000787	$2^{-25.9}$
CryptoProParamSetC	78780080 80070707	$2^{-25.5}$
CryptoProParamSetD	84000787 70707800	$2^{-25.4}$
SberbankHash	90000607 D4787800	$2^{-24.9}$
ISO 18033-3	80000707 F0787800	$2^{-23.8}$
GOST-P	F0707000 07000707	$2^{-27.0}$

8.6 Towards Optimal Size for Truncated Differentials

Using computer simulations we have (heuristically) shown that sets of differences with nearly 14 active bits give the best possible propagation for 8 rounds for at least two variants of GOST which uses the S-boxes TestParamSet and CryptoProParamSetA. In addition, a truncated differential of the same size is used later to break a simplified version of GOST with S-boxes replaced by the Identity Map. It seems that for each specific structure of a cipher there is an optimal size of truncated differentials property, which can be used to extend to a full attack.

This is a very important result since among all possible sets we tried for constructing a distinguisher and then extend to an attack, it seems that the choice of sets with 14 active bits is the best one. The results are presented in Figure 8.9 and Table 8.5.

Table 8.5: Best arbitrary invariant sets for some values of $9 \leq a \leq 24$ bits for closed to closed propagation for 8 rounds found by our discovery method.

S-box Set Name	Size	$[S]$	$P_{8R}([S] \rightarrow [S])$
TestParamSet	24	F0780780 F0070781	$2^{-28.3}$
TestParamSet	21	78780000 F0070783	$2^{-26.6}$
TestParamSet	19	78001078 07070780	$2^{-24.9}$
TestParamSet	17	D0707000 80000787	$2^{-23.7}$
TestParamSet	14	80707800 80000307	$2^{-22.6}$
TestParamSet	12	80707800 80000007	$2^{-22.8}$
TestParamSet	9	80700780 80000000	$2^{-25.2}$
CryptoProParamSetA	24	F0770700 F0700708	$2^{-31.0}$
CryptoProParamSetA	21	78780060 80070787	$2^{-25.4}$
CryptoProParamSetA	19	78780820 00070707	$2^{-25.2}$
CryptoProParamSetA	17	03070780 78008070	$2^{-24.2}$
CryptoProParamSetA	14	70780000 80030780	$2^{-23.8}$
CryptoProParamSetA	12	70780000 80080700	$2^{-26.7}$

The curve illustrated in Figure 8.9 suggests that there exist truncated differentials of optimal size which can be used in a possible attack against the full block cipher and as we have discussed later this can lead to a new security notion against DC based on truncated differentials.

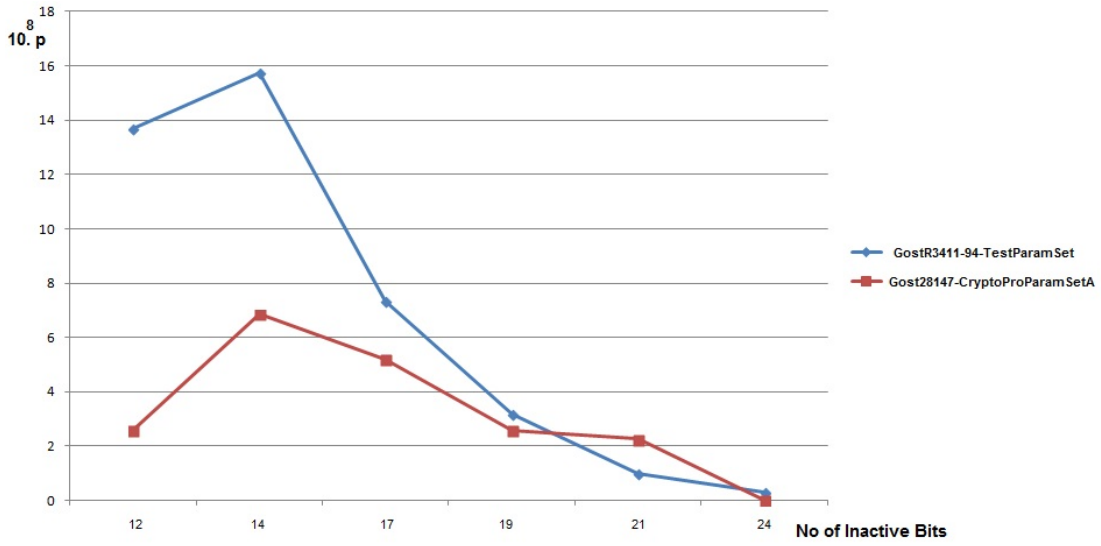

Figure 8.9: Sets of Bits vs Probability of Propagation - The variation of probability of transition for different sizes of sets of active bits for two variants of GOST for 8 rounds

8.7 GOST S-boxes

DC since its invention is considered as a guide for the design of block ciphers and designers concentrated on the establishment of criteria and design principles that make ciphers resistant against DC. All optimal *4-bit to 4-bit S-boxes against linear and differential cryptanalysis are known and classified up to affine equivalence [89].*

There are in total 16 disjoint equivalence classes for 4-bit to 4-bit optimal S-boxes with respect to affine equivalence. Following [89] we denote by G_i the representative of i-th class out of 16 total classes and then for each variant of GOST we have computed the equivalence class in which each S-box lies. Other notations in Table A.5 such as Lu1 or 36 are purely conventional and have no other meaning than show that certain S-boxes are Affine Equivalents *of some other known S-boxes.*

Using a simple constraint satisfaction algorithm as in [89] and then a SAT solver, we have obtained the results shown on Table A.5 [52] (cf. Appendix A.3). From Table A.5, we conclude that GOST designers followed exactly this design principles based on optimality of S-boxes with respect to DC. We observe that most S-boxes in GOST are affine equivalents of their own inverses. In addition, 9 out of 16 different class representatives appear in our table, which gives us some insights about the designers of GOST that they have followed precisely this theory about optimality of S-boxes.

8.8 Construction of Reduced Round Distinguishers

A distinguisher *is an algorithmic construction which allows us to distinguish a given block cipher from a random permutation [72, 50]. The existence of a distinguisher does not always imply that an attack is feasible against a cryptographic primitive since such an extension is not straightforward. However, it means that the primitive is in question weak and it is not sufficiently secure to be used for encryption of top classified documents.*

In this section, we describe a general methodology for the construction of efficient distinguishers for reduced versions of GOST, which is based on propagations of well-chosen general open sets and truncated differentials related to the variant of GOST we study [50, 51, 52, 49].

This methodology is heuristic, in a sense that it does not prove that the best distinguisher is obtained. However, the results obtained are sufficiently good for mounting attacks against full block cipher and the best we could find so far. Our construction allows us to distinguish a 20-round version of GOST from a random permutation. Similar construction has been found and successfully applied to obtain full differential attacks in [47, 46, 36].

Our construction works as follows. Using the discovery method we obtain general open sets $X_1, X_2, ..., X_r$ *which propagate with sufficiently high probability for m rounds of GOST (cf. Figure 8.10). Then, we compute the cumulative probability of a transition from any difference in* $\{X_1, X_2, ..., X_r\}$ *to itself for the middle* $n-2m$ *rounds. We select* X_i, X_j *as the input and output differences, which maximize the cumulative probability of the distinguisher.*

The aim is to distinguish a reduced version of GOST from a random permutation on 64 bits. We exploit the propagation of differences to build such distinguisher. Firstly, we study the statistical properties of a random permutation. For a random permutation on 64 bits we compute the transitional probability $X_i \rightarrow X_j$*, where* X_i *and* X_j *are open sets with sizes* $|X_i|$ *and* $|X_j|$ *respectively, as follows (cf. Lemma 10).*

Lemma 10. *(Random Permutation Property on 64 bits for sets)*

Let $P : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$ *be a uniformly random permutation. Given all pairs of inputs* (P_i, P_j) *with* $P_i \oplus P_j \in X_i$*, where* X_i *is a set of non-zero differences, then the average number of resulting pairs* $(P(P_i), P(P_j))$ *that satisfy* $P(P_i) \oplus P(P_j) \in X_j$*, denoted by* E_{ref}*, where* X_j *is a set of non-zero differences, is given by,*

$$E_{ref} = \frac{|X_i|.|X_j|}{2} \tag{8.3}$$

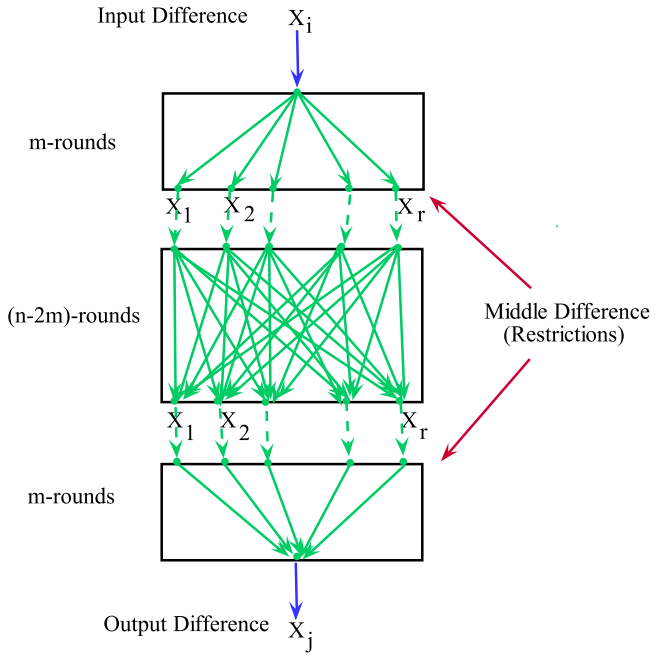

Figure 8.10: Distinguisher Construction - General construction of a distinguisher based on individual transitional probabilities of *general open sets*

Proof. For a random permutation P we have that every single combination of an input differential on 64 bits, and of an output differential on 64 bits, is expected to occur about $\frac{1}{2}$ times on average.

We have in total 2^{127} pairs of inputs and about 2^{128} possible sets of two differentials. In a pair of input output differences X_i,X_j we have $|X_i|.|X_j|$ possibilities.

Overall, we expect to obtain $\frac{1}{2}.|X_i|.|X_j|$ pairs of inputs (P_i, P_j) with $P_i \oplus P_j \in X_i$ such that $P(P_i) \oplus P(P_j) \in X_j$. □

In the rest of this section, we study the properties of the distinguishers of the form shown in Figure 8.10 based on the notion of general open sets and their closure.

Lemma 11. *(Size of General Open Set)*

Let Xbe a general open set on the alphabet $0, 7, 8, F$ and let $N_7(X)$, $N_F(X)$ and $N_8(X)$ denote the number of 7s, Fs and 8s respectively. Then, the cardinality of X is given by

$$|X| = 7^{N_7(X)+N_F(X)}$$

Proof. If X contains either 0 or 8, then these entries do no alter its size since they are fixed bytes to 0000 and 1000 respectively.

The entries 7 and F correspond to bytes of the form $0x_1x_2x_3$ and $1x_1x_2x_3$ respectively, such that not all x_1, x_2, x_3 can be zero simultaneously. Thus, each byte of this type corresponds to $2^3 - 1 = 7$ different options.

