NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

MSc THESIS

Security of lightweight cryptographic algorithms

Paul A. Psomiadis

Supervisor: Konstantinos Limniotis, Adjunct Faculty Member

ATHENS

February 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ασφάλεια Lightweight Κρυπτογραφικών Αλγορίθμων

Παύλος Α. Ψωμιάδης

Επιβλέπων: Κωνσταντίνος Λιμνιώτης, Διδάσκων εκτός Τμήματος

ΑΘΗΝΑ

Φεβρουάριος 2020

MSc THESIS

Security of lightweight cryptographic algorithms

Paul. A Psomiadis

S.N.: EN3180008

ATHEN

February 2020

Διπλωματική Εργασία

Ασφάλεια lightweight κρυπτογραφικών αλγορίθμων

Παύλος Α. Ψωμιάδης

Α.Μ.: ΕΝ3180008

ΑΘΗΝΑ

Φεβρουάριος 2020

ABSTRACT

This thesis studies the lightweight cryptographic algorithms, focusing on specific security features. In particular, many lightweight algorithms are being analyzed, evaluated and classified into several categories including but not limited to block/stream ciphers, either being authenticated or not. This categorization consists of algorithms that are in the progress of standardization, competing in the NIST (National Institution of Standards and Technology) Lightweight Crypto Standardization process. Next, emphasis will be given on the Boolean functions that these Lightweight cryptographic algorithms utilize, with the aim to calculate the cryptographic properties of such functions towards evaluating the resistance of these algorithms against several cryptanalytic attacks. Our analysis illustrates that there is no cryptographic Boolean function satisfying all the cryptographic properties and, thus, further research is needed in order to evaluate whether such vulnerabilities of underlying Boolean functions can actually be exploited in order to mount a cryptanalytic attack.

SUBJECT AREA: Lightweight cryptography

KEYWORDS: cryptography, lightweight, asymmetric, symmetric, IoT

ΠΕΡΙΛΗΨΗ

Η διπλωματική εργασία μελετά τους lightweight κρυπτογραφικούς αλγορίθμους, εστιάζοντας σε συγκεκριμένα χαρακτηριστικά ασφάλειας. Πιο συγκεκριμένα, θα αναλυθούν, θα αξιολογηθούν και θα ταξινομηθούν σε διάφορες κατηγορίες, όπως αν είναι block ή stream ciphers και αν είναι authenticated ή όχι, κάποιοι lightweight αλγόριθμοι. Αυτή η ταξινόμηση αφορά αλγόριθμους που βρίσκονται σε διαδικασία προτυποποίησης και συμμετέχουν στο διαγωνισμό του NIST (National Institution of Standards and Technology), Lightweight Crypto. Στη συνέχεια, θα δοθεί έμφαση στις Boolean Functions που χρησιμοποιούν αυτοί οι lightweight κρυπτογραφικοί αλγόριθμοι, με σκοπό να υπολογιστούν οι κρυπτογραφικές ιδιότητες των συναρτήσεων αυτών και να αξιολογηθεί η ανθεκτικότητα αυτών των αλγορίθμων ενάντια σε κρυπταναλυτικές επιθέσεις. Η ανάλυσή μας δείχνει πως δεν υπάρχει καμία Boolean function που να ικανοποιεί όλες τις κρυπτογραφικές ιδιότητες και έτσι απαιτείται περαιτέρω έρευνα για να διευκρινιστεί αν οι ευπάθειες αυτές των Boolean functions μπορούν να χρησιμοποιηθούν για την διεξαγωγή κρυπταναλυτικής επίθεσης.

Θεματική Περιοχή: Lightweight κρυπτογραφία

Λέξεις κλειδιά: κρυπτογραφία, lightweight, ασύμμετρη, συμμετρική, IoT

CONTENTS

LIST OF PICTURES

1. INTRODUCTION

Cryptography constitutes a main pillar for network and communication security, since it provides the means to achieve specific security goals, such as data confidentiality and authentication. Therefore, cryptography is being met in a plethora of applications (Internet, e-mail, e-banking, e-health services, cloud computing, Internet of Things etc.). Although there exist several cryptographic standards, they cannot be directly used to any application due to the fact that there is a high diversity between the specific requirements that each application sets.

This thesis is about Lightweight cryptographic algorithms, which forms a specific type of cryptographic algorithms focusing on devices with several constraints, and the level of security they can provide. Before the term Lightweight is defined and analyzed (Chapter 4), cryptography needs to be clarified in great detail as it is the basis for the whole thesis idea and purpose. This chapter will analyze the general term of cryptography and its objectives, along with the implementations, applications and the need for its utilization over the years. Furthermore, the two main categories of encryption algorithms, symmetric and asymmetric key encryption will be evaluated and lastly, a more detailed elaboration of the thesis will be given, consisting of the purpose and the scope of this dissertation.

1.1 What is Cryptography?

Cryptography comes from the word “krypto” that in ancient greek means hidden and “graphein” that translates into write. There are a lot of definitions, all similar to another that comes to the conclusion that cryptography is the science of securely transmitting a message, information and generally any communication over an insecure channel that third parties, also called adversaries have access to [1] [2].

This practice provides a method of securing information and communication, in such a way that only the indented recipient will be able to read and process the initial data.

This is achieved through the construction of rules and protocols based on mathematical concepts, principles and calculations called cryptographic algorithms.
Actually, cryptography is derived from a more general term called cryptology. Cryptology is the science that studies and practices methods and techniques of secure communication and storage of data in a secret and illegible form.

Cryptology is divided into two areas, the cryptography described above and Cryptanalysis. Cryptanalysis is used to discover breaches and vulnerabilities in cryptographic algorithms and security systems, so as to gain access to contents and knowledge of encrypted messages and data.

1.2 Objectives of Cryptography

Nowadays modern cryptography concerns itself with four objectives described below:

Confidentiality

The information and data are only read and received by the intended authorized recipients and can't be comprehended and decrypted by unauthorized parties [4].
Confidentiality ensures that the message is encoded in order to conceal it, so the sender encrypts the message (plaintext) to create a ciphertext that is transmitted. The receiver, who possesses the cryptographic key, decrypts the ciphertext into the original plaintext.

Data Integrity

The information and data sent can't be modified in storage or during the transmission between the source and destination in a way that the alteration is not detectable. Data Integrity assures that the message received is exactly the same as the one sent by the sender. This may be accomplished, e.g., with the use of Hash functions like SHA256 that create a unique digest from the original message, which is sent along with the message.

The recipient runs the same Hash Function on the decrypted message and compares it to the digest sent. If the two digests are identical then Data Integrity is achieved, otherwise the message has been modified.

Non-Repudiation

The creator/sender of the information can't deny in the future the creation and transmission of the data [3]. Generally, it is the assurance that the sender can't repudiate the validity of the message transmitted. This is accomplished with the use of digital signatures (especially used in online transactions) and Message Authentication Codes, which are basically Hash Functions containing a key. It should be noted that such cryptographic primitives also ensure the integrity of the information, in a more robust manner than a simple hash function as mentioned above.

Authentication

The sender and the receiver can confirm each other's identity without having a personal knowledge of it. Basically, it is the assurance that the sender of the information is who they say they are. To achieve this, digital certificates can be used, along with appropriate digital signatures.

Nowadays, most cryptographic algorithms support authenticated encryption (AE) or authenticated encryption with associated data (AEAD). This basically means that both confidentiality and authenticity of the data is achieved. When referring to the AEAD scheme it is assumed that the recipient is able to verify the integrity of both the encrypted and the decrypted message. To clarify this even more, the associated data (AD) are used to bind a ciphertext to the context that it is supposed to be. So, any attempt to place a valid ciphertext along with a different context is detectable and can be rejected.

1.3 Applications of Cryptography

Nowadays, cryptography is excessively used in many types of applications, by everyone and on a daily basis. Apart from the most common use, which is the communication between systems, encryption is also used by web browser communication with web servers and also between email clients and email servers [6]. These are just a few services everyone uses and contain cryptographic algorithms in our days.
Considering the rapid development of Networking over the last years, more and more sensitive and vital data are stored and transmitted over a network or the Internet. As a result, malicious parties attempt to sniff and steal credentials and data.

Cryptography is a method of preventing such actions, like identity theft, consequently it is considered as extremely important, especially when it comes to financial transactions over the Internet, storing personal data on a cloud or accessing a social media account with a username and a password.

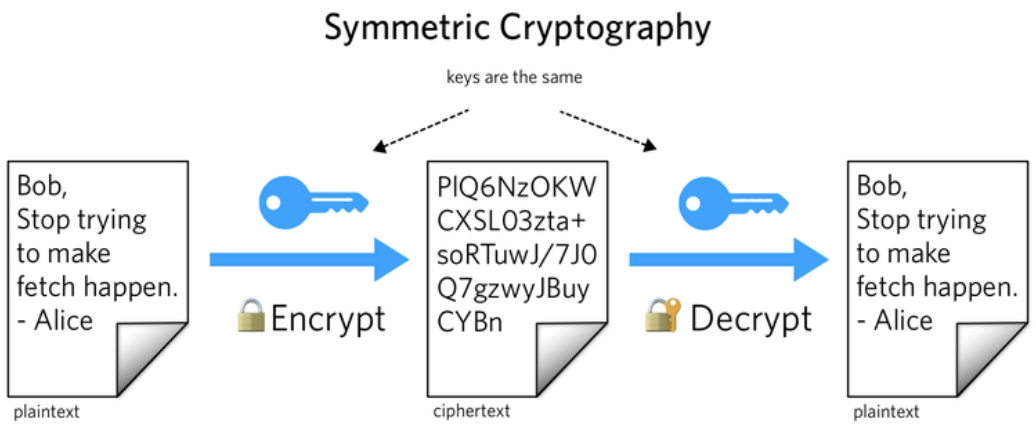
The need for strong cryptographic algorithms is very high and the design of them is challenging, especially by taking into consideration that the processing power of computers increases day by day and malicious parties tend to find new vulnerabilities and breaches in security systems all the time.

1.4 Symmetric and Asymmetric Cryptography

Cryptography is divided into two (2) main categories regarding the keys used for the encryption and the decryption, which are the symmetric (private) key cryptography and the asymmetric (public) key cryptography.

Symmetric Cryptography

This method of encryption is the simplest one, as there is only one secret key used for both ends of the communication and it is used for both encryption and decryption of the message [5] [6]. Symmetric encryption is also very fast, compared to asymmetric encryption, as the keys do not have to be very long, however there is an issue with how the key is shared prior the communication.
Some examples of symmetric key algorithms are the AES, 3DES, DES, RC4, RC5, etc. However, the last three (3) are not considered secure nowadays. The most widely known and used today is the AES algorithm consisting of AES-128, AES-192 and AES-256 depending on the key length.

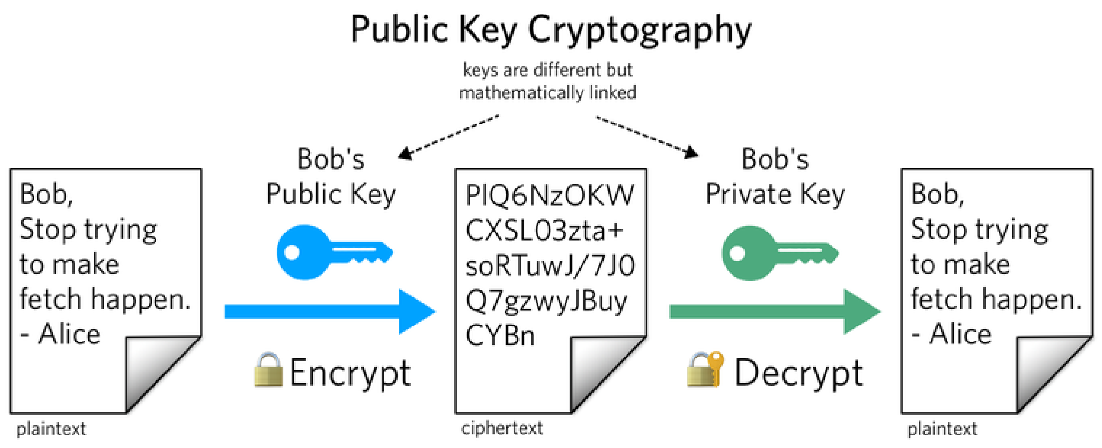


Picture 1: Symmetric Cryptography

Asymmetric Cryptography

In this method of encryption two keys are used, one public and one private key to strengthen the security. These two keys are linked mathematically, but it is almost impossible to find one from another [7]. Each party holds two keys, a public one, which is known and can be seen by anyone in the network, so no security is required for this key and one private, which is secret and known only by the owner.