The general open set consists of 16 bytes on the alphabet $\{0, 7, 8, F\}$, then it contains precisely $7^{\#(7's)+\#(F's)}$ elements. □

Lemma 12. *(Size of the Closure of Closed Sets)*

Given a general open set X on the alphabet $0, 7, 8, F$. Then, the cardinality of its closure set $[X]$ is given by

$$|[X]| = 2^{N_8(X)}.8^{N_7(X)}.16^{N_F(X)} - 1$$

Proof. By definition of the closure of the set we allow all possible differences except the zero difference. Thus, a byte 8 allows for 2 cases, a byte 7 allows for 8 cases and a byte F allows for 16 cases. □

Next we compute the number of expected pairs with this input and output difference but only due to propagation inside the cipher. By the term propagation we mean also a transition which also has the middle difference specified by the distinguisher.

Lemma 13. *(Cumulative Probability of Distinguisher)*

The expected number of pairs (P_i, P_j) with input difference in X_i which follow the differential characteristic shown in Figure 8.10 is approximately given by:

$$E_{ij} = 2^{63}.|X_i|.\sum_{m,n}(P(X_i \to X_m).P(X_m \to X_n).P(X_n \to X_j)) \tag{8.4}$$

Proof. The expected number of pairs (P_i, P_j) with $P_i \oplus P_j$ in open set X_i is given by $2^{64}.|X_i|.\frac{1}{2}$. Any difference in the set X_i is mapped to any difference in X_j over random key with probability,

$$p_{ij} = \textstyle\sum_{m',n'}(P(X_i \to X_{m'}).P(X_{m'} \to X_{n'}).P(X_{n'} \to X_j))$$

Then, the expected number of output pairs is given by,

$$E_{ij} = 2^{63}.|X_i|.p_{ij} \tag{8.5}$$

□

The aim of the distinguisher shown in Figure 8.10 is to distinguish the distributions of the number of pairs which have the input-output difference as specified by the distinguisher (naturally) and the number of pairs which additionally have the middle differences (propagation).

In the first case, we expect on average E_{ref} pairs, while in the second case we expect $E_{i,j} + E_{ref} - E_{inter}$ pairs, where E_{inter} is the number of pairs which occur naturally but also have this middle difference property. If this number is non-negligible, then the analysis becomes more complex. However, we can use middle difference which have the property to make the two sets to be entirely disjoint. Due to our freedom in the selection of middle sets we can assume that E_{inter} is negligible and this argument is described in details later.

Assuming that the sets are entirely disjoint, the distributions of the sample means $\mathcal{E}_{ref}$ and $\mathcal{E}_{i,j} + \mathcal{E}_{ref}$ can be approximated by Gaussian Distributions $\mathcal{X} \sim \mathcal{N}(E_{ref}, E_{ref})$ and $\mathcal{Y} \sim \mathcal{N}(E_{i,j} + E_{ref}, E_{i,j} + E_{ref})$ respectively. In our case, since we use intermediate difference we can assume that the intersection of the two sets is negligible. The variance in both cases equals the mean since we approximate the distributions of the variable of number of pairs by Poisson distribution.

In Chapter 7, we have studied the problem of distinguishing two Gaussian Distributions. We define the advantage of the distinguisher as a measure of expressing the number of standard deviations that the mean of distribution $\mathcal{X}$ deviates from that of $\mathcal{Y}$.

Definition 52. *(Advantage of Distinguisher)*

The advantage of a distinguisher $\mathcal{A}$ for distinguishing $\mathcal{X} \sim \mathcal{N}(\mu_1, \sigma_1^2)$ over $\mathcal{Y} \sim \mathcal{N}(\mu_2, \sigma_2^2)$ is given by

$$ADV = \frac{(\mu_2 - \mu_1)}{\sigma_1} \tag{8.6}$$

We know that by CLT the sample mean distribution will be approximately Gaussian with mean equal to the variance. Thus, the advantage is given by $\frac{E_{i,j}}{\sqrt{E_{ref}}}$. Then, we make a standard hypothesis test as described in Chapter 7 to distinguish the two cases. We apply this methodology for constructing efficient distinguishers for three different versions of GOST; GostR3411-94-TestParamSet, Gost28147-CryptoProParamSetA and GOST ISO 1803-3.

Theorem 16. *(20-Round Distinguisher on GOSTR3411-94-TestParamSet)*

$$8780070780707000$$
$$\downarrow (10R)$$
$$[8070070080700700]$$
$$\downarrow (10R)$$
$$80707000087800707$$

is a 20 rounds distinguisher where $[8070070080700700]$ *is a closed set, and satisfies the following properties,*

1. *If the 20 rounds are replaced by a random permutation, then out of the total of* 2^{77} *pairs of plaintexts* (P_i, P_j) *such that* $P_i \oplus P_j \in 8780070780707000$*, we expect on average* $2^{27.1}$ *to satisfy also the output difference at the end of the 20 rounds.*

2. *Among all input pairs with input difference in the set* 8780070780707000*, we expect on average* $2^{18.1} + 2^{27.1}$ *after 20 rounds to follow the differential characteristic 10+10 shown above.*

3. *The advantage of the distinguisher is 25.8 standard deviations*

Proof. For a typical permutation on 64 bits (which does not have to be a random permutation, it can be GOST with more rounds) out of total 2^{77} plaintext pairs (P_i, P_j) which satisfy the specified input difference, we expect on average $2^{27.10}$ such pairs to satisfy also the output difference after 20 rounds (cf. Lemma 10). The distribution of the expected number of pairs which satisfy both input and output difference is approximated by a Normal distribution $\mathcal{N}(2^{27.10}, 2^{13.55})$.

For 20 rounds of GOST and for a given random key, we expect such pairs to occur both by accident (naturally occurring as in a random permutation) and due to propagation in GOST. This is because the length of the key is larger than the size of the block.

Let $\mathfrak{X}$ denote the distribution of expected number of pairs occurring naturally and $\mathfrak{Z}$ the distribution of expected number of pairs occurring due to propagation. By computer simulations, we have obtained the probability of the following transition

$$8780070780707000 \rightarrow [8070070080700700]$$

after 10 rounds of GOST and was found approximately equal to $2^{-29.4}$. That implies that the mean of the distribution $\mathfrak{Z}$ is $2^{18.2}$.

In case of a random permutation, the expected number of pairs which have this additional middle difference is $2^{27.1-29.4-29.4} = 2^{-31.7}$ (no pairs in practise). Thus, this

middle difference property can be seen as an artificial assumption which separates the two sets.

Hence, the distribution $\mathcal{Y} = \mathcal{X} + \mathcal{Z}$ has mean approximately $2^{27.1} + 2^{18.2}$. Thus, the advantage of the distinguisher is given by $\frac{2^{18.2}}{2^{13.55}}$, which is approximately 25.8 standard deviations.

Remark 18. *If we set $2^{27.1} + 2^{18.2}$ as a threshold to accept the key as correct, then the guess is correct with probability set at $\frac{1}{2}$. The probability of a false positive (Type I error) is given by,*

$$P(\mathcal{Y} > 2^{27.1} + 2^{18.1}) = \tfrac{1}{2}(1 - erf(\tfrac{25.8}{\sqrt{2}})) \simeq 2^{-485}$$

□

Theorem 17. *(20-Round Distinguisher on GOST28147-CryptoProParamSetA)*

$$0770070077777770$$
$$\downarrow \textit{(10R)}$$
$$[7007070070070700]$$
$$\downarrow \textit{(10R)}$$
$$7777777007700700$$

is a 20 rounds distinguisher where [7007070070070700] *is a closed set, and satisfies the following properties,*

1. *If 20 rounds are replaced by a random permutation, we expect on average $2^{55.1}$ to satisfy both input-output differences after 20 rounds.*

2. *Among all input pairs with input difference in the set* 0770070077777770*, we expect on average $2^{33.0} + 2^{55.1}$ after 20 rounds to follow the differential characteristic*

3. *The advantage of the distinguisher is 42.2 standard deviations*

Proof. For a typical permutation on 64 bits out of the total plaintext pairs (P_i, P_j) with this input difference we expect $2^{55.10}$ such pairs to satisfy also the desired output difference (cf. Lemma 10). The distribution of the expected number of pairs is approximated by a Normal distribution of the form $\mathcal{N}(2^{55.10}, 2^{27.55})$.

We have computed the probability of transition

$$[7007070070070700] \rightarrow 7777777007700700$$

and found to be approximately equal to $2^{-24.01}$ after 10 rounds.

Again the two sets are entirely disjoint for same reasons explained in the previous theorem.

Thus, the distribution $\mathcal{Y} = \mathcal{X} + \mathcal{Z}$ has mean $2^{55.10} + 2^{33.0}$. The advantage of the distinguisher is approximately 42.24 standard deviations and corresponds to Type I error 2^{-1290}. □

Theorem 18. *(20-Round Distinguisher on GOST ISO 18033-3)*

$$
\begin{array}{c}
8000070770700000 \\
\downarrow (6R) \\
[7078000070000700] \\
\downarrow (8R) \\
[7000070070780000] \\
\downarrow (6R) \\
7070000080000707
\end{array}
$$

is a 20 rounds distinguisher where [7000070070780000] *is a closed set, and satisfies the following properties,*

1. *If 20 rounds are replaced by a random permutation, we expect on average* $2^{21.5}$ *to satisfy both input-output differences after 20 rounds.*
2. *Among all input pairs with input difference in the set* 8000070770700000*, we expect on average* $2^{15.9} + 2^{21.5}$ *after 20 rounds to follow the differential characteristic*
3. *The advantage of the distinguisher is 35.09 standard deviations*

Proof. For a typical permutation on 64 bits, we have the distribution $\mathcal{N}(2^{21.5}, 2^{10.75})$ (cf. Lemma 10). By computer simulations we have obtained the following transitional probabilities after 6 and 8 rounds respectively,

$$P([7078000070000700] \rightarrow 8000070770700000) = 2^{-16.47}$$

$$P([7007070070070700] \rightarrow [7000070070780000]) = 2^{-27.20}$$

Hence, out of the total 2^{77} pairs with the input difference as specified in the 20-round construction, we expect approximately $2^{77-17.47-27.20-16.47} = 2^{15.90}$ (The size of the set 8000070770700000 is half the size of the set [7078000070000700] and thus we can assume that the probability is halved in the reverse direction). Thus, the mean of the distribution $\mathcal{Z}$ (due to propagation) is $2^{15.90}$.

In case of a random permutation, the expected number of pairs which have in addition this specific middle difference is $2^{21.5-17.47-16.47} = 2^{-12.44}$ (no pairs in practise).

Thus, the distribution $\mathcal{Y} = \mathcal{X} + \mathcal{Z}$ has mean $2^{15.90} + 2^{21.50}$. The advantage of the distinguisher is given by $\frac{2^{15.90}}{2^{10.75}}$, which is approximately 35.09 standard deviations and this corresponds to Type I error 2^{-894}. □

8.9 Validation: Open to Open Transitions

The distinguishers presented in the previous section consist of combining transitions from open to open set after 20 rounds, but with some middle differences which are closed sets. We have re-done all the computations of transitional probabilities on entirely general open sets and for different partitions of the 20 rounds and we have obtained in all cases roughly about the same results. That implies that estimation based on closed sets is reliable enough. Transitional probabilities based on open sets give more reliable and robust results.

As a proof of concept, we include in the thesis a distinguisher on 20 rounds for the GostR3411-94-TestParamSet, which is based entirely on general open sets and the partition of the 20 rounds is of the form $6+4+4+6$. *The input and output of the 20R distinguisher are 8780070780707000 and 8070700087800707 respectively.*

In Table 8.6 some of the dominating paths inside the full construction are listed. The probabilities described are all of the form 2^{-x} *(only* $-x$ *is shown) and are ranked in decreasing order. Note that in this construction the expected mean of the distribution due to propagation is approximately* $2^{17.6}$ *events and this number can be improved even more in the future by running more simulations and aggregating the results. The sets* O_i *and probabilities* p_j *shown on Table 8.6 are associated with the following diagram.*

$$
\begin{array}{c}
8780070780707000 \\
\downarrow p_0 \\
O_1 \\
\downarrow p_1 \\
O_2 \\
\downarrow p_2 \\
O_3 \\
\downarrow p_3 \\
8070700087800707
\end{array}
$$

Table 8.6: Transitional Probabilities for 20 round distinguisher for GOST with set of s-boxes TestParamSet

O_1	O_2	O_3	p_0	p_1	p_2	p_3	Total
000800	807807	800000	-22.4	-11.1	-16.7	-9.4	-59.6
007870	077077	870007	-22.6	-16.9	-22.3	-14.2	-75.9
807877	870870	877807	-23.0	-22.3	-21.3	-17.4	-83.9
877077	877077	077877	-25.6	-21.7	-18.7	-23.5	-89.5
877077	007877	077877	-25.6	-19.6	-20.9	-23.5	-89.6
877077	077077	077877	-25.6	-20.4	-20.1	-23.5	-89.7
877077	077077	077877	-25.6	-20.7	-20.7	-23.5	-90.5
877077	807877	077877	-25.6	-21.3	-20.5	-23.5	-90.8
877077	877077	077877	-25.6	-23.3	-18.7	-23.5	-91.1
877077	800877	077877	-25.6	-23.4	-20.7	-23.5	-93.1

Using the results shown on Table 8.6 and given 2^{77} *pairs of plaintext* (P_i, P_j)*, such that* $P_i \oplus P_j \in 8780070780707000$*, we expect on average* $2^{77}.2^{-59.6} = 2^{17.4}$ *pairs to follow the differential characteristic shown on the first row of the table. Using the most dominant transitions shown in the same table we finally obtain approximately,*

$$2^{77}.(2^{-59.6} + 2^{-75.9} + 2^{-83.9} + 2^{-89.5}) = 2^{17.4}$$

such pairs. Note that all such results are subject to further improvements, if more simulations are performed. However, the aim is to validate that the results obtained based on closed sets in the middle are reliable enough.

Part IV

Attacks on Full GOST Block Cipher

9

Parametric Attacks on Full GOST

In Chapter 8, we discussed a framework for the construction of reduced round distinguishers for GOST block cipher based on special sets of truncated differentials dictated by the structure of GOST. This concept allowed us to narrow down the exponentially large space of truncated differentials and restrict the study to a smaller subspace, where interesting differential properties can be discovered. We used a heuristic technique for the discovery of good truncated differential properties that combines several steps like random guessing and there are several learning loops implemented, which perform operations like flip few bits, extend size of the set (number of active bits), decrease the size of the set or use repeated patterns in order to obtain a sufficiently good truncated differential property. It is a sort of an evolutionary *black-box algorithm. Note that due to the heuristic nature of this algorithm, not the best differential properties are guaranteed to be found.*

Based on this methodology, we constructed distinguishers (up to 20-rounds) for three different variants of GOST; TestParamSet, CryptoProParamSet *and* ISO 18033-3. *All these sets are of major importance since they are implemented in many standards and used by many organizations. The first one appears as the default set of S-boxes used in all available implementations. The second one is used in the hash function implementation and by many large bank organizations, while the last one is the one which is believed to be the strongest and was suggested in the ISO standardization process to become a global industrial standard [107].*

In this chapter, we study advanced differential attacks on full 32-rounds of GOST based on a statistical distinguisher. In particular, we apply our techniques to evaluate

the security of the three variants of GOST for which we have constructed 20-round distinguishers. In our attack, we split GOST into three parts with $6 + 20 + 6$ *rounds, where the middle 20 rounds can be 20 rounds of GOST, more rounds of GOST, or maybe a random permutation. Our attack is generic and it is a Depth-first search like approach where some key bits for several outer rounds are guessed and then confirmed or rejected by the differential properties of our distinguisher. Note that in all the attacks that we describe, we assume that the set of S-boxes is known to the attacker. Lastly, we describe an attack against a simplified version of GOST with S-boxes replaced by the Identity map (abbreviated as GOST-ID) based on a 26-round statistical distinguisher and following precisely the same methodology.*

9.1 On the Type I Error of the 20-round Distinguishers

*We recall that Type I error is about accepting wrong keys (**false positives**). The core idea of our attack is that we initially set a* threshold *for the expected number of pairs which satisfy both input and output differences as specified by a 20-round distinguisher. During the attack if the number of such events observed exceeds this pre-specified threshold, then we accept a given key assumption as correct.*

For example, if we set the threshold to be $E_{ref} + E_{ij}$ *(cf. Chapter 8), then we accept the correct key assumption. This implies that the probability of rejecting the correct key assumption during the attack (**false negative**) is set at* $\frac{1}{2}$*. On the other hand, the probability of accepting such a key assumption, while it is indeed false is predicted by the Gaussian error function. The corresponding values for the three different variants of GOST which we study are summarized in Table 9.1.*

Table 9.1: The Type I error related to the three variants of GOST which use the three different sets of S-boxes as shown. All these values were obtained using MAPLE software by typing $log[2.](erfc(X/sqrt(2.)))$;, where X is the associated advantage of the statistical distinguisher

S-box Set Name	**Adv**	P_I =**Type I Error**
TestParamSet	25.8	2^{-485}
CryptoProParamSetA	42.2	2^{-1290}
ISO 18033-3	35.1	2^{-894}

Assume that in an attack, we need to guess k key bits in order to compute the number of events of our interest, which in this case are pairs which follow certain differential properties. If the number of observed events exceeds the pre-specified threshold, then we accept the corresponding key assumption as correct. Thus, at the end of the attack we end up with $2^k.P_I$ *key candidates and with probability* $\frac{1}{2}$ *the correct key lies in this set. Since* $-log_2 P_I > 256$*, where 256 is the length of the key in GOST, then the probability of accepting a wrong key assumption as correct is practically zero for the three cases of our interest.*

Additionally, low values of Type I error correspond to large advantages for the statistical distinguishers and thus possible extension of the distinguisher to more rounds is worth studying. For example, we can gain a total of two rounds by extending a distinguisher on 20 rounds to a "weaker" distinguisher on 22 rounds. In the next section, we study possible extensions of our distinguishers to more rounds and in the other sections we study attacks against full 32-rounds of GOST for the three variants of our interest.

9.2 Extension of Statistical Distinguishers to More Rounds

In this section, we discuss possible extensions of our statistical distinguishers to more rounds. In particular, we study if an extension of a 20-round distinguisher to $20 + 2r$ *rounds is feasible. Suppose we are given a 20-round distinguisher with input differential set* $X_i = (X, Y)$ *and output differential set* $X_j = (Y, X)$ *which holds with probability* p_{ij} *in case of GOST. Then, we consider external transitions of the form* $X'_i \rightarrow X_i$ *(with probability of transition* p'*) and* $X_j \rightarrow X'_j$ *(with probability of transition* p*) as shown below. Note that the notation* (A, B) *is a set of differences on 64 bits and* A, B *are the differences on the left and right 32-bit halves respectively.*

$$
\begin{gathered}
X'_i = (Z, Z') \\
\textit{(r rounds)}\downarrow p' \\
X_i = (X, Y) \\
\textit{(20 rounds)}\downarrow p_{ij} \\
X_j = (Y, X) \\
\textit{(r rounds)}\downarrow p \\
X'_j = (Z', Z)
\end{gathered}
$$

Recall that in case of a 20-round distinguisher we have the following two Gaussian distributions to distinguish.

1. *Random Permutation:* $\mu_1 = \frac{|X_i|.|X_j|}{2}, \sigma_1 = \sqrt{\frac{|X_i|.|X_j|}{2}}$
2. *GOST Propagation:* $\mu_2 = \frac{|X_i|.|X_j|}{2} + 2^{63}.|X_i|.p_{ij}, \sigma_2^2 = \mu_2$

Considering the $(20 + 2r)$*-round construction and assuming that middle differences in 20 rounds are according to the previous construction, then we have the following Gaussian Distributions to distinguish.*

1. $\mathcal{D}_1$*:* $\mu_1 = \frac{|X'_i|.|X'_j|}{2}$, $\sigma_1^2 = \mu_1$
2. $\mathcal{D}_2$*:* $\mu_2 = \frac{|X'_i|.|X'_j|}{2} + 2^{63}.|X'_i|.p'.p_{ij}.p$, $\sigma_2^2 = \mu_2$

We assume that two sets of events are disjoint, otherwise we would have to subtract their intersection (cf. Chapter 8, Section 8.10).

The value of $\frac{ADV_{(20+2r)R}}{\sqrt{2}}$ *is now given by*

$$2^{63}.p_{ij}.p.p'.\sqrt{\frac{|X'_i|}{|X'_j|}}$$

Assuming that X_i *is the symmetric set of* X_j *and* X'_i *the symmetric set of* X'_j*, we have that* $p' = p.\frac{|X_i|}{|X'_i|}$*. By symmetric set we mean that one set is obtained by exchanging the right and left halves of the other set.*

Thus, the value of $\frac{ADV_{(20+2r)R}}{\sqrt{2}}$ *now becomes,*

$$2^{63}.p_{ij}.p^2.\frac{|X_i|}{|X'_i|}$$

Denote by ADV_{20R} *the advantage of the distinguisher on the 20 rounds. Then, the new construction has advantage given by*

$$ADV_{(20+2r)R} = ADV_{20R}.p^2.\frac{|X_i|}{|X'_i|} \tag{9.1}$$

We need to solve an optimization problem in order to find the best possible sets X'_i, X'_j *that give the highest possible advantage for* $ADV_{(20+2r)R}$*. We observe that the advantage of the new distinguisher is expected to decrease and thus the new extended distinguisher would be less effective. However, the decrease will be very rapid from what we have observed by trying to extend for 1 or 2 rounds more. Even though our distinguishers are very efficient for 20 rounds there is no way so far found by our searches to extend them to more rounds. Using computer simulations we have obtained some "weak" distinguishers which are presented below. By weak we mean that the corresponding Type I error is nearly 0.5 and thus cannot be exploited by our attack.*

Result 1. *(22-R distinguisher for GOST with TestParamSet set of S-boxes)*
The following construction is a 22-round distinguisher with $ADV = 4.6.10^{-6}$.

$$
\begin{gathered}
8078777787800707 \\
(1-R)\downarrow \\
8780070780707000 \\
(20-R)\downarrow \\
80707000087800707 \\
(1-R)\downarrow \\
8780070780787777
\end{gathered}
$$

Result 2. *(22-R distinguisher for GOST with CryptoProParamSetA set of S-boxes)*
The following construction is a 22-round distinguisher with $ADV = 0.03$.

$$
\begin{gathered}
F77F777707700700 \\
(1-R)\downarrow \\
0770070077777770 \\
(20-R)\downarrow \\
7777777007700700 \\
(1-R)\downarrow \\
07700700F77F7777
\end{gathered}
$$

Result 3. *(22-R distinguisher for GOST with ISO 18033-3 set of S-boxes)*
The following construction is a weak 22 round distinguisher with $ADV = 0.01$.

$$
\begin{gathered}
7078778080000707 \\
(1-R)\downarrow \\
8000070770700000 \\
(20-R)\downarrow \\
7070000080000707 \\
(1-R)\downarrow \\
8000070770787780
\end{gathered}
$$