If a message is encrypted with the public key of a person then it can only be decrypted by the private key of that same person. The converse also holds and is being appropriately used for constructing digital signatures for authentication. Some well-known public key algorithms are: Elliptic Curve Techniques, RSA, DSA and El Gamal.


Public Key Cryptography
keys are different but mathematically linked
Bob's Public Key
Bob's Private Key
Bob, Stop trying to make fetch happen. - Alice
plaintext
Encrypt
PIQ6NzOKW CXSL03zta+ soRTuwJ/7J0 Q7gzwyJBuy CYBn
ciphertext
Decrypt
Bob, Stop trying to make fetch happen. - Alice
plaintext

Picture 2: Asymmetric Cryptography

1.5 Introduction to Lightweight Cryptography

As mentioned above, this dissertation is about Lightweight cryptography. Generally, cryptographic algorithms are divided into two categories based on the target devices.

The first category is the Conventional cryptography, which targets and performs well in devices such as smart phones, tablets, servers, desktops, etc [35]. This type of devices can process and store great amounts of data effectively and fast as well, due to the high-performance computing they are capable of. Furthermore, when referring to this kind of cryptography the energy consumption caused by the cryptographic algorithms is negligible.

This is not the case with Lightweight cryptography, the target devices that fall into this category include sensors, RFID tags, embedded systems, etc. Such devices do not have the computational power of a smartphone or a desktop.

Thus, the cryptographic algorithms should be adjusted and designed to suit the requirements and needs of these devices. This is accomplished with the use of Lightweight cryptography.

The significance of this kind of cryptography becomes even greater considering the concept of Internet of Things (IoT). IoT devices have specific constraints in terms of computing capability and power consumption; for example, such devices could be sensors that collect data from their environment and transfer this data to remote locations for further computations and processing. This data needs to be protected and encrypted, as it can be the target of cyber attacks, especially when referring to automotive and healthcare applications. As a result, since conventional cryptography cannot be implemented in IoT devices, Lightweight cryptographic algorithms will serve this purpose.

More details about Lightweight cryptography will be given in a later chapter.

1.6 About the Thesis

The goal and purpose of the thesis is to analyze the security of Lightweight cryptographic algorithms based on the Boolean functions they utilize.

To get to this point, some terms will be explained and analyzed regarding the symmetric cryptography, types of cryptographic attacks, Lightweight cryptography, Boolean functions and the cryptographic properties of them, etc.

Afterwards, many Lightweight cryptographic algorithms will be analyzed, studied, classified based on specific criteria and evaluated considering the level of security they can offer. The focus will be on algorithms that have taken part in the NIST standardization process for Lightweight cryptographic algorithms. More specifically, some attacks on these algorithms will be examined, along with the properties of the cryptographic Boolean functions that the chosen algorithms use.

The properties of these Boolean functions will be computed and evaluated using SageMath, an open-source mathematical software system.

1.7 Thesis Overview

The rest of the thesis will follow the structure described below.

Chapter 2: Analysis of symmetric cryptography, reasons why it is preferred for Lightweight applications, dissection and comparison of block and stream ciphers, mention of known block and stream ciphers

Chapter 3: Classification of the different types of cryptographic attacks, detailed examination of the cryptographic Boolean functions, the properties they hold and their correlation with the aforementioned attacks

Chapter 4: Clarification of Lightweight cryptography, challenges and trade-offs in their design and implementation, connection with the IoT (Internet of Things) concept, mention of popular Lightweight cryptographic algorithms and the NIST (National Institute of Standards and Technology) Lightweight Crypto Standardization Process

Chapter 5: Brief analysis of the SageMath Mathematical Software that will be used for the computation of the Boolean functions cryptographic properties, evaluation of these properties regarding the Boolean functions used in the Grain-128 AEAD and the TinyJambu algorithms participating in the NIST competition, simulation of the S-Box used in the Skinny-AEAD algorithm and analysis of it in 8 district Boolean functions using Python

Finally, some conclusions are going to be drawn based on the results extracted from the practical part in Chapter 5.

2. SYMMETRIC CRYPTOGRAPHIC ALGORITHMS

This thesis will focus only on Symmetric encryption algorithms. The reason is that they are proved to be more efficient in Lightweight cryptography for various reasons, which are going to be explained later in this section. As mentioned in the previous chapter, in this kind of cryptography only one key is used for both the sender and the receiver. The key serves both the encryption and decryption process.

In this chapter the two types of Symmetric Ciphers will be analyzed, the Block and Stream Ciphers. More specifically, the functionality of them will be clarified, pointing out their different functions and structure and the advantages and disadvantages of each one as well. Moreover, the most known of such ciphers will be mentioned, along with some details for each one.

2.1 Why Symmetric Cryptography?

Symmetric cryptography is proved to be more efficient and more suitable for Lightweight cryptographic algorithms for many reasons.

First of all, since the asymmetric ciphers use two keys instead of one, which is the case in symmetric ciphers, they are certainly more complex and difficult to implement and they also utilize more resources that is a huge issue especially for IoT devices that are considered constrained devices and require Lightweight algorithms to operate [8]. However, it is a true fact that having one key (private key cryptography) brings up an issue, which is the secure exchange of this single key between the two parties that want to communicate.

In public key encryption algorithms, no such issue comes up, since the private key is never sent across the network as it is associated with the public key, which is considered as a drawback and will be explained later.

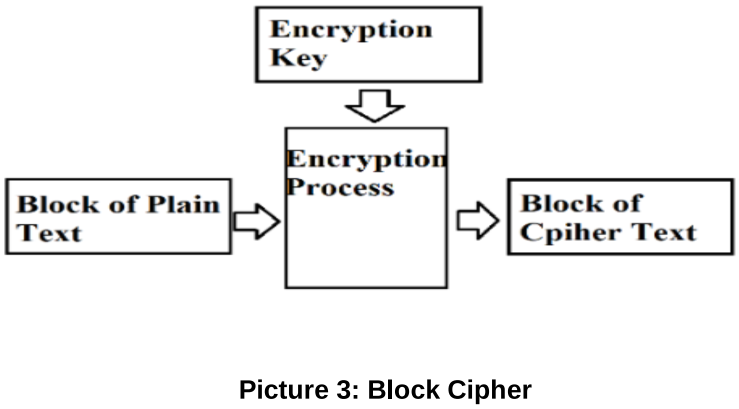
Nevertheless, since the focus here is the Lightweight cryptography, symmetric algorithms are the main cryptographic method used when it comes to constrained devices. These algorithms are easier to implement, require less key size than asymmetric, utilize fewer resources carrying low overhead and they are also considered secure, as long as the key is kept secret of course.

2.2 Block and Stream Ciphers

Symmetric cryptography is divided into two categories, the Block and the Stream ciphers. Generally, both of these ciphers are just two different ways of converting a plaintext (message) into a ciphertext (encrypted message).

Block Ciphers

Block Ciphers in comparison to Stream Ciphers convert the message into the ciphertext by processing one block at a time [9]. A block typically consists of 64, 128 or 256 bits. The message is divided into pieces with equal size to the block size and afterwards they are encrypted one by one. So, for instance, a block cipher will process and encrypt a 128-bit block at a time for the whole message. If the plaintext in some cases is shorter than the block size, padding techniques are used to fill in the gap.

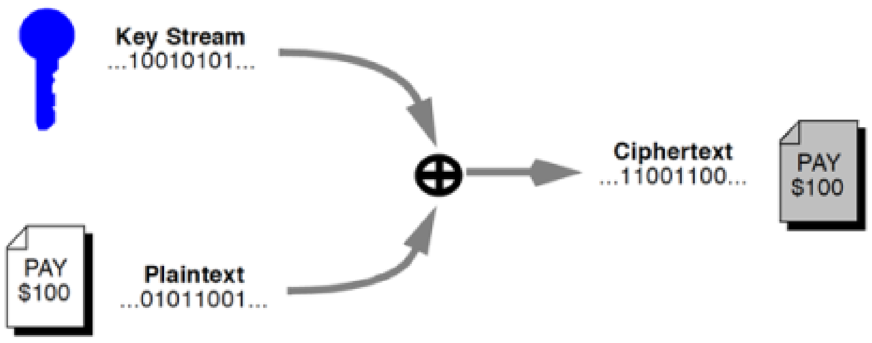


Picture 3: Block Cipher

Stream Cipher

Stream Ciphers on the other hand, convert the plaintext into the ciphertext bit by bit or byte by byte [10]. In particular it uses a pseudorandom bit generator, which basically generates keystream of bits or bytes that are XORed with the plaintext providing the ciphertext. The security of this cipher is extremely depended on the randomness of the keystream generator and the fact that the same key should never be used again. Stream Ciphers purpose is to simulate the function of the One-Time Pad, which is a theoretical Stream Cipher that achieves perfect secrecy. One-Time Pad is supposed to use a completely random key generator, which does not exist in practice and the key size is equal to the size of the message.

This is not feasible, considering that the message could be huge, even Gigabytes long, so the use and the distribution of such key would be impossible. Consequently, the key repetition seems to be unavoidable and perfect secrecy can't be accomplished, but a satisfying level of security can be achieved.



Picture 4: Stream Cipher

2.3 Known Block Ciphers Algorithms

DES (Data Encryption Standard)

This Block Cipher was the most popular in the world and was used by many industries all around the globe. Today it is being discussed, mostly for historical reasons. It was developed in 1970 when it was considered secure and the main standard for encrypting data [12] [13]. However, nowadays it is insecure and vulnerable to brute-force and other cryptanalytic attacks, especially due to the short key it utilizes. The key is 64 bits long but 8 bits are parity bits (error detecting code), so it is actually just 56 bits long, something that makes DES easy to crack, considering the computational power of computers today.

3DES (Triple Data Encryption Algorithm)

As the name implies 3DES applies the DES cipher algorithm three (3) times to each data block. In this algorithm three (3) DES keys, each one of 56 bits without the parity ones are used [15]. The first key encrypts the plaintext, the second decrypts and the third one encrypts as well, while the decryption process is the exact reverse procedure.

The key length now is increased to 168 bits (three times 56 bits); a variation of the algorithm utilizes a 112-bit key as the first key is the same with the third one. Although 3DES is not vulnerable to brute-force attacks, vulnerabilities of it were found to some chosen and known plaintext attacks (analyzed in a later chapter). Furthermore, due to the short block size it uses (64 bits) it is also vulnerable to block collision attacks, when it is applied to encrypt large amounts of data using the same key [14]. Very recently, NIST announced that 3DES should be considered as deprecated (that is it can be used under the assumption that some risks are acceptable), but it is expected to be officially disallowed after 2023.

AES (Advanced Encryption Standard)

Also known as Rijndael, it is divided into three categories based on the key length, which are: AES-128, AES-192 and AES-256, whereas the block size is 128 bits regardless of the key length [16] [17]. Established by NIST (National Institute of Standards) in 2001, AES is now chosen to protect and encrypt sensitive information and data worldwide. It is considered to be the successor of the DES algorithm, and is now also a standard for encrypting data. The encryption process is pretty complicated, consisting of a lot of stages like: Byte substitution, Mix columns, Shift rows and Add round key, whereas the Decryption process is the same but with the reversed order.

RC5 (Rivest Cipher)

Invented by Ronald Rivest in 1994, this Block Cipher, unlike the others ones, has various block sizes and key sizes [18]. The block size can be 32, 64 or 128 bits and the key size can be from 0 up to 2040 bits. Additionally, the number of rounds can be from 0 up to 255. It consists of some modular additions and exclusive OR (XOR) operations. It is vulnerable to a differential attack using numerous chosen plaintexts, where the inputs differences can also affect the outputs.

Blowfish

Designed in 1993 by Bruce Schneier, Blowfish has a block size of 64 bits and the key can be from 0 up to 448 bits [19]. It was developed as an alternative for the DES algorithm and unlike other algorithms that are proprietary and are kept secret; Blowfish is public and can be seen and used by anyone. Twofish, also designed by Schneier is the successor of Blowish and it contains a 128-bit block along with key up to 256 bits long.

2.4 Known Stream Ciphers Algorithms

RC4 (Rivest Cipher 4)

Unlike RC5, RC4 is a Stream Cipher that is well-known for its simplicity and speed. The key it uses can be from 40 up to 2048 bits [20]. The key input is a pseudorandom bit generator that creates a byte number, which is the output of the generator and it is called the keystream. The keystream is then XORed with the plaintext byte by byte to produce the ciphertext. It is impossible to predict the keystream without having knowledge of the key.

However, there are known vulnerabilities of the RC4 algorithm. These breaches occur when the beginning of the output key-stream is not discarded and when the keys used are non-random or related to each other.

Salsa20 and ChaCha

Developed by Daniel J.Bernstein these two Stream Ciphers are very similar to each other. The first one to be developed is the Salsa20 in 2005, whereas the ChaCha was created 3 years later in 2008 [21].

Both of them are built on a pseudorandom function based on ARX (add-rotate-xor) operations, which makes them fast and cheap in hardware and software requirements. The key sizes are 128 or 256 bits for both of them.

The most recent version of the Transport Layer Security (TLS) protocol – i.e. TLS 1.3 – has replaced the stream cipher RC4 by ChaCha.

2.5 Comparison of Block and Stream Ciphers

It is not an easy task to decide which method is better for the Lightweight cryptography. Both sides have some advantages and disadvantages and the answer is not clear for sure.

First of all, Stream Ciphers are generally faster than Block Ciphers, due to the fact that in the second category each block needs to be processed, one by one in order to be encrypted, which is not the case in Stream Ciphers where only one bit or byte is processed at a time [11] [22]. As a result, Block Ciphers require more memory allocation, since they have to work on bigger chunks of data and sometimes, they have to continue the operation from previous blocks as well. On the contrary, Stream Ciphers process at most a byte at a time, so they have relatively low memory requirements and as a result they are cheaper to implement in constrained devices like embedded systems and IoT devices and more general in Lightweight cryptographic algorithms.

Nevertheless, Stream Ciphers are more difficult to develop and design effectively and they are vulnerable depending on the usage. The pseudorandom keystream generator needs to have also very strict requirements as it has to be close to the One-Time Pad, be unpredictable and as mush random as possible. Furthermore, Stream Ciphers do not offer integrity protection and authentication, whereas some Block Ciphers depending on the mode they use are able to provide integrity together with confidentiality.

Additionally, because of the fact that Block Ciphers encrypt a whole block at a time and most of them have feedback modes, they are prone to adding noise in the transmission that could alter the data, so the rest of the transmission will not be the appropriate for the algorithm. Stream Ciphers do not face such problem as the bits or bytes are encrypted separately from the other data and most of the times there are solutions in case of connection issues.

To sum up, Stream Ciphers are more efficient in cases where the amount of data to be transferred is not known or in continuous network streams like live video streaming, whereas the Block Ciphers are more suitable in cases where the amount of data to be transmitted is known, like a certain file for instance.

3. ATTACKS IN CRYPTOGRAPHIC ALGORITHMS

There are several different kinds of cryptographic attacks and they are divided into various categories regarding the purpose of each attack and the cryptographic algorithm it intends to break. Cryptanalysis must not be confused with cryptographic attacks. The reason is that despite the fact that they both aim to crack a cryptographic algorithm and discover its breaches and vulnerabilities; cryptanalysis purpose is not to gain access to confidential and sensitive information but to correct and strengthen the security of the algorithm. In this chapter, several kinds of cryptographic attacks will be introduced and analyzed in detail, along with the purpose and the functional procedure of each attack. Additionally, measures and ways to prevent and protect from these attacks will be examined. Moreover, cryptographic Boolean functions are going to be introduced as well, which are fundamental tools in symmetric cryptographic algorithms. The cryptographic properties of these functions play a huge role on the mitigation of such attacks.

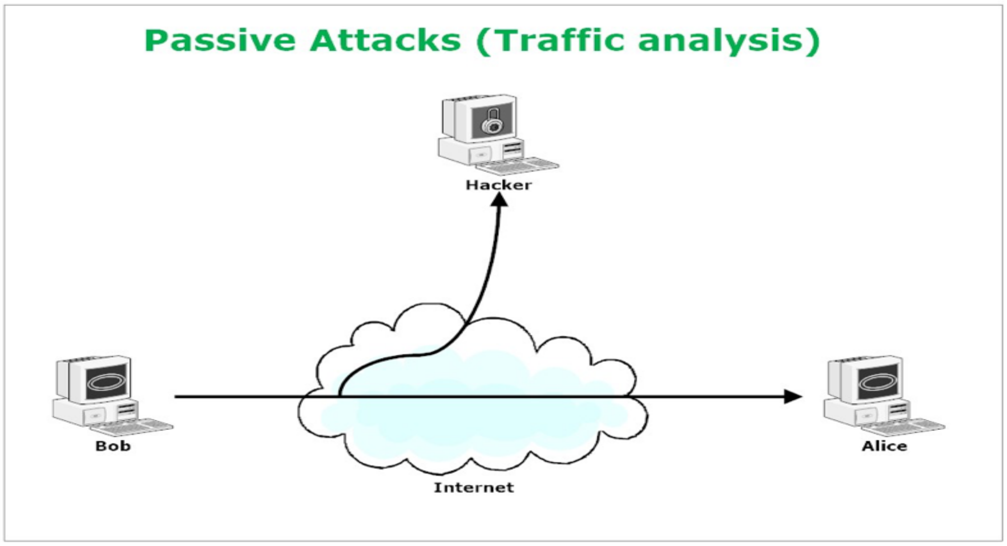
3.1 Passive and Active Attacks

All attacks are categorized regarding the action that the attacker performs. So, the attacks can be either passive or active.

Passive Attacks

This kind of attacks do not alter or affect at any other way the information and they do not cause any issue to the communication channel [23].

The main goal here is to acquire unauthorized access to sensitive and confidential information and data. Passive attacks are often called as stealing Information. What really makes this attack harmful is the fact that most of the times the owner is not aware that an unauthorized person has knowledge of the owner’s data. For instance, an attacker could intercept and eavesdrop a communication channel and gain knowledge to confidential information and neither the sender nor the receiver could figure that out.

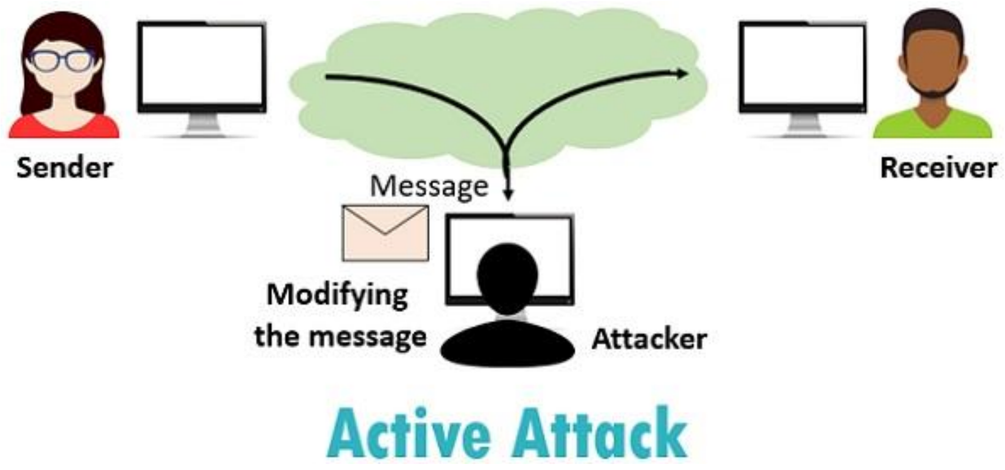


Picture 5: Passive Attacks

Active Attacks

In this type of attacks, the attacker is able to process the information and alter it in many different ways. More specifically, the attacker could change specific fields of the data like the originator name, the timestamp and generally modifying the information in an unauthorized way.

Moreover, unauthorized deletion of data, initiation of unintended transmission of information or data and lastly, denial of access to data by legitimate users the so-called denial of service (DoS) attack are also examples of Active attacks.



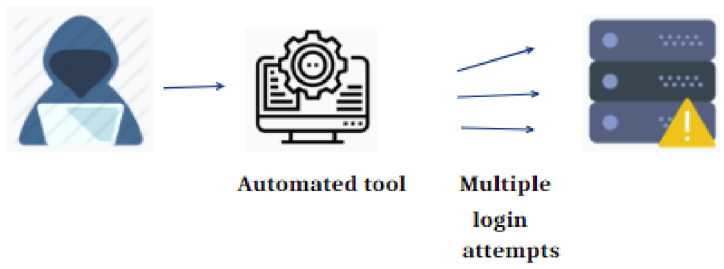
Picture 6: Example of an Active Attack

3.2 Different Types of Cryptographic Attacks

Brute Force

In this type of attack the attacker submits a lot of passwords or passphrases, hoping that eventually the right one will be submitted [24]. Also called exhaustive key search, this attack is also considered a cryptanalytic attack that is used to achieve the decryption of any ciphertext. Brute-force speed is extremely depended on the length of the password, so checking all the possible short passwords might be fast, but this is not the case for long passwords. In the latter case another type of attack is used called dictionary attack. This is reasonable, since in an exhaustive key search attack the adversary submits all the possible passphrases and the more the length of the password the more possible passwords exist.

This attack is often used when it is impossible to discover any other breaches and weaknesses of the algorithm. To protect from brute-force attacks the size of the key needs to be at least 112 bits, as NIST guidance mentioned in 2015.

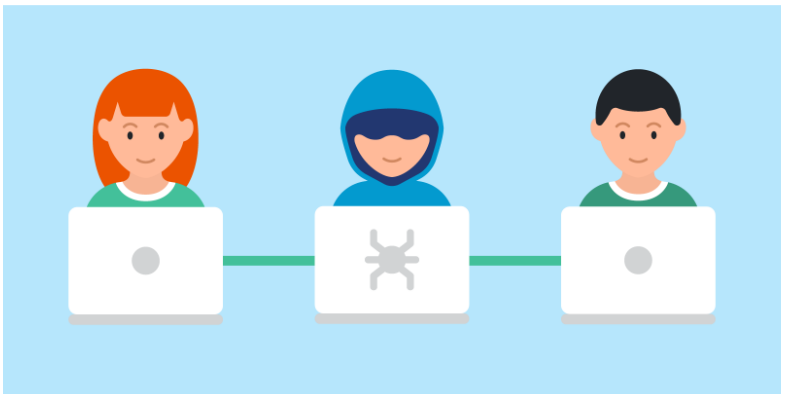


Picture 7: Brute-Force Attack

Man-in-the-Middle

In this type of attack the adversary stands in the middle between the two (2) communication parties and is able to intercept the message of one party to another and possibly alter it or just read it [25]. For example, the adversary could eavesdrop and act as a relay, making the sender and the receiver believe that they are communicating with each other, while in fact the attacker controls the whole communication by initiating independent connections. This is possible if the adversary is able to intercept all the messages and even inject new ones to the channel. This attack is very common and frequently seen in unencrypted wireless access points, the so-called WI-FI. The success of the Man-in-the-Middle attack is depended on whether or not the attacker can impersonate both end users at the edge of the communication channel.

This is difficult, since most cryptographic protocols provide authentication, which means that the receiver can verify if the sender is who they claim to be.

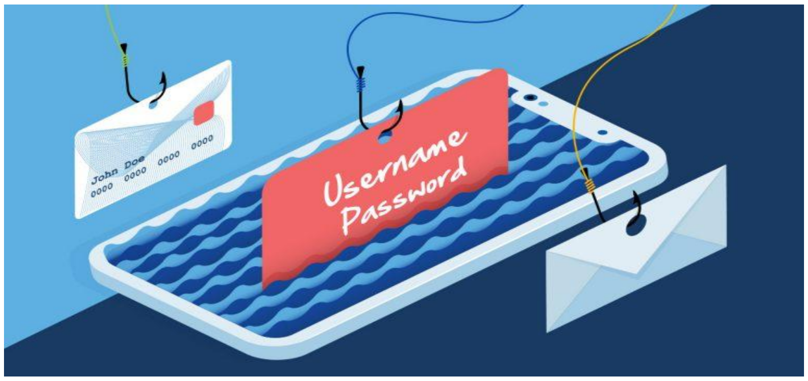


Picture 8: Man-in-the-Middle Attack

Social Engineering Attacks

This type of attacks does not require any cryptanalytic or computer knowledge, since it is based on psychological manipulation of people, with the purpose of performing specific actions or revealing sensitive data [26]. This attack could happen in one or more steps. The adversary often performs some research regarding the victim, trying to gain some knowledge of the victim’s background, so as to discover weaknesses or breaches to plan and perform the attack. Moreover, sometimes the attacker aims to gain the trust of the victim, in order to acquire knowledge to sensitive information or even grant access to vital resources. Social Engineering, in comparison to other types of attacks, relies only on human error, rather than the vulnerabilities of a cryptographic and generally a security system.