Remark 19. *Note that none of the above three 22-R distinguishers is suitable to be used in our attacks since they have very small Advantage (cf. Chapter 8) and hence very large Type I error.*

9.3 From Distinguishers to Filters

The main idea behind our attack is to guess a few key bits for some outer rounds and then by encrypting and decrypting the 2^{64} (P, C) *pairs given for the full 32 rounds of GOST, we get pairs* (P', C') *for the middle 20 rounds. Then, we can either discard the initial key assumptions or proceed to the next stage of our attack by guessing the remaining key bits. This number has to exceed a pre-specified threshold depending on the 20-round statistical distinguisher in order to proceed to the next stage of the attack.*

The most costly part is the encryption and decryption of these pairs and then determine how many pairs satisfy the required properties. Guessing 192 bits of the key and then determining the $E_{ref} + E_{ij}$ *is very costly. For example, given the entire codebook* 2^{64} (P, C) *pairs and a 20-round distinguisher, in order to count the number of events of our interest, we need to guess* 192 *key bits for the* $6+6$ *outer rounds and encrypt the entire codebook. This step already has time complexity (in terms of GOST encryptions) given by,*

$$2^{192}.2^{64}.\frac{12}{32} \simeq 2^{254.6} \tag{9.2}$$

Since our attack needs approximately 2 iterations (due to Type II error)to succeed, this means that the complexity is $2^{255.6}$ *which is very close to the brute-force attack. However, we can decrease the time complexity by exploiting the poor diffusion for limited number of rounds in GOST so that we have to guess keys for less outer rounds. This is due to very poor diffusion for limited number of rounds which allows us to guess the certain difference bits with sufficiently high probability. We can improve this by* progressive filtering*, where we guess less key bits, determine more pairs and then guess a few more key bits but work with less data (pairs).*

By exploiting the poor diffusion inside GOST for limited number of rounds, we can guess less key bits instead of the full $6+6$ *rounds by guessing some external transitions which take place with sufficiently high probability. Then, instead of computing pairs for the middle 20 rounds, we compute pairs for some outer rounds but by computing less key bits.*

By exploiting the poor diffusion for limited number of rounds, we can find outer differences X_i' *some rounds before the input differences in the 20-round distinguisher and* X_i *some rounds after the output differences which are "provoked" by the middle 20-round distinguisher. Since we have constructed symmetric distinguishers, we can assume that* X_i *is obtained from* X_i' *by exchanging the left and right half. If the transition* $X_j \rightarrow X_j'$ *occurs with probability* p*, then we have the following two Gaussian*

distributions to distinguish.

1. $\mathcal{D}_1$: $p^2.\frac{|X_i|.|X_j|}{2}$
2. $\mathcal{D}_2$: $p^2.\frac{|X_i|.|X_j|}{2} + 2^{63}.|X_i|.p_{ij}.p^2$

Thus, the advantage of the filter now becomes,

$$ADV_{filter} = ADV_{20R}.p \tag{9.3}$$

Thus, in case of filtering we are interested in finding external differences which propagate with probability as large as possible and have the output differences specified in the input and the output of the 20 round distinguisher.

Thus, now our threshold c is set to

$$c = p^2.\left(\frac{|X_i|.|X_j|}{2}\right) + 2^{63}.|X_i|.p_{ij} \tag{9.4}$$

and we accept a given key assumption as correct if the average number of events of our interest observed exceeds this value c. The Type I error is computed using the value of ADV_{filter}, while the Type II error is set at $\frac{1}{2}$.

9.4 Attacks Against Full GOST

In this section, we describe attacks against full 32 rounds of GOST for three variants of GOST using 20-round distinguishers and extra filtering steps as explained precisely in the previous section. Our attack is a Depth-first search approach (cf. Fig. *9.1), where we initially guess some key bits for some outer rounds in order to determine the number of pairs which satisfy some constraints imposed by the filtering step and then guess more key bits and check the number of pairs which satisfy the constraints imposed by the middle 20-round distinguisher.*

We use a partition of the 32 rounds as $x + (6 - x) + 20 + (6 - x) + x$*, where* x *is any number of rounds less or equal to 5. In the filtering step, we use transitions of differences of the form* $X'_i \rightarrow X_i$ *and* $X_j \rightarrow X'_j$ *which hold with sufficiently high probability for* $6 - x$ *rounds (natural events "provoked" by the middle 20-R event). The following diagram represents the way we split the full 32 rounds.*

Plaintext Pairs
(x rounds)$\downarrow$
$X'_i = (Z, Z')$
$(6 - x$ *rounds)*$\downarrow p'$
$X_i = (X, Y)$
(20 rounds)$\downarrow p_{ij}$
$X_j = (Y, X)$
$(6 - x$ *rounds)*$\downarrow p$
$X'_j = (Z', Z)$
(x rounds)$\downarrow$
Ciphertext Pairs

Attack 1 describes the steps of our attacks with the required complexity.

Attack 1. *(20-R Distinguisher $X_i \rightarrow X_j$, Transitions $X'_i \rightarrow X_i$ and $X_j \rightarrow X'_j$ for $6-x$ rounds)*

1. For each guess of the k key bits for the first x ($x \leq 5$) rounds, do the following steps.

1a. For all 2^{64} pairs (P_l, C_l) (full 32-R):

Compute $P'_l = G_{x,k}(P_l)$ and $C'_l = G_{x,k}(C_l)$, where $G_{x,k}$ is the encryption for the first x rounds using key k (which is the same for the last x rounds due to the Key Schedule). At this step we have computed all the (P'_l, C'_l) for the middle $32-2x$ rounds.

Store a list of $(32-2x)$-round (P'_l, C'_l) pairs in a hash table, sorted by their $128 - log_2(|X'_i|) - log_2(|X'_j|)$ inactive bits. While we are computing a (P', C') pair for the middle $(32-2x)$ rounds, we check if for a new pair computed we have a collision on the inactive bits. If such a collision is found, this corresponds to pair of plaintexts (P'_l, P'_m) such that $P'_l \oplus P'_m \in X'_i$ and $C'_l \oplus C'_m \in X'_j$ after $(32-2x)$ rounds (Because we do it for fixed number of rounds which is 20 rounds we assume the complexity of hash table construction is constant).

This list requires memory of about $2^{64}.64 = 2^{70}$ bits.

The time complexity of this step in terms of GOST encryptions is

$$T_1(x) = 2^{32x}.2^{64}.\frac{2x}{32} \simeq 2^{60+32x+log_2(x)} \tag{9.5}$$

and it returns about $\frac{|X'_i|.|X'_j|}{2}$ triples $(k, (P'_i, C'_i), (P'_j, C_j))$.

1b. For the total of $\frac{|X'_i|.|X'_j|}{2}$ collisions of the form $((P'_m, C'_m), (P'_n, C'_n))$ which have been computed in the previous step, we want to count the number of pairs, which satisfy both input and output difference as specified by the middle 20-R distinguisher. Let T the number of such pairs which satisfy the required constrained imposed by the distinguisher.

We compute T by guessing the remaining $192-32x$ bits for the remaining $6-x$ rounds and each time the new pair $((P''_m, C''_m), (P''_n, C''_n))$ for the middle 20 rounds satisfy the required property we increase the counter by 1. This has time complexity in terms of GOST encryptions given by,

$$T_2(x) = 2^{32x}.2^{32(6-x)}.\frac{|X'_i|.|X'_j|}{2}.\frac{12-2x}{32} \simeq 2^{187+log_2(6-x)+log_2|X'_i|+log_2|X'_j|} \tag{9.6}$$

If the counter $T > c$ then we accept the 192-bit key assumption as correct, otherwise we reject it.

2. If the Type I error equals to 2^{-y}, this implies that we are left with approximately 2^{192-y} possible key candidates on the 192 bits of the key. The remaining $256-192=64$ can be found using additional pairs for the full 32-rounds.

The complexity of this step is given by,

$$T_3(y) = 2^{192-y+64} = 2^{256-y} \tag{9.7}$$

The overall time complexity C_T (in terms of GOST encryptions) is given by,

$$C_T = 2.(T_1(x) + T_2(x) + T_3(y)) \tag{9.8}$$

since the Type II error is set to $\frac{1}{2}$.

In the rest of this section we study the three variants of GOST of our interest. Using computer simulations we have computed some sufficiently good propagations which can be used in the filtering step for extending the 20-round distinguisher to a 22-round filter. Filtering which will allow us to gain 4 rounds was not achieved so far by our methodology. Table 9.2 presents our best results found so far by our heuristic discovery method.

Table 9.2: Best 1-round transitions in absolute value between general open sets for the the three variants of GOST of our interest

Set	X_j	X'_j	$p(X_j \to X'_j)$	ADV_{filter}
TestParamSet	8070700087800707	8780070780787777	$2^{-5.34}$	0.6
CryptoParamSet	7777777007700700	07700700F77F7777	$2^{-3.73}$	3.2
ISO	7070000080000707	8000070770787780	$2^{-3.27}$	3.6

Based on these transitions we have computed the associated Type I error for each of the three cases and they are found to be $2^{-0.9}$, $2^{-9.51}$ and $2^{-11.62}$ respectively. Table 9.3 presents the complexity for each step of our attack and the complexity of the overall attack for each variant of GOST.

As we observe from Table 9.3, the attack is not good against the GOST variant which uses the set of S-boxes TestParamSet, since its complexity exceeds brute-force. However, there are already plenty of attacks on this variant [37, 34, 36]. Using our technique, we can break the other two variants of GOST which use the sets CryptoParamSet and ISO in time complexity approximately $2^{253.2}$ (slightly faster than brute-force but not significantly) and $2^{245.4}$ GOST encryptions respectively. The ISO version was supposed to be the strongest one and was proposed for standardization.

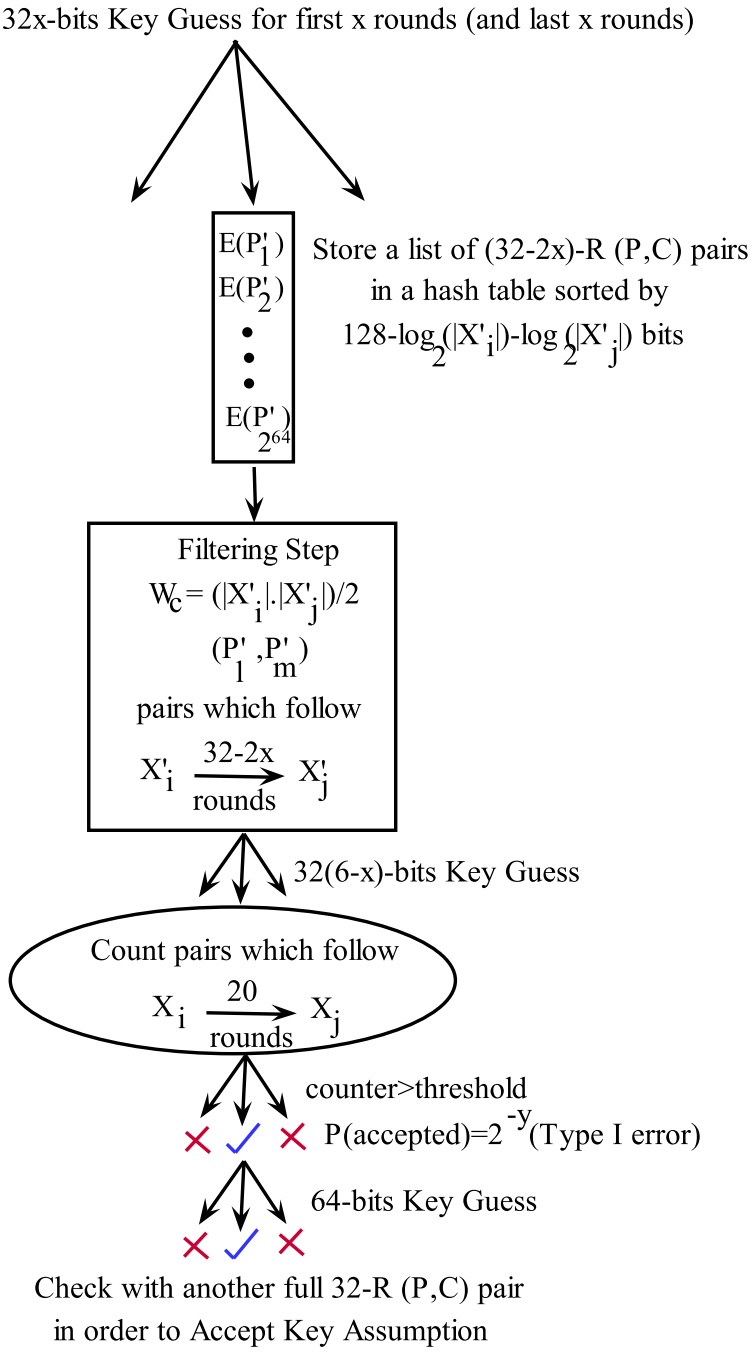

Figure 9.1: Parametric Attack on Full GOST - A parametric attack against full GOST which is essentially a Depth-first search approach combined with an additional filtering step