The training and raise of awareness of the personnel is important for the prevention of Social Engineering attacks.

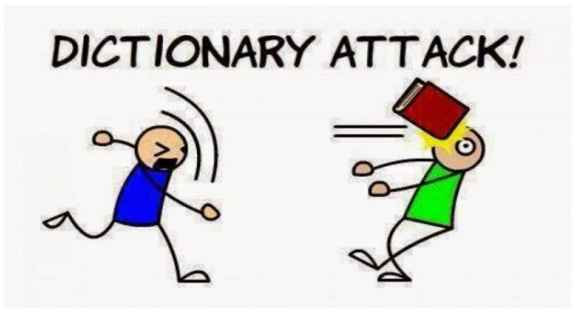


Picture 9: Social Engineering Attack

Dictionary Attack

This attack is quite similar to the brute-force attack with the difference that the adversary builds a dictionary instead of just trying all the possible passphrases [29]. There are various forms of this attack, including creating a dictionary of plaintext and ciphertext pairs over the time and then when a ciphertext is sniffed the attacker searches the dictionary to find the corresponding plaintext. Another form is the use of a wordlist as a dictionary, where the adversary tries all the words inside the dictionary that contains possible passwords and passphrases.

To be immune against these attacks cryptographic algorithms need to add randomness to the encryption process. Thus, even if the same message is sent two times the ciphertext will not be the same.

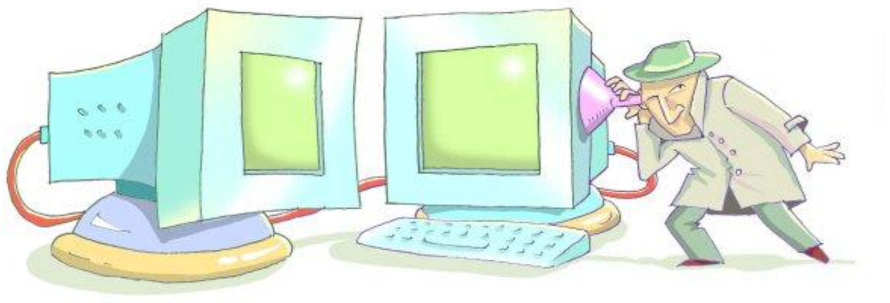


Picture 10: Dictionary Attack

Side Channel Attacks

This kind of attack is totally based on the implementation details of the cryptographic algorithm. These details are related to timing information, power consumption, radiation emissions and even sounds generated [27]. Despite the fact that the majority of cryptographic attacks are software-based, Side Channel Attacks can be both software and hardware based. The adversary attempting such attack collects information during the cryptographic operations and with the knowledge of such information they perform reverse engineering to crack the cryptographic algorithm. Consequently, this means that most of the times the attacker needs to have access to the hardware where the algorithm is running. This is not something difficult when it comes to Lightweight cryptography and devices like sensors, embedded systems, and RFID tags. Usually, such devices are in isolated places with no physical security, so the access to them is not impossible.

There are many countermeasures against this attack, including changing the time of the key transmission or encryption to confuse the adversary and filtering the power line conditioning to prevent power-monitoring acts.



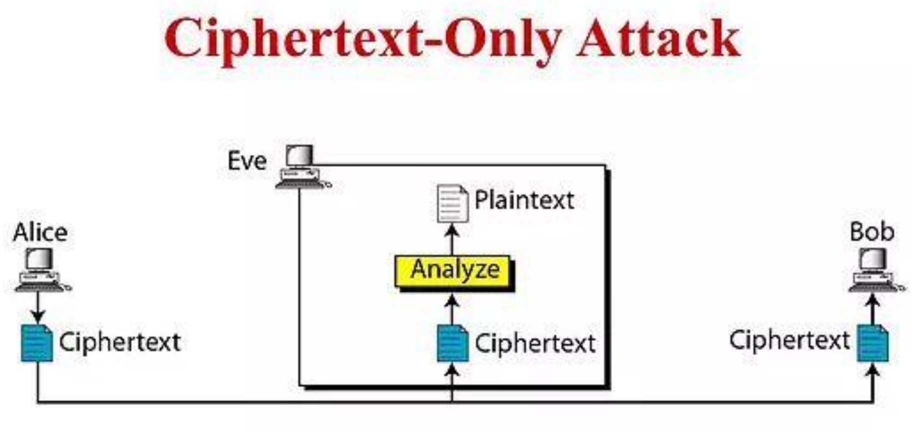
Picture 11: Side-Channel Attacks

Another way to classify and categorize attacks rests with the supposed capabilities that the attacker have. The types of attacks under this classification are presented below.

Ciphertext Only Attacks (COA)

Here the attacker possesses multiple ciphertexts, but without the corresponding plaintexts. This attack becomes effective when the corresponding plaintext can be extracted from one or more ciphertexts [28]. Additionally, sometimes the encryption key can be discovered by this attack. In practice however, the adversary performing this attack have also some knowledge about the plaintext. This information could be the language that the plaintext is written or the foreseeable statistical distribution of the characters in it. Modern Ciphers are immune to such attacks, as COA is considered to be the first step of a cryptanalysis and taking into consideration the confusion and diffusion properties it is improbable to be successful. These properties state that ciphertext and symmetric key correlation must be as complex as possible and changing even one bit of the plaintext must have a huge impact on the corresponding ciphertext.

As a result, if these properties are satisfied this type of attack becomes less likely to be effective.

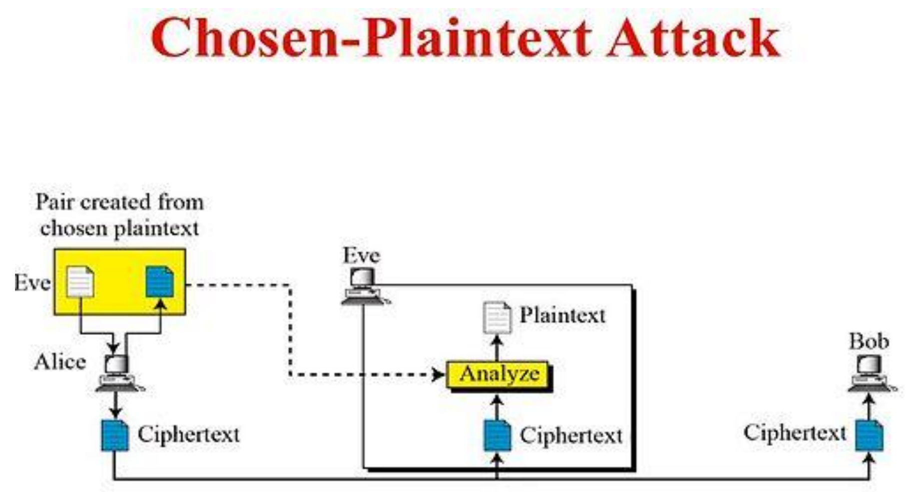


Picture 12: Ciphertext Only Attacks (COA)

Chosen Plaintext Attack (CPA)

The adversary here has free access to the encryption process and can create any ciphertext from any plaintext of his choice [23]. So basically, the attacker can have any desirable pair of plaintext-ciphertext. This makes the process of finding the encryption key easier, as the attacker can gain more knowledge of the encryption operation, the more pairs of messages and ciphertexts created. Despite the fact that this attack constitutes a bigger threat in public key cryptographic algorithms than private key ones, since the public key is known and can be used by anyone, there are cases where the adversary has access to a symmetric cryptographic system and can attempt this kind of attack.

As mentioned previously, if an algorithm is designed to meet properties like confusion and diffusion it can become immune to this attack, because if these properties are fulfilled it does not matter how many pairs an adversary can create, as the discovery of the symmetric key will still be impossible.

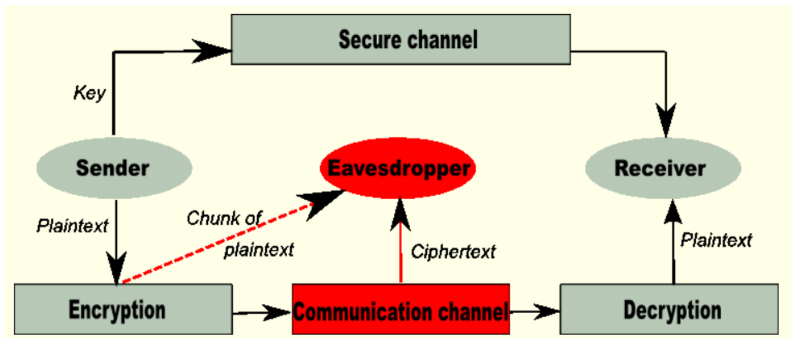


Picture 13: Chosen-Plaintext Attack (CPA)

Known Plaintext Attack (KPA)

This attack is quite similar to the previous one. Here the attacker knows the plaintext that the sender has sent and the corresponding ciphertext [23]. The goal of the adversary is to gain information by taking advantage of the ciphertext-plaintext pairs they have. This could result to the discovery of the encryption key or other information for the algorithm as well. The difference with the chosen-plaintext attack is that the plaintext is not chosen by the attacker but the sender of the message.

This attack becomes effective against simple and not secure ciphers, however since in Lightweight cryptography the security is not the first priority, due to the limitations that exists in constrained environments, caution and measurements must be taken to prevent KPA from breaking the algorithm.



Picture 14: Known-Plaintext Attack (KPA)

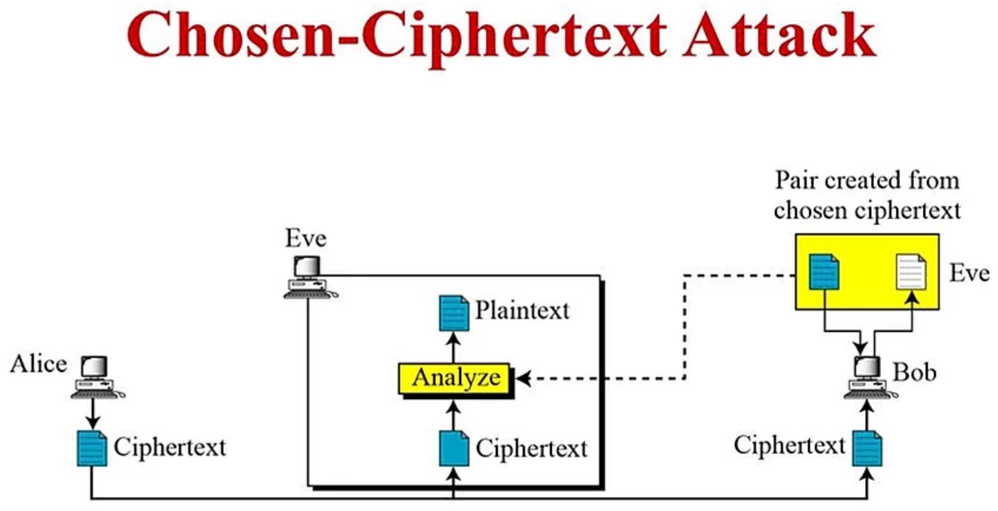
Chosen Ciphertext Attack (CCA)

In this type of attack the adversary or the cryptanalyst have the ability to analyze any chosen ciphertexts along with the corresponding plaintexts. The goal is to gain the secret key or as much information as possible for the attacked cryptographic system.

This attack holds with the assumption that the attacker can make the victim decrypt any encrypted message and send it to him. The more decrypted ciphertexts the attack owns the more information is gained for the system and thus it is more likely to break it.

Despite the fact that Chosen Ciphertext Attacks are most of the times used for breaking public key encryption systems, it must be taken into consideration for Lightweight cryptography, as an adversary e.g. could control a network of sensors to receive all the decrypted messages with the purpose of guessing the secret key.

Early versions of the RSA cipher were vulnerable to this kind of attack.



Picture 15: Chosen Ciphertext Attack (CCA)

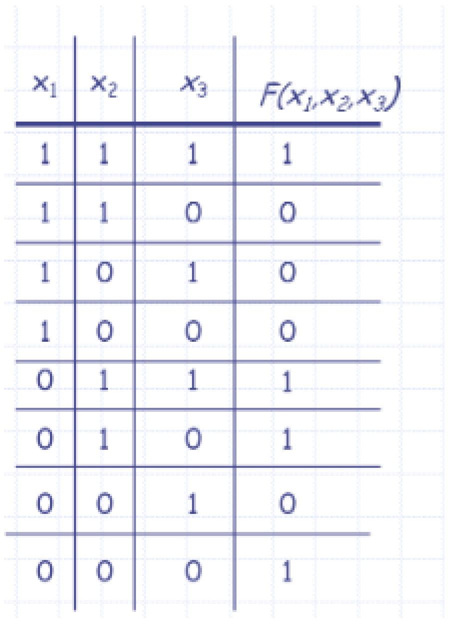
3.3 Cryptographic Boolean Function

Cryptographic Boolean Functions are logical functions that are described by an algebraic expression, which consists of binary variables zeros and ones (0s and 1s) and logical operation symbols (XOR, AND, OR, etc.) [30]. These functions can also be expressed with a truth table. They are also presented with a number of inputs, for example k (x_0, x_1, …, x_k) and one output, which can be either one or zero (0 or 1) depending on the input.

An example of a Boolean Function is presented below.

$$F(x_1,x_2,x_3) = \overline{x}_1x_2+\overline{x}_1\overline{x}_3+x_1x_2x_3$$

Picture 16: Algebraic Expression of Boolean Function



x_1	x_2	x_3	$F(x_1,x_2,x_3)$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	1

Picture 17: Truth Table of Boolean Function

Boolean functions play a vital role in the design of Symmetric key algorithms for both Stream and Block ciphers. XOR is a logical operation that is widely used in Cryptography and thus it is the dominant operation used in cryptographic Boolean functions.

Most of the attacks described in the previous section are related with the properties of the Boolean functions used in the cryptographic algorithms and consequently such functions have a huge effect in the security of the algorithm as well.

3.4 Properties of Cryptographic Boolean Functions

Balancedness

This property states that the truth table of a cryptographic Function needs to strike a balance between ones and zeros, meaning that the numbers should be equal [31]. To make this clearer, for a uniformly random input string of bits, the probability of getting a one (1) is 0.5, while the same accounts for zero (0). If this property does not hold and the probability of getting a zero (0) for example is much higher than 0.5, then the cryptographic algorithm will probably be vulnerable to correlation attack, which is a type of Known Plaintext Attacks described in a previous sub-chapter. The Boolean Function shown in Picture 16 and 17 is a balanced Boolean Function, which is obvious by looking at the truth table.

Nonlinearity

Nonlinearity is equal to the minimum number of the truth table outputs that if modified the function is transformed into a linear function [30]. This number of the different outputs in the truth table is also called Hamming Distance or just distance between two functions.

So, the nonlinearity is equal to the distance between the function examined and the closest (minimum distance) linear function. Linear function's degree is equal to one (1). The degree of a function is the number of variables that are used to the biggest product of the function's algebraic expression. For instance, the function: $f(x) = x0 * x1 * x2 + x0 + x4 * x2$ has a degree of two (2), whereas the function: $g(x) = x0 + x1$ has a degree of one (1) and as a result it is a linear function.

Cryptographic Boolean functions need to have high nonlinearity, in order to provide resistance against specific cryptanalytic attacks.

To compute the value of this property, the Boolean Function should be compared with all the possible linear functions having the same number of variables and check the linear function with the least Hamming distance. The higher the nonlinearity of a Boolean Function the more resistant it is to linear attacks.

The bent function is related to this property. A Boolean function is called bent if the nonlinearity of it is equal to $2^{n-1} - 2^{n/2-1}$, where n is the number of the variables in the function. This is the maximum possible value for the nonlinearity that any Boolean function can attain. Bent functions are determined only for even value of n. If n is an odd number, then the maximum possible value that the nonlinearity may have, is still not known. The previous formula becomes approximately $2^{n-1} - 2^{(n-1)/2-1}$ and function with this value for nonlinearity are of cryptographic importance.

Correlation Immunity

A Boolean Function satisfies this property if the outputs have no relation and are uncorrelated with some subsets of its inputs [32]. More specifically, a Boolean function is characterized to be correlation-immune of order m if every subset of m or less variables in its inputs is statistically independent of the output values. As a result, if a function with low-order correlation immunity is used in a cryptographic algorithm then this algorithm is more vulnerable to the so-called correlation attack than another one that uses a Boolean function with high-order correlation immunity.

Algebraic Immunity

As the name implies, this property measures the durability of a Boolean function or the cipher using the function, against the so-called algebraic attacks [33]. The Algebraic attack is a cryptanalysis technique that attempts to crack a cipher by expressing it as a system of equations, using some known data and eventually solving it to discover the key.

In order for a Boolean Function f to be Algebraic immune there must not be another function g of low degree such that $f * g = 0$ or $(f \oplus 1) * g = 0$. The algebraic immunity is equal to the minimum degree of the function that satisfies this property. The Boolean functions need to have a high algebraic immunity, so as to not be vulnerable to algebraic attacks. The maximum possible value that the algebraic immunity may have is n/2, where n is the number of variables in the function. In case that n is an odd number the value of the property is rounded up to the closest whole number. So, for a Boolean function to be considered algebraic immune the value of the algebraic immunity needs to be equal or very close to n/2.

4. LIGHTWEIGHT CRYPTOGRAPHIC ALGORITHMS

In this chapter the main subject of this thesis will be examined, which is the Lightweight cryptographic algorithms. At first, a more detailed analysis will be given about the term, the importance and the development of it will be evaluated, along with its correlation with the Internet of Things (IoT) concept, which is rapidly developing and growing day by day. Furthermore, the challenges of their design will be examined that is a significant issue and is the reason for the constraints that exist in Lightweight cryptography. Lastly, the NIST Lightweight algorithm standardization competition will be mentioned and summarized as well.

4.1 What is Lightweight Cryptography?

To begin with, cryptography is divided into two classes regarding the target devices. These two classifications are the Conventional and the Lightweight cryptography. In order to comprehend and analyze Lightweight cryptography efficiently the other category must be examined as well [34] [35].

Conventional cryptography refers to the common term and that’s why it is also just called cryptography. It targets devices such as smartphones tablets laptop, desktops, etc. Such devices generally have high computational power, Gigabytes of memory, no limitations of disk capacity and other similar constraints.

On the other hand, Lightweight cryptographic algorithms target devices like embedded systems, RFID devices and sensor networks. These kinds of devices have a lot of constraints, including the memory, energy consumption, processing speed, which is a vital problem especially in real-time applications, the implementation cost and so forth.

Additionally, most of the microcontrollers used in such devices have limited random-access memory (RAM) and read-only memory, something that restricts even more the capabilities and the flexibilities of the possible algorithms that can be applied.

Consequently, all these restrictions make the design of the cryptographic algorithms aiming such devices difficult and more challenging than conventional cryptography.

4.2 Challenges of Lightweight Cryptography

As a further matter, there are also some tradeoffs that exist in the design of Lightweight cryptographic algorithms. They are between the performance, resources and the security level that needs to be achieved. Resources are also referred as cost. These tradeoffs that result from the restrictions mentioned above are actually the reasons that the design and implementation of such algorithms is so challenging and difficult to be done.

Performance

The performance can be separated into three (3) terms that are the power and energy consumption, the latency and the throughput.

The power consumption is measured in Watts and it actually represents the amount of power required to run the circuit. The lower the energy or power consumption the better for the algorithm. This metric is extremely vital, especially for devices that gain power and energy from their surroundings (e.g. solar energy), devices using a battery and having a specific amount of energy stored. Also, in many cases it is difficult to recharge or replace the batteries.

The throughput is used for the number of plaintexts processed per second (bps). Here the higher it is the better for the algorithm.

Due to the constraints, the high throughput is almost never a design goal, but an average throughput is necessary for Lightweight algorithms.

The latency is the time needed to create the output of the circuit by the time the input has been given. In a cryptographic algorithm this could correspond to the time needed to produce the ciphertext given the plaintext. This is called the latency of the encryption operation. The lower the latency the better for the algorithm, especially for real-time applications, where speed is of utmost importance.

Resources – Cost

The resources are classified in hardware and software specifications. Hardware resources depend on the area available for the gates, which is measured in gate equivalents and it actually specifies the physical area needed for the circuit. All these parameters define the memory consumption and the deployment size of the algorithm.

In the software case, the resources include, but not limited to, the ROM and RAM consumption and the code size that is needed. RAM is actually how much data is written to the memory every time the algorithm is evaluated and intermediate computations occur, while ROM is used to store the program code, so it is linked with the code size, but it also stores information like some keys used in the algorithm. Both ROM and RAM are measured in bytes and the less the bytes are the better for the algorithm. On the other hand, throughput is measured in bytes per cycle and it indicates the average amount of data that can be processed in every clock cycle. The goal here is to have a high throughput.

All these criteria compete with each other and to satisfy one, another may be lost. For example, to achieve low latency means that a higher area must be implemented. So, it is true that the slower implementations are also the smaller ones, regarding the physical area. Moreover, the higher throughput needed for the algorithm the more memory (ROM/RAM) is required so the cost is increased together with the physical area of the circuit.

Security

The security now is also related with those terms. There is a minimum-security strength required that is the length of the secret key and according to NIST it should be at least 112 bits, in order to be secure to attacks like brute-force, etc. For the algorithm to use a key of 112 bits or more, it means that more memory is needed to process the key and also the latency of the encryption operation may be increased, in comparison to the throughput that might be decreased.

Additionally, regarding the implementation and the design of cryptographic algorithms, robustness against Side-Channel Attacks (SCAs) is essential, as this kind of attack is the most common when referring to constrained devices and consequently Lightweight cryptography.

As mentioned in a previous chapter, this kind of attack uses information and knowledge about the design and the implementation of the algorithm. For instance, measuring the power consumption of the encryption or the time required for the decryption process, even the temperature of the device during an operation, could reveal significant information for the adversary to eventually break the cipher.

However, this type of attack usually indicates that the attacker has physical access to the device using the algorithm. This would not be easy to do when talking about conventional cryptography and devices like smartphones and desktops, which are protected most of the times. On the other hand, when referring to constrained and IoT devices like sensors that are placed in public areas, gaining physical access is not hard at all.

Consequently, it is vital for Lightweight algorithms to have defensive mechanisms against SCAs by hiding as much information as possible. This is accomplished by randomizing inputs of different operations so that they are statistically not depending from secret data of the algorithm. Another way to prevent an SCA would be to use a fixed time for the encryption and decryption operation regardless of the key size, so as to confuse the attacker.

To conclude, considering all these criteria and requirements for the Lightweight cryptographic algorithms it is pretty obvious that the design of them is difficult, complex and also challenging as there is a thin line between every criterion and the others.

4.3 Lightweight Cryptography and Internet of Things (IoT)

Something that makes the research field of Lightweight cryptography even more important and upcoming is another concept, which is the Internet of Things, known as IoT.

IoT is a technology that is developing day by day and it is getting into people's lives more and more as the time goes by. It refers to the infinitive devices that are connected to the Internet, in order to share and exchange data with other devices, machines and so on, without regard for the distance [36]. IoT devices in most of the cases are sensors that collect data and act in different ways depending on the data collected. The goal of IoT is to enhance and make people's life and work easier. Example of an IoT implementation is the smart homes that for example, are capable of controlling the heating and the lighting of the house automatically using sensors connected to the Internet.

Furthermore, IoT technology is already involved in the Healthcare industry, automotive applications, automated factories and even smart cities and will be even more active in the next years as well. As a result, this brings the need to secure those devices and the data they carry and transfer. These data need to be kept secret and also unchangeable, as this kind of information is sensitive and crucial, especially when referring to healthcare industries and automotive applications.

All these devices that are called IoT devices are constrained, which basically means that they are not suitable for conventional cryptography, so they use Lightweight cryptographic algorithms.

In conclusion, the IoT technology plays a huge role in the significance of Lightweight cryptography for the reasons explained above.

Despite the fact that it is difficult to achieve high level of security in constrained environments, the data involved are intensely sensitive and vital, so strong Lightweight cryptographic algorithms in terms of security are required. This is the reason NIST has initiated a Lightweight cryptography standardization process, so as to encourage the design and deployment of such algorithms. This standardization process will be presented later in this chapter.

Picture 18: Security in IoT Devices

4.4 Popular Lightweight Cryptographic Algorithms

There are plenty Lightweight cryptographic algorithms designed trying to achieve a balance along the challenges and tradeoffs analyzed previously. The balance differs depending on the application and the weights that each designer sets. Here, some of the most popular ones will be presented, so as to estimate the work that has been done in this field so far. Both Block and Stream ciphers will be cited.