Table 9.3: T_1, T_2, T_3, C_T values in terms of GOST encryptions

Set	x	T_1	T_2	T_3	C_T
TestParamSet	5	$2^{222.3}$	$2^{231.9}$	$2^{255.1}$	$2^{256.1}$
CryptoParamSet	5	$2^{222.3}$	$2^{248.8}$	$2^{252.2}$	$2^{253.2}$
ISO	5	$2^{222.3}$	$2^{220.7}$	$2^{244.4}$	$2^{245.4}$

Lastly, we would like to emphasize that the most important fact about our attack is its parametric nature, in terms that if better distinguishers or outer transitions for the filtering step are discovered this might lead to significant improvements in the complexity of the attack. In addition, it is a new method for evaluating the security of a cipher against advanced differential attacks. In the next section, we follow exactly the same methodology to develop an attack against a simplified version of GOST with S-boxes replaced by the identity map (abbreviated as GOST-ID).

9.5 Cryptanalysis of GOST-ID

According to Shorin et al, *if the S-boxes are replaced by identity, then GOST is still secure after 6 rounds against LC at a prescribed* 256*-bit level security [115, 114]. Moreover, in the same paper the authors have claimed that 7 rounds are sufficient to protect GOST against DC at* 128*-bit level security.*

In this section, we study this simplified version of GOST with its S-boxes replaced by the identity. Using our discovery method, we discovered a good set of differences with 14 active bits (cf. Result 4), which we later use to construct a distinguisher on more number of rounds and finally convert it to an attack against the full 32 rounds.

Result 4. *The truncated differential* $[A0100700C0700700]$ *with uniform sampling of all differences it allows, produces an element of the same truncated differential set after 8 rounds of GOST with probability about* $2^{-13.7}$ *on average over all possible keys (Sample size used is approximately* 2^{23}*).*

Remark 20. *The best differential property obtained for GOST-ID has 14 active bits again.*

Following the same methodology as before, we have constructed a distinguisher on 26 rounds (Theorem 19), which we use to construct an attack against full 32 rounds (Theorem 20).

Theorem 19. *(26-Round Distinguisher on GOST-ID)*
The following construction

$$[A0100700C0700700]$$
$$\downarrow (8R)$$
$$[A0100700C0700700]$$
$$\downarrow (10R)$$
$$[A0100700C0700700]$$
$$\downarrow (8R)$$
$$[A0100700C0700700]$$

is a 26-round differential property which satisfies the following properties,

1. *If the 26 rounds are replaced by a random permutation, then out of the total* 2^{77} *pairs of plaintexts* (P_i, P_j) *such that* $P_i \oplus P_j \in [A0100700C0700700]$*, we expect on average* 2^{27} *to satisfy also the output difference after 26 rounds.*

2. *Among all input pairs with input difference in the set* $[A0100700C0700700]$*, we expect on average* $2^{30.6} + 2^{27}$ *to have output difference in the same set and satisfy also the intermediate differences due to propagation*

3. *The advantage of the distinguisher is* $2^{17.1}$ *standard deviations*

Proof. Suppose that the 26 rounds of GOST are replaced by a random permutation on 64 bits. The size of the set $[A0100700C0700700]$ is $2^{14.0} - 1$ (excluding the zero difference). Hence, we expect that given $2^{63}.2^{14} = 2^{77}$ pairs (P_i, P_j) with XOR difference in this set, then approximately $2^{77}.\frac{2^{14}}{2^{64}} = 2^{27}$ such pairs will have this output difference at the end of the 26 rounds by accident.

Based on computer simulations we have obtained the following transitional probability for 10 rounds,

$$P(x \in [A0100700C0700700] \rightarrow y \in [A0100700C0700700]) = 2^{-19.0}$$

The sample size used is approximately 2^{28}.

In case of the model of propagation we obtain approximately $2^{30.6}$ such output pairs. This is because out of all 2^{77} pairs which have this input difference, only $2^{77-13.7-19.0-13.7}$ will have such output difference after 26 rounds of GOST and satisfy simultaneously the intermediate differences.

Assume that the middle 10 rounds are replaced by a random permutation, while the first and last 8 rounds are still GOST. Then, we expect approximately $2^{27-13.7-13.7} =$

$2^{-0.4}$ pairs to have this middle-difference property. Thus, in practice the intersection is negligible..

Thus, the number of pairs due to propagation follows a (approximately) Gaussian distribution with mean $\mu_1 = 2^{30.6} + 2^{27}$, while in a random permutation we have a Gaussian with mean $\mu_2 = 2^{27}$ and standard deviation $\sigma_2 = 2^{13.5}$. This corresponds to an advantage of about $2^{17.1}$. □

The 26-round distinguisher described above can be extended to an attack against full GOST-ID as presented in Theorem 20.

Theorem 20. *(A Differential Attack against full GOST-ID)*

The distinguisher construction in Theorem 19 can be extended to an attack against full 32 rounds with time complexity approximately 2^{160} *GOST encryptions given* 2^{64} *known pairs and memory* 2^{64}*.*

Proof. We can attack the full 32 rounds of GOST using the entire codebook as follows.

For all 2^{64} pairs (P, C) and all 2^{96} values of the first three round keys, obtain pairs for the middle 26 rounds $P' = G_{3,k}(P), C' = G_{3,k}(C)$.

For each key out of the 2^{96} candidates we guess, we encrypt and decrypt for the first and last 3 rounds and we count the number of pairs which have both input and output difference as specified by the middle 26 round distinguisher. If the key assumption on 96 bits is wrong we expect approximately 2^{27} pairs, while if it is correct we expect $2^{30.6} + 2^{27}$ pairs. Using this threshold, the probability to reject the correct key is $\frac{1}{2}$.

The associated Type I error is lower than 2^{-256}, which means that with one iteration we are approximately left with one choice for the right key. Thus, either the right key is found or no key is found and the attack has to be repeated for another one time since the Type II error is set at $\frac{1}{2}$.

The computational cost is given by

$$2^{96}.2^{64}.\tfrac{6}{32} \simeq 2^{157.6} \text{ GOST encryptions.}$$

Thus, 96 bits are found in this way and rest $256 - 96 = 160$ by brute force attack which gives complexity 2^{160}. Thus, the overall attack has time complexity approximately about 2^{161} GOST encryptions.

□

10

Conclusion and Further Research

The objective of this thesis is to enhance current advanced differential and to some extent algebraic techniques. The frameworks we suggest involve considering and solving some underlying combinatorial and optimization problems based on the special configurations of each different algorithm. Such optimization problems may arise either from the general structural properties of the cipher or from the specific properties of its underlying components such as the S-boxes.

In the first part, we study computationally hard problems such as efficient MM and combinatorial circuit optimization with respect to several metrics related to cryptology [117, 29, 26, 25, 44]. No analytic algorithms which solve efficiently such problems are known and most of the existing techniques are based on well-chosen ad-hoc heuristics [23]. We propose a 2-step methodology for solving such problems [41, 44]. In the first step, we algebraically encode the problem, then we convert it to its corresponding CNF-SAT form and finally we solve it using dedicated SAT solver software [11]. This SAT-solver based methodology can sometimes lead to provably optimal exact results.

Following this methodology, we have been able to discover new bilinear algorithms *for multiplying two matrices of sufficiently small dimensions over the general non-commutative setting [43]. Additionally, we have been able to obtain* provably optimal *circuit representations with respect to several metrics for the S-boxes of prominent ciphers such as PRESENT and GOST. We have obtained the best known* bit-slice *type implementation of PRESENT S-box with 14 gates [42, 6]. Our technique can be used to compute the MC of whole ciphers, which is one of the four main measures of non-linearity of a cipher [20]. Moreover, we show a proof of concept that MC can be reduced in a cryptanalysis setting. This leads to a new method for algebraic cryptanalysis of ciphers in general which we have tentatively proposed.*

A major theme in this thesis is the study of advanced differential attacks on block ciphers based on the construction of reduced-round distinguishers [50]. The construction of such distinguishers is a highly non-trivial optimization problem. We apply advanced DC on GOST [122]. GOST is a special cipher due to the presence of modulo 2^{32} *addition, which makes the transitional probability to depend also on the sub-key values and thus naive DC fails. However, advanced DC techniques, such as truncated DC were successfully applied and broke ciphers which were believed to be secure against naive DC [84].*

Discovering sufficiently good truncated differential properties is a very complex task since the space of differences is exponentially large. Thus, some ad-hoc heuristics are needed in order speed-up the process [51]. We introduce specific sets of differences, which we name general open sets *and can be seen as a refinement of* truncated differentials *[51, 50].* General Open Sets *(partitioning type) are dictated by the internal structure of GOST and especially the connections between its S-boxes. Based on these sets, we construct distinguishers for up to 20 rounds for different variants of GOST and we discuss a general framework of parametric attacks faster than brute-force against full GOST. All these attacks exploit both* poor diffusion *for limited number of rounds and the* self-similarity *of the cipher due to its weak key schedule. In order to achieve the best possible extensions, a series of optimization problems are solved related to the special configurations of each cipher. We present attacks faster than brute force against 4 variants of GOST, including the latest GOST-ISO [107]. A lot of effort was spent on making all the steps of these attacks as rigorous as possible and to quantify how much the practise differs from the theory. Improving the complexity of such attacks is a future work, as well as the application of similar attacks to other ciphers, such as PRESENT or SIMON [18, 12].*

References

[1] *Wikipedia Entry.* http://en.wikipedia.org/wiki/Matrix_multiplication. *Accessed: 2014-08-19.* 108

[2] *GOST 34.11-94. Information Technology Cryptographic Data Security Hashing Function (In Russian). 1994.* 42, 45

[3] *Martin R. Albrecht. Algorithmic algebraic techniques and their application to block cipher cryptanalysis.* PhD Thesis Dissertation, Royal Holloway, University of London, *2010.* 79