Lightweight Block Ciphers

Present

Present was developed by the Orange Labs (France), Ruhr University Bochum (Germany) and the Technical University of Denmark back in 2007 [37]. It uses blocks with 64 bits of length and keys of 80 and 128 bits as well. In the first case of the 80 bits key length, as it is covered previously, it is not considered secure nowadays. However, the designers mention in their paper: “Present: An Ultra-Lightweight Block Cipher” that since the application using this algorithm will only require moderate security levels the 80 bits are adequate. This algorithm is well-known for its simplicity and its very small and compact size, as it is proven to be 2.5 times smaller than AES.

Moreover, as it is pointed out in the paper, Present fulfills the design goals set for the eSTREAM project, which was organized by EU ECRYPT network for the identification of new stream ciphers designed to be suitable for widespread adoption.

Clefia

Clefia is a proprietary Lightweight block cipher algorithm, developed by SONY in 2007 [38]. It uses blocks of 128 bits and the keys can be of 128, 192 and 256 bits of length. Despite the fact that it is a proprietary algorithm the specification of Clefia is publicly available, with the purpose of enabling the evaluation and comments by cryptographers from all over the globe. Although it is a Lightweight block cipher it maintains a high level of security, while it can be applied in both hardware and software implementations, which is not the case in many other algorithms. In hardware it can achieve the highest gate efficiency as SONY claims and when implemented in software high speed performance can be accomplished with the use of various kinds of processors.

Additionally, as SONY points out, the functions used in the algorithm are shared between the data processing part and the key scheduling part, in order to have a lower cost as the gate size is decreased.

Last but not least, both Clefia and Present are the internationally-standardized ciphers in ISO/IEC 29192:2012 and they remain until now since last reviewed in 2017 and standards are reviewed every five years.

Klein

This algorithm is not a standard like the others, but has great software performance on legacy sensor platforms and also its hardware deployment can be compact as well [39]. Klein uses a 64-bit block and the key can be 64, 80 and 96 bits of length. As the authors, Zheng Gong, Svetla Nikova and Yee Wei Law claim in their paper: Klein: A New Family of Lightweight Block Ciphers, the key length and block size are vital in the trade-offs between security and performance.

They also suggest that the 64-bit key is used for hash function and message authentication codes (MAC) implementations (Klein can be used for hashing and authentication) and that 80 along with 96-bit key is used for data encryption operations. However, even 96 bits are not considered secure nowadays, unlike 2011 when Klein was developed.

Of course, these are not the only Lightweight block ciphers developed. There are many more, but it is not in the scope of this thesis to analyze and evaluate all of them. Some of them are: LED (Lightweight Encryption Device), Midori, Mantis, HIGHT, GOS, etc.

Lightweight Stream Ciphers

Enocoro

Enocoro is a family of pseudorandom number generators, developed in 2007 by Hitachi, Ltd and was updated in 2010 to Enocoro-128v2, which was submitted to CRYPTREC (Cryptography Research and Evaluation Committees) in the same year [40].

The first version was called Enocoro v1 and was divided into Enocoro-80v1 and Enocoro-128v1 depending on the length of the key. The most recent version is v2 that uses a 128-bit key. The algorithm contains an initialization function that is applied from an initialization vector (IV) and a certain key. Consequently, Enocoro has the ability to create different keystreams using different IVs and the same key, making it a probabilistic model and not a deterministic one.

Furthermore, despite the fact that the implementation cost is low (achieves encryption process of AES with 1/10th the amount of power), it is considered a solid work and quite resilient to cryptanalytic attacks. This is also the reason it is specified as standardized and dedicated keystream generator for Lightweight stream ciphers by ISO. This standard is still valid as it was reviewed in 2018.

Trivium

Trivium was developed as a flexible trade-off between speed performance and gate area [41]. Despite that it was hardware-oriented it can be efficient in software implementations as well. It was submitted to the eSTREAM project and is now an International Standard under ISO/IEC 29192-3 together with Enocoro. Trivium is designed to generate up to 2^{64} bits of key stream from an 80-bit secret key along with an IV of the same length. This makes Trivium a probabilistic model as the Enocoro stream cipher.

In addition, the security level of the algorithm is certainly not the ideal one, since the key length is too small, however it is stated that no cryptanalytic attack is easier to apply in Trivium than any other cipher with the same parameters (80-bit key, 80-bit IV, etc).

Grain

Grain was designed and submitted to the eSTREAM competition in 2004 by Martin Hell, Thomas Johansson and Willi Meier [42]. Firstly, it was developed to use an 80-bit key and a 64-bit IV, however, the newer version called Grain-128 uses a 128-bit key which is considered secure nowadays and an IV of 96 bits. Even though the key and IV is increased, Grain still remains very small and compact and can be easily implemented in hardware. What makes Grain special is that the performance and speed can be improved at the expense of more hardware. This is not the case in many families of stream ciphers.

Last but not least, when it comes to hardware implementations Grain requires low gate area and power consumption along with small chip area. The fact that it uses a 128-bit key is essential for the level of security as there are not many other Lightweight ciphers offering this key length.

Three of the most popular Lightweight stream ciphers has been analyzed and evaluated. However, there are many more that exist and are also equally important as well.

Some of them are: Bean, Hummingbird, Snow 3G, Mickey v2, rabbit-MAC, Sablier, etc.

4.5 NIST Lightweight Crypto Standardization Process

The National Institute of Standards and Technology (NIST) has initiated a process to request, evaluate and eventually standardize Lightweight cryptographic algorithms suitable for constrained environments, where current NIST cryptographic standards cannot be implemented and are not acceptable [43] [44].

As the project overview indicates, there are numerous emerging areas where highly-constrained devices are connected to one another, exchanging data wirelessly and cooperating to perform specific tasks. Some of these emerging areas are the: sensor networks, the healthcare industry, smart home or cities, the Internet of Things concept (IoT), automotive industry, etc.

The current cryptographic algorithms are not suitable to be applied and implemented on such devices, due to the limitations, constraints and requirements that exist in constrained environment, which are analyzed and assessed previously.

Thus, the need for the design and implementation of new algorithms appropriate for Lightweight cryptography is huge and has grown over the last years as well. This is the reason of this process initiated by NIST.

At first, NIST received 57 submissions to be evaluated for standardization. Following the initial review, 56 were selected to participate in the 1st round. These algorithms, as NIST suggests, are the Round 1 Candidates.

The requirements of the submission packages were the: cover sheet, the specification and supporting documentation of the algorithm, the source code and test vectors and intellectual property statements, agreements and disclosures,

The deadline for the submission was at February 25, 2019 and the Round 1 candidates were announced at April 18, 2019. Currently, the NIST Lightweight Crypto Standardization process is in the 2nd Round and from the 56 algorithms, 32 are chosen to be the Round 2 candidates.

It is not in the scope of this thesis to analyze and evaluate all the 56 Lightweight cryptographic algorithms that have taken part in the NIST standardization process.

However, a simple reference will be given for the Round 1 candidates that were not chosen to proceed in the 2nd Round and a classification will be presented for the Round 2 candidates containing several categories.

The Round 1 candidates that did not make the 2nd Round are: Bleep64, CiliPadi, CLAE, CLX, FlexAEAD, Fountain, GAGE and InGAGE, HERN & HERON, LAEM, Lilliput-AE, Limdolen, Qameleon, Quartet, REMUS, Shamash & Shamashash, SIMPLE, SIV-Rijndael256, SIV-TEM-PHOTON, SNEIK, Sycon, Thank Goodness It’s Friday (TGIF), Triad, TRIFLE and Yanará and Coral.

Below a classification is presented for the Round 2 candidates. The categories chosen are:

1. Whether the algorithm is a block or a stream cipher

2. If the algorithm possesses hashing capabilities

3. The Universities or Industries submitters work for

Candidate	Block Cipher	Stream Cipher	Hash Function	Universities or Industries
ACE	√		√	Department of Electrical and Computer Engineering, University of Waterloo
ASCON	√		√	Graz University of Technology
COMET	√			University of Haifa, Israel Amazon Web Services Inc., Seattle, USA Indian Statistical Institute Kolkata, India
DryGASCON	√		√	No institute presented
Elephant	√		√	KU Leuven and imec- Computer Security and Industrial Cryptography (COSIC), Belgium Radboud University, The Netherlands
ESTATE	√		√	NTT Secure Platform Laboratories, Japan Indian Statistical Institute, Kolkata, India Computer Science Department, CINVESTAV-IPN, Mexico
ForkAE	√			Université de Lorraine, CNRS, Vandœuvre-les-Nancy, France University of Bristol
GIFT-COFB	√			National Science Foundation
Gimli	√		√	Department of Computer Science, University of Illinois at Chicago

Grain-128AEAD		√		Lund University, Sweden FHNW University of Applied Sciences and Arts, Switzerland
HYENA	√			Avik Chakraborti - NTT Secure Platform Laboratories, Japan Nilanjan Datta - Indian Statistical Institute, Kolkata, India
ISAP	√			Netherlands Organization for Scientific Research (NWO)
KNOT	√		√	No institute presented
LOTUS-AEAD and LOCUS-AEAD	√			NTT Secure Platform Laboratories, Japan Indian Statistical Institute, Kolkata, India Computer Science Department, CINVESTAV-IPN, Mexico
mixFeed	√			Indian Statistical Institute,Kolkata
ORANGE	√		√	Indian Statistical Institute,Kolkata
Oribatida	√			Indian Statistical Institute Kolkata Computer Science Department, CINVESTAV-IPN, Mexico
PHOTON-Beetle	√		√	Nanyang Technological University, Singapore NTT Secure Platform Laboratories, Japan Indian Statistical Institute, Kolkata, India
Pyjamask	√			School of Physical and Mathematical Sciences Nanyang Technological University, Singapore NTT Secure Platform Laboratories, Japan
Romulus	√			Nagoya University, Japan Nanyang Technological University, Singapore NEC Corporation, Japan

SAEAES	√			Mitsubishi Electric Corporation, Japan The University of Electro-Communications, Japan
Saturnin	√		√	UCL Crypto Group, Belgium NCC Group, Canada
SKINNY-AEAD/SKINNY-HASH	√		√	SnT, University of Luxembourg, Luxembourg ANSSI, Paris, France Cybercrypt A/S, Denmark Horst Görtz Institute for IT Security, Ruhr-University at Bochum, Germany School of Physical and Mathematical Sciences Nanyang Technological University, Singapore NTT Secure Platform Laboratories, Japan Rambus Cryptography, The Netherlands
SPARKLE(SCHWAEMM and ESCH)	√		√	SnT and CSC, University of Luxembourg, Luxembourg Inria, Paris, France University of Edinburgh, U.K.
SPIX	√			Department of Electrical and Computer Engineering University of Waterloo
SpoC	√			Department of Electrical and Computer Engineering University of Waterloo
SPook	√			ICTEAM Institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium Institute for IT Security, Ruhr-University at Bochum, Germany Team SECRET, Inria Paris Research Center, France
Subterranean 2.0		√	√	Radboud University, Digital Security Department, Nijmegen

SUNDAE-GIFT	√			Nanyang Technological University, Singapore Temasek Laboratories@NTU, Singapore LASEC, ÉcolePolytechnique Fédéralede Lausanne, Switzerland Technical University of Denmark, Denmark Cybercrypt A/S, Denmark NTT Secure Platform Laboratories, Japan
TinyJambu	√			Division of Mathematical Sciences Nanyang Technological University
WAGE		√		Department of Electrical and Computer Engineering University of Waterloo
Xoodyak	√		√	STMicroelectronics Radboud University

All of these Lightweight cryptographic algorithms, including the ones that did not made it to the 2nd Round of the NIST standardization process, provide authenticated encryption. Consequently, it was unnecessary to categorize the algorithms based on the authentication scheme.

5. SECURITY OF LIGHTWEIGHT CRYPTOGRAPHIC ALGORITHMS

In this chapter, several Lightweight cryptographic algorithms will be evaluated regarding their security. More specifically, some algorithms that took part in the National Institution of Standards and Technology (NIST) standardization process for Lightweight cryptography will be analyzed, so as to find Boolean functions used in the ciphers and estimate their level of security by their properties. The Boolean function's properties, which are analyzed in a previous chapter, define whether the cipher is immune to specific types of attacks or not. Consequently, they serve a vital role in the level of security the algorithm can offer.
After examining all the candidate algorithms of the NIST competition, Boolean functions were discovered in 3 of them that are:

- Grain-128 AEAD
- TinyJambu
- SKINNY-AEAD/SKINNY-HASH

At first, the Grain-128AEAD algorithm, which is currently in Round 2 of the NIST standardization process, will be evaluated regarding the Boolean function it uses. Some properties of its Boolean function will be estimated and assessed regarding the security strength they can provide.

Secondly, TinyJambu, which is also currently in the 2nd Round of the competition uses a non-linear feedback function that is actually a Boolean function, so it will be evaluated with the same criteria.

Lastly, SKINNY-AEAD/SKINNY-HASH (also in the 2nd round of the standardization process) uses a S-box of 8 inputs and 8 outputs. This S-box can be analyzed as 8 district Boolean functions. These functions will also be evaluated regarding their properties.

5.1 SageMath Mathematical Software

The Boolean functions of the algorithms chosen are going to be evaluated using SageMath.

SageMath is a free open-source mathematics software system, which is designed over many existing open-source packages like: NumPy, matplotlib, GA, R, SciPy and more [45]. All of these packages can be combined into one and used in a Python-based language specialized in mathematical operations and functions.

SAGE stands for System for Algebra and Geometry Experimentation and it can cover various aspects of mathematics such as algebra, number theory, statistics, numerical analysis etc.

This powerful mathematical software was created by William Stein and its initial goal was to become an open source alternative of software similar to Magma, Mathematica and Matlab.

Since cryptography is extremely based on mathematical operations, Sage can be an essential tool in analyzing, simulating and estimating specific functions used in cryptographic algorithms.

The real reason this software is used in this thesis is because there is a module used in Sage especially for the Boolean functions. Using this module, the functions chosen from the algorithms mentioned above, will be evaluated based on the results produced by Sage. More specifically the cryptographic properties of every Boolean function will be calculated. All the process will be described and presented in great detail.

Picture 19: SageMath

5.2 Evaluation of Grain-128 AEAD Boolean function

Grain-128 AEAD is an authenticated encryption algorithm that is based on the Grain family of algorithms that are mentioned in a previous chapter. In particular, it is based on Grain-128a, which was introduced in 2011 [46].

According to the designers this algorithm that currently participates in the NIST standardization process only has minor changes made to Grain128a, so the security level it can offer is based on the results taken from Grain-128a. The benefit here is that the previous algorithm, Grain-128a has been extensively analyzed regarding the security.

Consequently, the results that are going to be presented in this thesis can be easily verified as well.

Grain-128 AEAD algorithm is actually divided in 2 blocks. The 1st one is a pre-output generator that creates a stream of pseudo-random bits used for encryption and authentication and consists of a Liner Feedback Shift Register (LFSR) and a Non-linear Feedback Shift Register and a pre-output function.

The 2nd block is an authenticator generator that is composed by a shift register and an accumulator, which is actually a temporary register.

What is important here is that in the 1st block, a Boolean function is being used with 9 input bits. 2 of them are taken from the NFSR and the rest 7 from the LFSR. The Boolean function is defined as:

$$h(x) = x_0 * x_1 \oplus x_2 * x_3 \oplus x_4 * x_5 \oplus x_6 * x_7 \oplus x_0 * x_4 * x_8$$

Where the variables x_0 up to x_8 are the state variables of the linear and non-linear shift registers.

This Boolean function is also used in the calculation of the pre-output function.

So basically, the first evaluation is going to happen in this Boolean function used by the Grain-128 AEAD algorithm.

As mentioned above, the Software SageMath is going to be used and with this mathematical tool, several properties of the Boolean function will be calculated.
Since Sage is a python-based program, the module needed will be imported first, so as to use the functions already implemented to compute the desirable properties. There is a module called crypto, which is specialized for cryptography function and inside crypto there are functions related explicitly for Boolean functions [47].
So, the following command will import the module and function needed:

from sage.crypto.boolean_function import BooleanFunction

Then the variables that will be used need to be defined. Generally a Boolean function in Sage, can be constructed in many ways, like a string that represents the truth table in hexadecimal, or a binary list that is the truth table of the result (has to be of length that is a power of 2), a Boolean Polynomial, which the result of it is the corresponding Boolean function, etc.
The last method is chosen, so the command will look like this.

B.<x0,x1,x2,x3,x4,x5,x6,x7,x8> = BooleanPolynomialRing()

B is the name of the Boolean Polynomial. This is mandatory, so as to define the variables x_0 up to x_8 that are going to be used.
Now the Boolean function destined for the evaluation has to be stated. Based on the h(x) function described above the command shown below is entered:

*h = BooleanFunction(x0*x1 + x2*x3 + x4*x5 + x6*x7 + x0*x4*x8)*

In cryptography, generally the symbol "+" implies the XOR operation; this is why SageMath does not use the ⊕ symbol.
Now that the Boolean function has been defined, the properties of it can be computed.

There are build-in functions in SageMath and especially in the module imported that can be used to compute the properties directly.
At first, the degree of the function will be computed, which is quite obvious. This command is run just to see the way function can be executed given a Boolean function

h.algebraic_degree()

The output here is 3, which can be verified from the product: x0*x4*x8 in the function.
Next, the balancedeness of the function is going to be measured. This value can be either true or false. It is true if the output has a 0.5 probability to be 1 and the same one to be 0. If this does not apply the output is false.
By running the following command:

h.is_balanced()

The output is False. So, this specific Boolean function does not have a balanced truth table as this property implies.
However, from the truth table that can be extracted through SageMath it is easy to see that from the total of 2^9 = 512outputs, 272 of them are 0 and the rest 240 are 1. So, the probability of getting a 0 is 0.53125 and the probability of getting a 1 is 0.46875, which is quite balanced but not completely balanced to satisfy the property.

Now the correlation immunity property will be computed. A Boolean function is said to be correlation immune of order k, when the output of the function is statistically unrelated of any combination of k inputs of the function.

The command is shown beneath:

```
h.correlation_immunity()
```

The output here is 0. This means that this function is not immune to correlation attacks. Nevertheless, it has to be clear that since this function is not directly related to the creation of the keystream, this vulnerability does not directly result in a conclusion that can be easily used to compromise the function.

The next cryptographic property is the non-linearity. In general, this property shows how “close” the Boolean function analyzed is to a linear function, which basically is the Hamming distance of the 2 functions. More specifically, it represents the least number of the outputs in the truth table that if modified in a certain way the Boolean function becomes a linear one.
The command is represented below:

```
h.nonlinearity()
```

The output here is 240. This means that if 240 of the 512 outputs in the truth table are modified the function becomes linear. For a 9-variable function the nonlinearity according to the relation that resembles the case of bent functions, as discussed earlier, is $2^{(9-1)} - 2^{(9-1)/2-1}$, which equals to 248 and it is very close to 240 as computed. So, this number is pretty high, and this Boolean function does provide a great security level against linear attacks.

Last but not least, the algebraic immunity cryptographic property is measured as well. This specific property, as the name suggests the degree of immunity this function can give against algebraic attacks, which basically are cryptanalytic attacks that try to express the function as a mathematical system and attempt to solve it, so as to discover the secret key. This is measured by the minimum degree of a function that if multiplied with the function examined the result is 0.

h.algebraic_immunity()

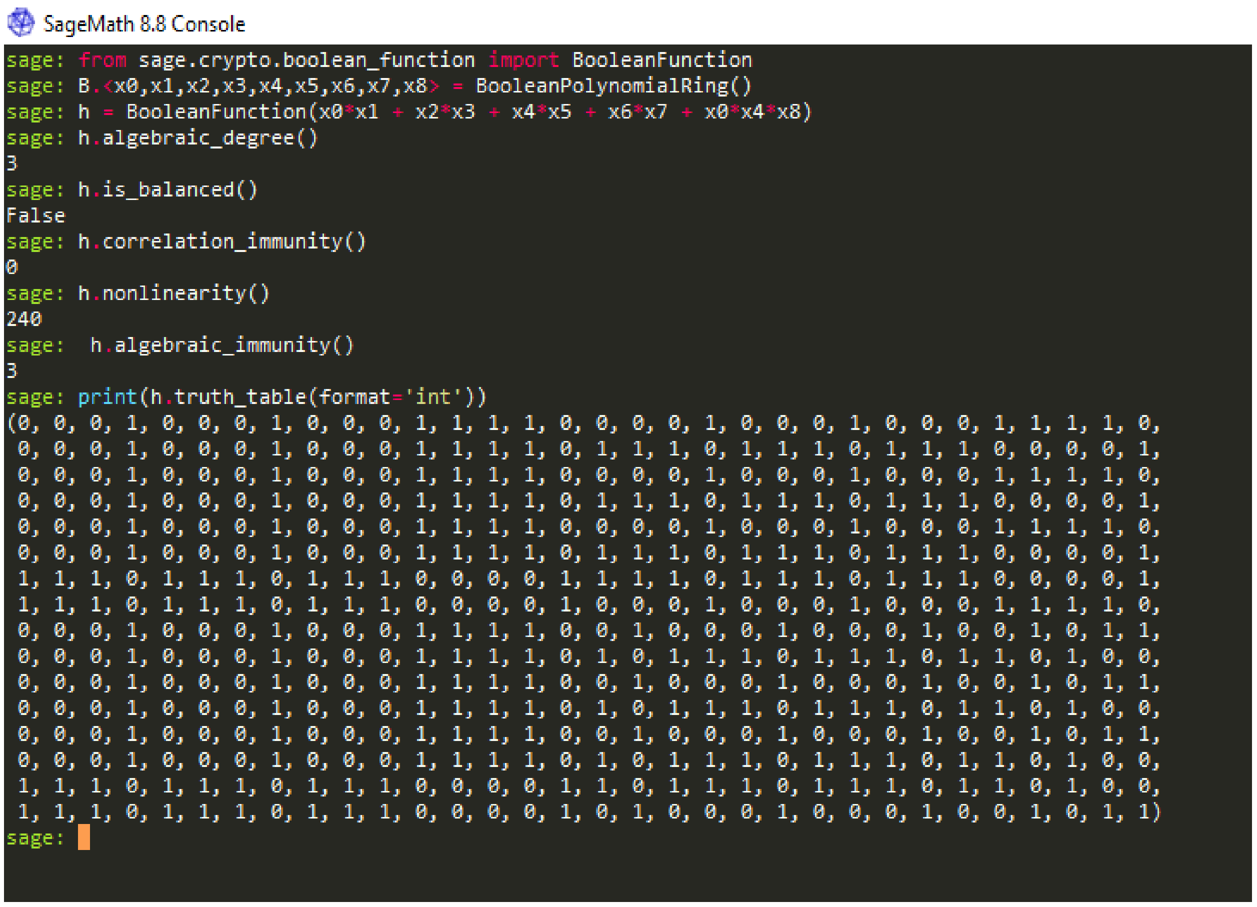
The output of this command is 3. Consequently, this means that there is a function, for instance $\mathrm{g(x)}$ of degree 3, such that $g(x) * h(x) = 0$, where $\mathrm{h(x)}$ is the Boolean function evaluated. Since the number of variables in this function is 9, the maximum possible algebraic immunity it can attain is n/2 = 9/2 = 4.5 and rounded up is equal to 5. So, the value of 3 is considered low and the function could be vulnerable to algebraic attacks.

Furthermore, there is also the capability to produce the truth table of a Boolean function as well. This can be done by executing the simple command following:

h.truth_table(format='int')

The format is changed to integer, so as to display the outputs in 1s and 0s. Alternatively, the output would be presented in true or false format.

Next, a screenshot is shown of the code created to compute all these cryptographic properties described above.



```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: B.<x0,x1,x2,x3,x4,x5,x6,x7,x8> = BooleanPolynomialRing()
sage: h = BooleanFunction(x0*x1 + x2*x3 + x4*x5 + x6*x7 + x0*x4*x8)
sage: h.algebraic_degree()
3
sage: h.is_balanced()
False
sage: h.correlation_immunity()
0
sage: h.nonlinearity()
240
sage:  h.algebraic_immunity()
3
sage: print(h.truth_table(format='int'))
(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1)
sage: 
```

Picture 20: Properties of Grain-128 AEAD Boolean function

In addition, the truth table is represented in the integer format as mentioned above. There are $2^9 = 512$ outputs as 9 variables x_0 up to x_9 are used.

5.3 Evaluation of TinyJambu's Boolean function

TinyJambu is a Lightweight cryptographic algorithm that came from Jambu, which participated in the CAESAR competition and was the smallest authenticated encryption mode in this competition [48]. It was also selected for the 3rd round of CAESAR and was exhibited at the NIST Lightweight cryptography workshop in 2015.

TinyJambu, which is a small variant of its processor, uses the half message block size and the state size is just 2/3 of Jambu's. The state size is 128 bits and the block size is 32 bits. When the 96-bit nonce of the algorithm is reused, TinyJambu can offer more enhanced authentication security than Jambu despite its size.

This algorithm can support 3 key sizes of 128, 192 and 256 bits.