[4] *Martin R. Albrecht and Carlos Cid. Algebraic techniques in differential cryptanalysis.* In Fast Software Encryption, pp. 193-208, Springer Berlin Heidelberg, *2009.* 79

[5] *Martin R. Albrecht, Carlos Cid, Thomas Dullien, Jean-Charles Faugere, and Ludovic Perret. Algebraic precomputations in differential and integral cryptanalysis.* In Information Security and Cryptology, pp. 387-403, Springer Berlin Heidelberg, *2011.* 79

[6] *Martin R. Albrecht, Nicolas T. Courtois, Daniel Hulme, and Guangyan Song. Bit-slice implementation of PRESENT in pure standard C, v1.5. 2011.* bitbucket.org/malb/algebraic_attacks/src/tip/present_bitslice.c. iv, 113, 114, 127, 191

[7] *Martin R. Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers.* In Selected Areas in Cryptography, pp. 1-15, Springer Berlin Heidelberg, *2013.* 138, 139

[8] *Alex Alekseychuk and L. V. Kovalchuk. Towards a theory of security evaluation for GOST-like ciphers against differential and linear cryptanalysis.* Cryptology ePrint Archive, Report 2011/489, *2011.* 158

[9] *A.Tardy-Corfdir and H.Gilbert. A known plaintext attack of FEAl-4 and FEAL-6.* Advanced in Cryptology, CRYPTO 1991, Lecture Notes in Computer Science, Vol. 576, pp. 172-182*, 1991.* 56

[10] *Gregory V. Bard. Algorithms for solving linear and polynomial systems of equations over finite fields to cryptanalysis.* PhD thesis*, 2007.* 77, 78

[11] *Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. 2007.* 77, 78, 100, 111, 112, 119, 132, 191

[12] *Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Lous Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers.* Released by NSA, 19 June 2013*, 2013.* 81, 192

[13] *Mihir Bellare and Philip Rogaway. Introduction to modern cryptography. 2005.* 39

[14] *Guido Bertoni, Joan Daemen, Michael Peeters, and G.V. Assche. The keccak SHA-3 submission.* Submission to NIST(Round 3)*, 2011.* 47

[15] *Eli Biham. New types of cryptanalytic attacks using related keys.* Journal of Cryptology 7.4, pp 229-246.*, 1996.* 54

[16] *Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES.* In Advances in Cryptology, CRYPTO 92, E. F. Brickel, Ed.,vol. 740 of Lecture Notes in Computer Science, pp. 487 496*, 1992.* 60, 160

[17] *Eli Biham and Adi Shamir. Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, ISBN: 0-387-97930-1, 3-540-97930-1*, 1993.* 60

[18] *Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An ultra-lightweight block cipher.* In Cryptographic Hardware and Embedded Systems, CHES 2007, pp. 450-466, Springer Berlin Heidelberg*, 2007.* 100, 125, 192

[19] *Joppe Bos, Marcelo Kaihara, Thorsten Kleinhurg, Arjen Lenstra, and Peter Montgomery. On the security of 1024-bit RSA and 160-bit elliptic curve cryptography.* IACR Cryptology ePrint Archive 2009/389*, 2009.* 14

[20] *Joan Boyar, Magnus Find, and Rene Peralta. Four measures of nonlinearity.* In Algorithms and Complexity, pp. 61-72. Springer Berlin Heidelberg, *2013.* iv, 15, 23, 37, 84, 88, 89, 191

[21] *Joan Boyar, Philip Matthews, and Rene Peralta. On the shortest linear straight-line program for computing linear forms.* In MFCS 2008, pp. 168-179, *2008.* 100, 118

[22] *Joan Boyar and Rene Peralta. Tight bounds for the multiplicative complexity of symmetric functions.* Theoretical Computer Science 396, no. 1, pp. 223-246, *2008.* 113

[23] *Joan Boyar and Rene Peralta. A new combinational logic minimization technique with applications to cryptology.* In Experimental Algorithms, pp. 178-189, Springer Berlin Heidelberg, *2010.* iv, 100, 191

[24] *Joan Boyar and Rene Peralta. A depth-16 circuit for the AES S-box.* IACR Cryptology ePrint Archive, *2011.* http://eprint.iacr.org/2011/332/. 115, 125

[25] *Joan Boyar, Rene Peralta, and Denis Pochuev. On the multiplicative complexity of Boolean functions over the basis.* Theoretical Computer Science 235, no. 1, pp. 43-57, *2000.* 99, 113, 115, 191

[26] *Richard Brent.* Algorithms for matrix multiplication. *Stanford University, 1970.* 99, 100, 101, 110, 191

[27] *Claude Carlet. On the degree, nonlinearity, algebraic thickness, and nonnormality of Boolean functions, with developments on symmetric functions.* IEEE Transactions on Information Theory, *2004.* 88

[28] *Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks.* IBM Journal of Research and Development 38 (3): 243. doi:10.1147/rd.383.0243, *1994.* 19, 60

[29] *Don Coppersmith and Sanjoy Winograd. Matrix multiplication via arithmetic progressions.* Journal of Symbolic Computation, Volume 9:pp 251-280, *1990.* 99, 110, 191

[30] *Nicolas Courtois, Gregory Bard, and Daniel Hulme. A new general-purpose method to multiply 3x3 matrices using only 23 multiplications.* arXiv preprint arXiv:1108.2830, *2011.* iv

[31] *Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback.* In Advances in CryptologyEUROCRYPT 2003, pp. 345-359. Springer Berlin Heidelberg, *2003.* 88, 89

[32] *Nicolas T. Courtois. How fast can algebraic attacks be on block ciphers ?* IACR Cryptology ePrint Archive, *2006.* http://eprint.iacr.org/2006/168/. 19, 25, 26, 100

[33] *Nicolas T. Courtois. CTC2 and fast algebraic attacks on block ciphers revisited.* IACR Cryptology ePrint Archive, *2007.* http://eprint.iacr.org/2007/152/. 27

[34] *Nicolas T. Courtois. Algebraic complexity reduction and cryptanalysis of GOST, 2011.* http://eprint.iacr.org/2011/626/. 34, 132, 133, 145, 146, 185

[35] *Nicolas T. Courtois. Security evaluation of GOST 28147-89 in view of international standardisation, 2011.* http://eprint.iacr.org/2011/211/. 31, 143, 145

[36] *Nicolas T. Courtois. An improved differential attack on full GOST, 2012.* http://eprint.iacr.org/2012/138/. 145, 146, 161, 164, 185

[37] *Nicolas T. Courtois. Low complexity key recovery attacks on GOST block cipher.* In Cryptologia, Volume 37, Issue 1, pp. 1-10, *2013.* 133, 185

[38] *Nicolas T. Courtois. New combined differential and complexity reduction attacks on GOST.* In CECC 2013, Central European Conference on Cryptology, *2013.* 133

[39] *Nicolas T. Courtois and Gregory V. Bard. Algebraic cryptanalysis of the data encryption standard.* In Cryptography and Coding, 11-th IMA Conference, pp.152-169, LNCS 4887, *2007.* 75, 113, 133

[40] *Nicolas T. Courtois, Jerzy A Gawinecki, and Guangyan Song. Contradiction immunity and guess-then-determine attacks on GOST.* Tatra Mountains Mathematical Publications, *53(1):65–79, 2012.* 79

[41] *Nicolas T. Courtois, Daniel Hulme, and Theodosis Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis.* In electronic proceedings of 2nd IMA Conference Mathematics in Defence 2011, *2011.* 191

[42] *Nicolas T. Courtois, Daniel Hulme, and Theodosis Mourouzis. Solving circuit optimisation problems in cryptography and cryptanalysis (extended version).* IACR Cryptology ePrint Archive, *2011.* http://eprint.iacr.org/2011/475/. *iv, 37, 84, 100, 113, 114, 116, 128, 191*

[43] *Nicolas T. Courtois, Daniel Hulme, and Theodosis Mourouzis. Multiplicative complexity and solving generalized Brent equations with SAT solvers.* In COMPUTATION TOOLS 2012, The Third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, pp. 22-27, *2012. 111, 191*

[44] *Nicolas T. Courtois, Daniel Hulme, and Theodosis Mourouzis. Exact logic minimization and multiplicative complexity of concrete algebraic and cryptographic circuits.* To Appear in IARIA Journal: IntSys13v6n34, *2013. iv, 84, 99, 128, 191*

[45] *Nicolas T. Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations.* In Advances in Cryptology, EUROCRYPT 2000, B. Preenel, Ed., vol. 1807 of Lecture Notes in Computer Science, Springer-Verlag, pp. 392407, *2000. 74, 114*

[46] *Nicolas T. Courtois and Michal Misztal. Differential cryptanalysis of GOST.* IACR Cryptology ePrint Archive, *2011. 146, 164*

[47] *Nicolas T. Courtois and Michal Misztal. First differential cryptanalysis of full round 32- round GOST.* In ICICS'11, Beijing, China, pp. 216-227, Springer LNCS 7043, *2011. 145, 164*

[48] *Nicolas T. Courtois and Michal Misztal. Aggregated differentials and cryptanalysis of PP-1 and GOST.* Periodica Mathematica Hungarica 65, no. 2, pp. 177-192, *2012. 145, 161*

[49] *Nicolas T. Courtois and Theodosis Mourouzis. Advanced differential cryptanalysis and GOST cipher.* To Appear in IMA Maths in Defence 2013, *2013. 164*

[50] *Nicolas T. Courtois and Theodosis Mourouzis. Enhanced truncated differential cryptanalysis of GOST.* In SECRYPT 2013, *2013. v, 143, 146, 147, 153, 164, 192*

[51] *Nicolas T. Courtois and Theodosis Mourouzis. Propagation of truncated differentials in GOST.* In SECURWARE 2013, *2013.* v, 160, 161, 164, 192

[52] *Nicolas T. Courtois, Theodosis Mourouzis, Michal Misztal, Jean-Jacques Quisquater, and Guangyan Song. Can GOST be made secure against differential cryptanalysis?* To Appear in Cryptologia Journal, *2013.* x, 160, 161, 163, 164, 209

[53] *Nicolas T. Courtois, Theodosis Mourouzis, Guangyan Song, Pouyan Sepehrdad, and Petr Susil. Combined algebraic and truncated differential cryptanalysis on reduced-round SIMON.* In SECRYPT 2014, *2014.* 81

[54] *Nicolas T. Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations.* In Advances in Cryptology, ASIACRYPT 2002, pp. 267-287. Springer Berlin Heidelberg, *2002.* 26

[55] *Nicolas T. Courtois, Pouyan Sepehrdad, Petr Suil, and Serge Vaudenay. Elimlin algorithm revisited.* In Fast Software Encryption, pp. 306-325, Springer Berlin Heidelberg, *2012.* 74

[56] *Joan Daemen and Vincent Rijmen. The design of Rijndael: AES-the advanced encryption standard. 2002.* 29

[57] *Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers.* IACR Cryptology ePrint Archive, *2005.* http://eprint.iacr.org/2005/212/. 139

[58] *Deepak Dalai, Maitra Subhamoy, and Sarkar Sumanta. Basic theory in construction of Boolean functions with maximum possible annihilator immunity.* Designs, Codes and Cryptography 40, no. 1, pp. 41-58, *2006.* 89

[59] *Ivan Damgard. A design principle for hash functions.* In Advances in Cryptology, CRYPTO89 Proceedings, pp. 416-427. Springer New York, *1990.* 15, 40

[60] *George Danezis. Anonymity and cryptography.* Microsoft Research Report presented at Weworc 2007, *2007.* 4

[61] *Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Virkumar Vazirani. Algorithms. 2006.* 101, 102, 103, 104, 105

[62] *Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved attacks on full GOST.* In Fast Software Encryption, pp. 9-28, Springer Berlin Heidelberg, *2012.* *31*, *145*, *146*

[63] *Vasily Dolmatov. RFC 5830: GOST 28147-89 encryption, decryption and MAC algorithms. 2010.* *iv*, *12*, *29*, *31*, *34*, *36*, *100*

[64] *Orr Dunkelman and Nathan Keller. Linear cryptanalysis of CTC.* IACR Cryptology ePrint Archive, *2006.* `http://eprint.iacr.org/2006/250/`. *27*

[65] *Alan Edelman. Large dense numerical linear algebra in 1993: The parallel computing influence.* International Journal of High Performance Computing Applications 7, no. 2, pp. 113-128, *1993.* *106*

[66] *Jean-Charles Faugere. A new efficient algorithm for computing Gröbner bases (F4).* Journal of pure and applied Algebra, Vol. 139, pp. 61-88, *1999.* *74*

[67] *Horst Feistel. Cryptography and computer privacy.* Scientific American, 228(5), pp. 15-23, *1973.* *22*, *91*

[68] *Aviezri Fraenkel and Yesha Yaacov. Complexity of solving algebraic equations.* Information Processing Letters 10, no. 4, pp. 178-179, *1980.* *74*

[69] *Carsten Fuhs and Peter Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT.* In SAT 2010, Theory and Applications of Satisfiability Testing, Springer LNCS 6175, pp. 71-84, *2010.* *100*, *118*

[70] *Soichi Furuya. Slide attacks with a known-plaintext cryptanalysis.* In Information Security and CryptologyICISC 2001, pp. 214-225, Springer Berlin Heidelberg, *2002.* *34*

[71] *Ian P. Gent and Toby Walsh. The search for satisfaction.* Department of Computer Science University of Strathclyde, Scotland, *1999.* *76*, *78*

[72] *Julio Hernandez and Pedro Isasi. Finding efficient distinguishers for cryptographic mappings, with an application to the block cipher TEA.* In Evolutionary Computation, 2003, CEC'03, The 2003 Congress on, vol. 3, pp. 2189-2193, *2003.* *164*

[73] *John Hopcroft and Jean Musinski. Duality applied to the complexity of matrix multiplication and other bilinear forms.* SIAM Journal on Computing 2, no. 3, pp. 159-173, *1973.* *123*

[74] *Jialin Huang and Xuejia Lai. What is the effective key length for a block cipher: An attack on every block cipher.* Cryptology ePrint Archive, Report 2012/677, *2012.* 14, 55

[75] *Takanori Isobe. A single-key attack on the full GOST block cipher.* In Fast Software Encryption, pp. 290-305, Springer Berlin Heidelberg, *2011.* 31, 143, 145, 146

[76] *Rodney W. Johnson and Aileen M. McLoughlin. Noncommutative bilinear algorithms for 3x3 matrix multiplication.* In SIAM J. Comput., vol. 15 (2), pp.595-603, *1986.* 112, 124, 125

[77] *Antoine Joux. Multicollisions in iterated hash functions. Application to cascaded constructions.* In Advances in Cryptology, CRYPTO 2004, pp. 306-316, Springer Berlin Heidelberg, *2004.* 46

[78] *David Kahn. The codebreakers: The comprehensive history of secret communication from ancient times to the internet.* Scribner, *1996.* 4, 5, 6

[79] *Orhun Kara and Ferhat Karakoc. Fixed points of special type and cryptanalysis of full GOST.* In Cryptology and Network Security, pp. 86-97, Springer Berlin Heidelberg, *2012.* 146

[80] *Jonathan Katz and Yehuda Lindell. Introduction to modern cryptography: Principles and Protocols.* Chapman and Hall/CRC Cryptography and Network Security Series, *2008.* 7, 11, 13, 20, 40, 41

[81] *Thomas Kelly. The myth of skytale.* Cryptologia 22, no. 3, pp. 244-260., *1998.* 5

[82] *Auguste Kerckhoffs. La cryptographie militaire.* Journal des sciences militaires IX, *1883.* 11

[83] *Dmitry Khovratovich and Ivica Nikolic. Rotational cryptanalysis of ARX.* In Fast Software Encryption, pp. 333-346, Springer Berlin Heidelberg, *2010.* 36

[84] *Lars Knudsen. Truncated and higher order differentials.* In Fast Software Encryption, pp. 196-211, Springer Berlin Heidelberg. v, 16, 66, 147, 192

[85] *Lars R. Knudsen. Block Ciphers: Analysis, Design and Applications.* PhD thesis, Department of Computer Science, University of Aarhus, *1994.* 19, 52

[86] *Lars R. Knudsen and Matthew J.B. Robshaw. The block cipher companion.* Springer, *2011.* 3, 21, 46, 59, 63, 66, 67, 68, 69, 71, 138

[87] *Julian D. Laderman. A non-commutative algorithm for multiplying 3x3 matrices using 23 multiplications.* Bull. Amer. Math. Soc. Volume 82, Number 1, pp. 126-128, *1976.* viii, 112, 124, 206

[88] *Xuejia Lai and James L. Massey. Markov ciphers and differential cryptanalysis. 1991.* 38, 62, 63, 138

[89] *Gregor Leander and Axel Poschmann. On the classification of 4 Bit S-boxes.* In Arithmetic of Finite Fields, pp. 159-176, Springer Berlin Heidelberg, *2007.* ix, 23, 25, 84, 93, 94, 163, 205

[90] *Aleksandr Malchik. An English translation of GOST Standard by Aleksandr Malchik with an English preface co-written with Whitfield Diffie. 1994.* 19, 29, 30

[91] *James L. Massey. Cryptography:fundamentals and applications.* 21

[92] *Mitsuru Matsui. Linear cryptanalysis method for DES cipher.* In Advances in Cryptology, EUROCRYPT 93, pp. 386-397. Springer Berlin Heidelberg, *1994.* 56

[93] *Mitsuru Matsui and Yamagishi Atsuhiro. A new method for known plaintext attack of FEAL cipher.* In Advances in Cryptology, Eurocrypt 92, pp. 81-91, Springer Berlin Heidelberg, *1993.* 56

[94] *Florian Mendel, Norbert Pramstaller, and Christian Rechberger. A (second) preimage attack on the GOST hash function.* In Fast Software Encryption, pp. 224-234. Springer Berlin Heidelberg, *2008.* 46

[95] *Florian Mendel, Norbert Pramstaller, Christian Rechberger, Marcin Kontak, and Janusz Szmidt. Cryptanalysis of the GOST hash function.* In Advances in Cryptology, CRYPTO 2008, pp. 162-178, Springer Berlin Heidelberg, *2008.* 44, 46, 48

[96] *Matthew W. Moskewicz, Madigan F. Conor, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver.* In Proceedings of the 38th annual Design Automation Conference, pp. 530-535, *2001.* 78

[97] *John Nash. John Nash letters - National Security Agency. 1955.* 14

[98] *NIST/SEMATECH. 6.3.3.1. counts control charts.* e-Handbook of Statistical Methods. 139

[99] *Kaisa Nyberg. Differentially uniform mappings for cryptography.* Eurocrypt 1993, LNCS 765, pp. 55-64, Springer, *1994.* 83

[100] *National Bureau of Standards. Data Encryption Standard.* Federal Information Processing Standard (FIPS), *1977.* 22, 29

[101] *Kurepkin Popov and Serguei Leontiev. Additional cryptographic algorithms for use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 algorithms. draft-popov-cryptopro-cpalgs-03.txt.* 31, 35

[102] *Axel Poschmann, San Ling, and Huaxiong Wang. 256 bit standardized crypto for 650 GE GOST revisited.* In CHES 2010, LNCS 6225, pp. 219-233, *2010.* iv, 30, 31

[103] *Bart Preneel. Hash functions and MAC algorithms based on block ciphers.* IMA Int. Conf. 1997, pp. 270-282, *1997.* 41

[104] *Emmanuel Prouff, Christophe Giraud, and Sacuteebastien Aumonier. Provably secure S-box implementation based on Fourier Transform.* In CHES 2006, Springer LNCS 4249, pp: 216-230, *2006.* 100, 114, 127

[105] *S. Rickmann. Logic friday (version 1.1.3), (computer software). 2011.* 128

[106] *Vincent Rijmen. Cryptanalysis and design of iterated block ciphers.* PhD Thesis, K.U.Leuven, *1997.* 83

[107] *Vladimir Rudskoy and Andrey Chmora. Working draft for ISO/IEC 1st WD of AMd1/18033-3. in Russian block cipher GOST, ISO/IEC JTC 1/SC 27 N9423, 2011-01-14. 2011.* iv, 36, 175, 192

[108] *Markku-Juhani Saarinen. A chosen key attack against the secret s-boxes of GOST.* Unpublished manuscript, *1998.* 34

[109] *Markku-Juhani Saarinen. Cryptographic analysis of all 4-bit S-boxes.* In Selected Areas in Cryptography, pp. 118-133, Springer Berlin Heidelberg, *2012.* 94

[110] *Bruce Schneier. Applied cryptography, second edition. In* John Wiley and Sons, *1996.* 3, 7, 10, 11, 12, 28, 29, 30, 31, 35, 40, 53, 54

[111] *Haruki Seki and Toshinobu Kaneko. Differential cryptanalysis of reduced rounds of GOST.* In Selected Areas in Cryptography, pp. 315-323, Springer Berlin Heidelberg, *2001.* 145

[112] *Pouyan Sepehrdad. Statistical and algebraic cryptanalysis of lightweight and ultra-lightweight symmetric primitives.* PhD dissertation, Ecole Polytechnique Federale de Lausanne, *2012.* 74

[113] *Claude E. Shannon. Communication theory of secrecy systems.* Bell System Technical Journal 28, *1949.* 12, 20, 73

[114] *Vitaly Shorin, Vadim Jelezniakov, and Ernst Gabidulin. Security of algorithm GOST 28147-89.* In Abstracts of XLIII MIPT Science Conference, *2000.* 145, 187

[115] *Vitaly Shorin, Vadim Jelezniakov, and Ernst Gabidulin. Linear and differential cryptanalysis of Russian GOST.* Electronic Notes in Discrete Mathematics 6, pp. 538-547, *2001.* 145, 187

[116] *Gustavus Simmons. Symmetric and asymmetric encryption.* ACM Computing Surveys (CSUR) 11, no. 4, pp. 305-330, *1979.* 10

[117] *Andrew James Stothers. On the complexity of matrix multiplication.* PhD Dissertation, The University of Edinburgh, *2010.* 99, 110, 191

[118] *Serge Vaudenay. On the need for multipermutations: Cryptanalysis of MD4 and SAFER.* In FSE 1994, pp. 286-29, LNCS 1008, *1995.* 83

[119] *David Wagner. A generalized birthday problem.* In Advances in cryptology CRYPTO 2002, pp. 288-304, Springer Berlin Heidelberg, *2002.* 45, 46

[120] *Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd.* In Proceedings of the 44th symposium on Theory of Computing, pp. 887-898, *2012.* 110

[121] *Shmuel Winograd. A new algorithm for inner product.* In SIAM J. Comput., vol. 15 (2), pp.595-603, *1968.* 108

[122] *I.A. Zabotin, G.P. Glazkov, and V.B Isaeva. Cryptographic protection for information processing systems, government standard of the USSR, GOST 28147-89, government committee of the USSR for standards. 1989.* 29, 30, 192

Appendix A

APPENDIX

A.1 Classification of S-boxes Based on Affine Equivalence (AE)

Table A.1: The 16 generators G_i $(1 \leq i \leq 16)$ for each class under AE [89]

G0	0,1,2,13,4,7,15,6,8,11,12,9,3,14,10,5
G1	0,1,2,13,4,7,15,6,8,11,14,3,5,9,10,12
G2	0,1,2,13,4,7,15,6,8,11,14,3,10,12,5,9
G3	0,1,2,13,4,7,15,6,8,12,5,3,10,14,11,9
G4	0,1,2,13,4,7,15,6,8,12,9,11,10,14,5,3
G5	0,1,2,13,4,7,15,6,8,12,11,9,19,14,3,5
G6	0,1,2,13,4,7,15,6,8,12,11,9,10,14,5,3
G7	0,1,2,13,4,7,15,6,8,12,14,11,10,9,3,5
G8	0,1,2,13,4,7,15,6,8,14,9,5,10,11,3,12
G9	0,1,2,13,4,7,15,6,8,14,11,3,5,9,10,12
G10	0,1,2,13,4,7,15,6,8,14,11,5,10,9,3,12
G11	0,1,2,13,4,7,15,6,8,14,11,10,5,9,12,3
G12	0,1,2,13,4,7,15,6,8,14,11,10,9,3,12,5
G13	0,1,2,13,4,7,15,6,8,14,12,9,5,11,10,3
G14	0,1,2,13,4,7,15,6,8,14,12,11,3,9,5,10
G15	0,1,2,13,4,7,15,6,8,14,12,11,9,3,10,5

A.2 MM Problem and SAT-solvers

```
P01 := (a_1_1-a_1_2-a_1_3+a_2_1-a_2_2-a_3_2-a_3_3) * (-b_2_2);
P02 := (a_1_1+a_2_1) * (b_1_2+b_2_2);
P03 := (a_2_2) * (b_1_1-b_1_2+b_2_1-b_2_2-b_2_3+b_3_1-b_3_3);
P04 := (-a_1_1-a_2_1+a_2_2) * (-b_1_1+b_1_2+b_2_2);
P05 := (-a_2_1+a_2_2) * (-b_1_1+b_1_2);
P06 := (a_1_1) * (-b_1_1);
P07 := (a_1_1+a_3_1+a_3_2) * (b_1_1-b_1_3+b_2_3);
P08 := (a_1_1+a_3_1) * (-b_1_3+b_2_3);
P09 := (a_3_1+a_3_2) * (b_1_1-b_1_3);
P10 := (a_1_1+a_1_2-a_1_3-a_2_2+a_2_3+a_3_1+a_3_2) * (b_2_3);
P11 := (a_3_2) * (-b_1_1+b_1_3+b_2_1-b_2_2-b_2_3-b_3_1+b_3_2);
P12 := (a_1_3+a_3_2+a_3_3) * (b_2_2+b_3_1-b_3_2);
P13 := (a_1_3+a_3_3) * (-b_2_2+b_3_2);
P14 := (a_1_3) * (b_3_1);
P15 := (-a_3_2-a_3_3) * (-b_3_1+b_3_2);
P16 := (a_1_3+a_2_2-a_2_3) * (b_2_3-b_3_1+b_3_3);
P17 := (-a_1_3+a_2_3) * (b_2_3+b_3_3);
P18 := (a_2_2-a_2_3) * (b_3_1-b_3_3);
P19 := (a_1_2) * (b_2_1);
P20 := (a_2_3) * (b_3_2);
P21 := (a_2_1) * (b_1_3);
P22 := (a_3_1) * (b_1_2);
P23 := (a_3_3) * (b_3_3);
expand(-P06+P14+P19-a_1_1*b_1_1-a_1_2*b_2_1-a_1_3*b_3_1);
expand(P01-P04+P05-P06-P12+P14+P15-a_1_1*b_1_2-a_1_2*b_2_2-a_1_3*b_3_2);
expand(-P06-P07+P09+P10+P14+P16+P18-a_1_1*b_1_3-a_1_2*b_2_3-a_1_3*b_3_3);
expand(P02+P03+P04+P06+P14+P16+P17-a_2_1*b_1_1-a_2_2*b_2_1-a_2_3*b_3_1);
expand(P02+P04-P05+P06+P20-a_2_1*b_1_2-a_2_2*b_2_2-a_2_3*b_3_2);
expand(P14+P16+P17+P18+P21-a_2_1*b_1_3-a_2_2*b_2_3-a_2_3*b_3_3);
expand(P06+P07-P08+P11+P12+P13-P14-a_3_1*b_1_1-a_3_2*b_2_1-a_3_3*b_3_1);
expand(P12+P13-P14-P15+P22-a_3_1*b_1_2-a_3_2*b_2_2-a_3_3*b_3_2);
expand(P06+P07-P08-P09+P23-a_3_1*b_1_3-a_3_2*b_2_3-a_3_3*b_3_3);
```

Figure A.1: Laderman's Tri-Linear algorithm for MM of two 3×3 matrices [87]

```
a_1_1_01=0   b_1_1_01=0   c_1_1_01=0
a_1_1_02=1   b_1_1_02=1   c_1_1_02=1
a_1_1_03=0   b_1_1_03=1   c_1_1_03=0
a_1_1_04=0   b_1_1_04=0   c_1_1_04=-1
a_1_1_05=-1  b_1_1_05=0   c_1_1_05=0
a_1_1_06=0   b_1_1_06=0   c_1_1_06=0
a_1_1_07=0   b_1_1_07=0   c_1_1_07=0
a_1_2_01=1   b_1_2_01=1   c_1_2_01=0
a_1_2_02=0   b_1_2_02=0   c_1_2_02=0
a_1_2_03=0   b_1_2_03=1   c_1_2_03=0
a_1_2_04=1   b_1_2_04=0   c_1_2_04=1
a_1_2_05=1   b_1_2_05=1   c_1_2_05=-1
a_1_2_06=0   b_1_2_06=0   c_1_2_06=-1
a_1_2_07=-1  b_1_2_07=1   c_1_2_07=-1
a_2_1_01=0   b_2_1_01=0   c_2_1_01=-1
a_2_1_02=0   b_2_1_02=0   c_2_1_02=0
a_2_1_03=1   b_2_1_03=1   c_2_1_03=1
a_2_1_04=0   b_2_1_04=-1  c_2_1_04=1
a_2_1_05=-1  b_2_1_05=0   c_2_1_05=0
a_2_1_06=-1  b_2_1_06=1   c_2_1_06=0
a_2_1_07=1   b_2_1_07=1   c_2_1_07=-1
a_2_2_01=1   b_2_2_01=1   c_2_2_01=1
a_2_2_02=0   b_2_2_02=0   c_2_2_02=0
a_2_2_03=0   b_2_2_03=1   c_2_2_03=0
a_2_2_04=0   b_2_2_04=0   c_2_2_04=-1
a_2_2_05=1   b_2_2_05=0   c_2_2_05=0
a_2_2_06=1   b_2_2_06=1   c_2_2_06=1
a_2_2_07=-1  b_2_2_07=1   c_2_2_07=1
```

Figure A.2: (Heuristic) Lifting from $\mathbb{F}_2$ to $\mathbb{F}_4$ for MM of two 2×2 matrices

A.3 GOST's Sets of S-boxes

Table A.2: Gost28147-TestParamSet set of S-boxes

S-boxes	Gost28147-TestParamSet
1	12,6,5,2,11,0,9,13,3,14,7,10,15,4,1,8
2	9,11,12,0,3,6,7,5,4,8,14,15,1,10,2,13
3	8,15,6,11,1,9,12,5,13,3,7,10,0,14,2,4
4	3,14,5,9,6,8,0,13,10,11,7,12,2,1,15,4
5	14,9,11,2,5,15,7,1,0,13,12,6,10,4,3,8
6	13,8,14,12,7,3,9,10,1,5,2,4,6,15,0,11
7	12,9,15,14,8,1,3,10,2,7,4,13,6,0,11,5
8	4,2,15,5,9,1,0,8,14,3,11,12,13,7,10,6

Table A.3: GostR3411-94-SberbankHashParamset set of S-boxes

S-boxes	GostR3411-94-SberbankHashParamset
1	8,7,3,12,14,13,2,0,11,10,4,1,5,15,9,6
2	12,8,9,13,3,4,1,5,7,6,2,15,10,0,11,14
3	10,5,12,8,13,2,1,9,11,7,14,15,0,3,6,4
4	10,12,11,0,6,2,14,8,15,5,7,13,3,9,4,1
5	5,3,15,14,10,0,11,8,7,1,13,9,2,4,12,6
6	3,9,4,0,14,7,8,15,5,13,6,10,11,2,1,12
7	9,4,0,15,7,13,10,11,2,3,5,6,14,1,12,8
8	3,6,10,14,2,11,1,9,13,12,8,15,4,5,0,7

Table A.4: CryptoProParamSet set of S-boxes

S-boxes	GostR3411-94-CryptoProParamSet
1	10,4,5,6,8,1,3,7,13,12,14,0,9,2,11,15
2	5,15,4,0,2,13,11,9,1,7,6,3,12,14,10,8
3	7,15,12,14,9,4,1,0,3,11,5,2,6,10,8,13
4	4,10,7,12,0,15,2,8,14,1,6,5,13,11,9,3
5	7,6,4,11,9,12,2,10,1,8,0,14,15,13,3,5
6	7,6,2,4,13,9,15,0,10,1,5,11,8,14,12,3
7	13,14,4,1,7,0,5,10,3,12,8,15,6,2,9,11
8	1,3,10,9,5,11,4,15,8,6,7,14,13,0,2,12

Table A.5: Affine equivalence (AE) of all known GOST S-Boxes (and their inverses) [52]

S-box Set Name	S1	S2	S3	S4	S5	S6	S7	S8
GostR3411-94-TestParamSet	36	02	03	04		06	08	
-inverses		02	03	04		06		08
GostR3411-94-CryptoProParamSet			Lu1	14	G_{10}		G_8	
-inverses			Lu1	14	G_{10}		G_8	
Gost28147-TestParamSet	21	21			25			28
-inverses	21	21			25			28
Gost28147-CryptoProParamSetA	31	32	33	G_8	35	36	37	38
-inverses	31	32	33	G_8			37	38
Gost28147-CryptoProParamSetB	G_{13}	G_{13}	G_{13}	G_{11}	G_7	G_7	G_{11}	G_6
-inverses	G_{13}	G_{13}	G_{13}	G_{11}	G_7	G_7	G_{11}	G_6
Gost28147-CryptoProParamSetC	G_7	G_4	G_6	G_{13}	G_{13}	G_6	G_{11}	G_{13}
-inverses	G_7	G_4	G_6	G_{13}	G_{13}	G_6	G_{11}	G_{13}
Gost28147-CryptoProParamSetD	G_{13}	G_{13}	G_{13}	G_4	G_{12}	G_4	G_{13}	G_7
-inverses	G_{13}	G_{13}	G_{13}	G_4	G_{12}	G_4	G_{13}	G_7
GostR3411-94-SberbankHash			74	75	76		78	
-inverses			74	75	76		76	
GOST ISO 18033-3 proposal	G_9	G_9	G_9	G_9	G_9	G_9	G_9	G_9
-inverses	G_9	G_9	G_9	G_9	G_9	G_9	G_9	G_9