Furthermore, a 128-bit keyed permutation of n rounds is used inside the algorithm. During this permutation and in the *ith* round, a 128-bit nonlinear feedback shift register is utilized, in order to update the state, as shown below:

The feedback is equal to the Boolean function and then a XOR operation is done with the mod operation of the key in that round and the key length.

To make it more comprehensible the function is given beneath:

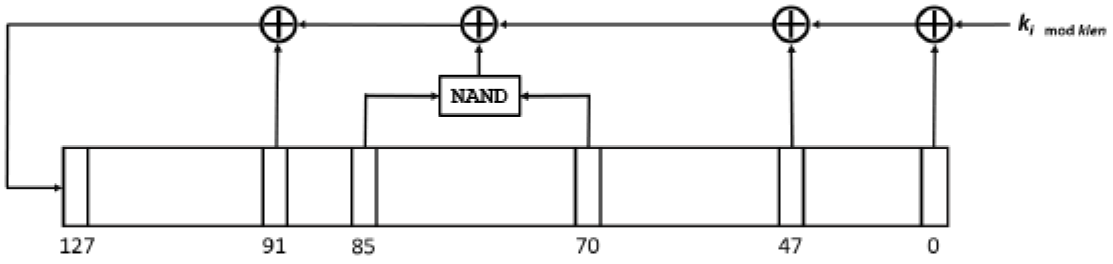
$$\text{feedback} = x_0 + x_{47} + (\sim(x_{70} * x_{85})) + x_{91} + (k_i \text{mod}\, k_{len})$$

k_i: the *ith* bit of the key

k_{len}: the key length in bits

x_i: the *ith* bit of the state of the permutation

A figure is also presented from the TinyJambu specification document.



Picture 21: The 128-bit Nonlinear Feedback Shift Register in TinyJambu

As a result, the Boolean function that is going to be evaluated is:

$f(x) = x_0 + x_{47} + (\sim(x_{70} * x_{85})) + x_{91}$

However, the NAND operation needs to be replaced by a XOR one, so it is transformed as shown underneath:

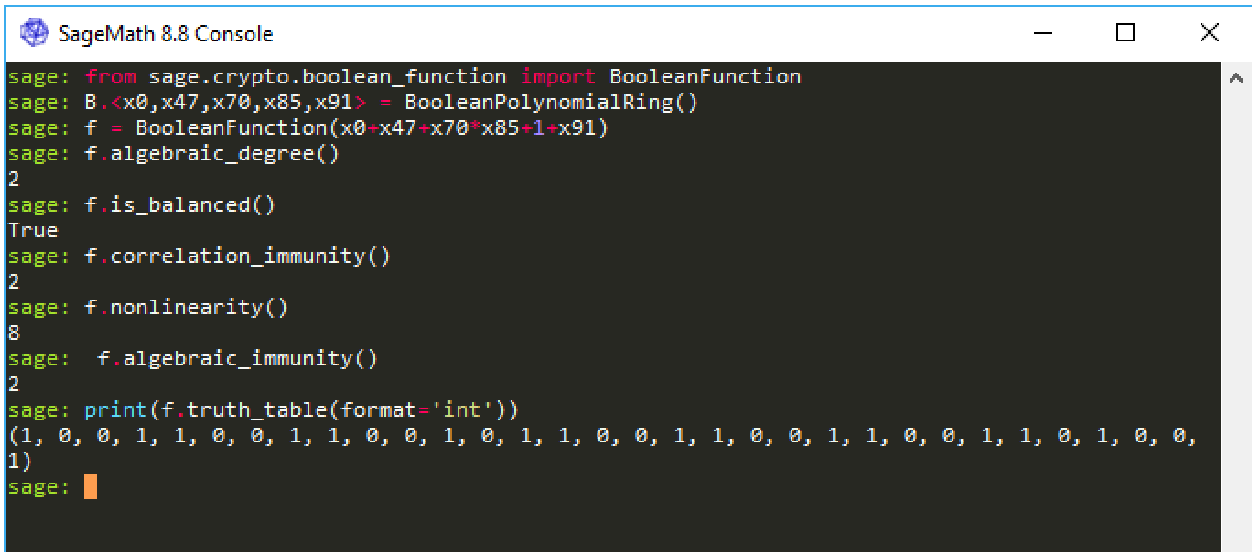
$\sim(x_{70}*x_{85}) = x_{70}*x_{85} + 1$

So, the final form of the Boolean function will be:

$f(x) = x_0 + x_{47} + (x_{70} * x_{85}+1) + x_{91}$

Now the same commands will be executed as the Boolean function of the Grain-128 AEAD cryptographic algorithm. Thus, there is no need to describe in great detail the process.

First, a screenshot will be presented with the code executed for this Boolean function and then the results regarding the cryptographic properties of the Boolean function will be evaluated and commented.



```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: B.<x0,x47,x70,x85,x91> = BooleanPolynomialRing()
sage: f = BooleanFunction(x0+x47+x70*x85+1+x91)
sage: f.algebraic_degree()
2
sage: f.is_balanced()
True
sage: f.correlation_immunity()
2
sage: f.nonlinearity()
8
sage:  f.algebraic_immunity()
2
sage: print(f.truth_table(format='int'))
(1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0,
1)
sage: 
```

Picture 22: Properties of TinyJambu’s Boolean function

To begin with, it is obvious that the algebraic degree of this function is 2 since there is only one product and it is between 2 variables.

The function is also completely balanced since from the $2^5 = 32$ outputs half of them are 0s and the other half are 1s. So, despite the fact that this function has only 5 variables and a degree of 2, it is not vulnerable to correlation attack due to its balancedness.

In addition, the correlation immunity of this function is of order 2 and in collaboration with the balancedness of the function it makes it immune to correlation related attacks.

The nonlinearity here is just 8. For a 5-variable function the maximum possible nonlinearity according to the Bent function is approximately $2^{5-1} - 2^{(5-1)/2-1}$, which is equal to 14. Consequently, this is not a great property for the function, as it basically implies that if 8 outputs are changed in the truth table then the function becomes a linear one and it is actually vulnerable to linear attacks.

The algebraic immunity is equal to 2. As explained in the previous subchapter this means that there is a function of degree 2 that if multiplied with f the result will be 0. Since the maximum possible value the algebraic immunity can attain is n/2 = 5/2 = 2.5, which equals to 3 when rounded up, the output is not equal but is close to this value.

Nevertheless, it must be taken into consideration that TinyJambu’s security level is not exclusively based on this cryptographic Boolean function. This Boolean function is just a part of the permutation that is used and in particular its purpose is to update the state.

There are other techniques in the algorithm that are used as safety nets, apart from the Boolean function.

For instance, the state of the permutation is updated 1024 times during the key setup, which means that the procedure analyzed above is performed 1024 times and not just 1. Also, for the nonce setup the state is updated 384 times.

Nevertheless, the purpose of this thesis is to evaluate only Boolean functions, so no more detail is going to be given in the other security techniques or procedures utilized in the cryptographic Lightweight algorithms.

5.4 Evaluation of Skinny-AEAD

Skinny is a family Lightweight block ciphers, which was proposed at Crypto in 2016, the 36th International Cryptology Conference, held at the University of California, Santa Barbara (UCSB) [49].

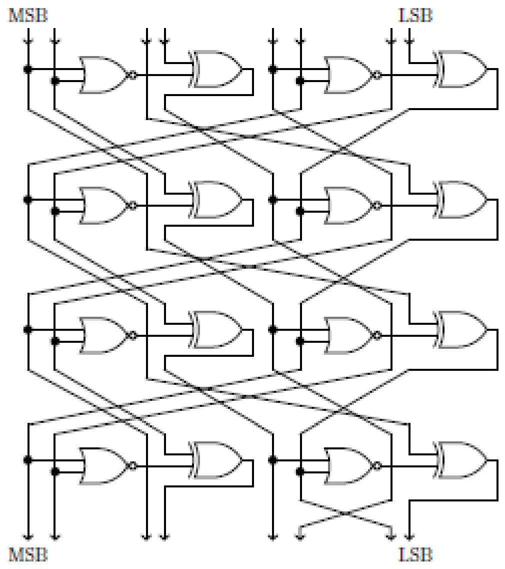
The primary design uses a 128-bit key, along with associated data and message of up to 2^{64} * 16 bytes. After the encryption, a ciphertext is produced of the same length as the plaintext with a 128-bit tag. Moreover, there is also another design more suitable to applications where the support for such great amount of input bytes is not necessary. This design uses also a 128-bit key, a 96-bit nonce and an associated data with the message of up to 2^{28} bytes both. The ciphertext is produced with a t_l-bit tag, where t_l is between 64 and 128. It is obvious that this design provides a smaller and faster scheme than the first one, but it does not support input messages of up to 2^{50} bytes, which is a submission requirement.

Despite the fact that this Lightweight cryptographic algorithm does not include a Boolean function in its implementation, there is a way to evaluate the Substitution Box that is utilized, as 8 separate Boolean functions.

In particular, an encryption round in Skinny consists of 5 operations that are performed in the following order: SubCells, AddConstants, AddRoundTweakey, ShiftRows and MixColumns. It is not in the scope of this thesis to analyze every transformation.

Nevertheless, in the 1st operation, the Subcells, an 8-bit Sbox S_8 is implemented to each cell of the ciphers internal state. Briefly, a Substitution Box is a vital component for Symmetric cryptography, as its name implies it performs substitution. Basically, it takes a number of bits as input and transforms them into a number of bits as the output. The input number could be equal to the output or not, it depends on the Sbox. This particular substitution box used is an 8-bit one. This means that it takes an 8-bit input and produces an output of the same length.

Below, a construction of the Sbox is shown.



Picture 23: Construction of the Skinny's Sbox

The design of this substitution box is quite uncomplicated and is inspired by the Piccolo Sbox. Piccolo is a 64-bit block cipher that also supports 80 and 128 bits and it contains a 4-bit Sbox, which is similar to the one introduced here.

In addition, there is a more detailed analysis of the design of this Sbox provided by the authors.

More specifically, the 8 inputs of the Sbox are represented as $x_{7,}x_{6,}x_{5,}x_{4,}x_{3,}x_{2,}x_{1,}x_{0,}$ where x_0 is the least significant bit and x_7 is most significant one. Firstly, a transformation is implemented as shown below.

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \rightarrow (x_7, x_6, x_5, x_4 \oplus (\overline{x_7 \vee x_6}), x_3, x_2, x_1, x_0 \oplus (\overline{x_3 \vee x_2}))$$

The $\oplus$ symbol indicates the XOR operation, the V the OR operation and the dash above the OR operation indicates a NOT. After that a bit permutation is applied as follows:

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \longrightarrow (x_2, x_1, x_7, x_6, x_4, x_0, x_3, x_5)$$

This process is performed 4 times, but in the last iteration, instead of the permutation there is just a bit swap between x_1 and $x_{0.}$

Since this Sbox, takes an 8-bit input and produces an 8-bit output, it could be interpreted as 8 discrete Boolean functions. So, in this dissertation these 8 Boolean functions are going to be evaluated. In order to make this clearer, since each input produce an 8-bit output, only one bit of the output is going to be used at a time.

Consequently, the 1st Boolean function's truth table will consist of the 1st bit of every Sbox output, making a truth table of 256-bit length. The 2nd Boolean function's truth table will consist of the 2nd bit of every Sbox output and so forth.

Next, since the truth table of every Boolean function is calculated, it will be imported into the SageMath tool and evaluated the same way with the previous Boolean functions of the Grain and TinyJambu Algorithms.

The Sbox is going to be simulated using Python and the PyCharm IDE (Integrated Development Environment). Moreover, the authors provide an Appendix with the table of the Sbox, so the results are going to be verified as well.

The code of the program created is presented below. Comments are added to explain the procedure followed.

```
# XOR operation between 2 variables
def xor(x, y):
return int((x and not y) or (not x and y))


# NOR(NOT OR) operation between 2 variables
def nor(x, y):
return int(not(x or y))


# Declaration of Sbox Output (binary and hexadecimal) and Boolean functions
sbox_output_hex = [None] * 256
sbox_output_bin = [None] * 256
boolean_function_1 = [None] * 256
boolean_function_2 = [None] * 256
boolean_function_3 = [None] * 256
boolean_function_4 = [None] * 256
boolean_function_5 = [None] * 256
boolean_function_6 = [None] * 256
boolean_function_7 = [None] * 256
boolean_function_8 = [None] * 256




# Sbox Function
def skinny_aead_sbox(input_bits):  # input bits are in 8-bit string format
x = list(input_bits)  # making the input bits a list
```

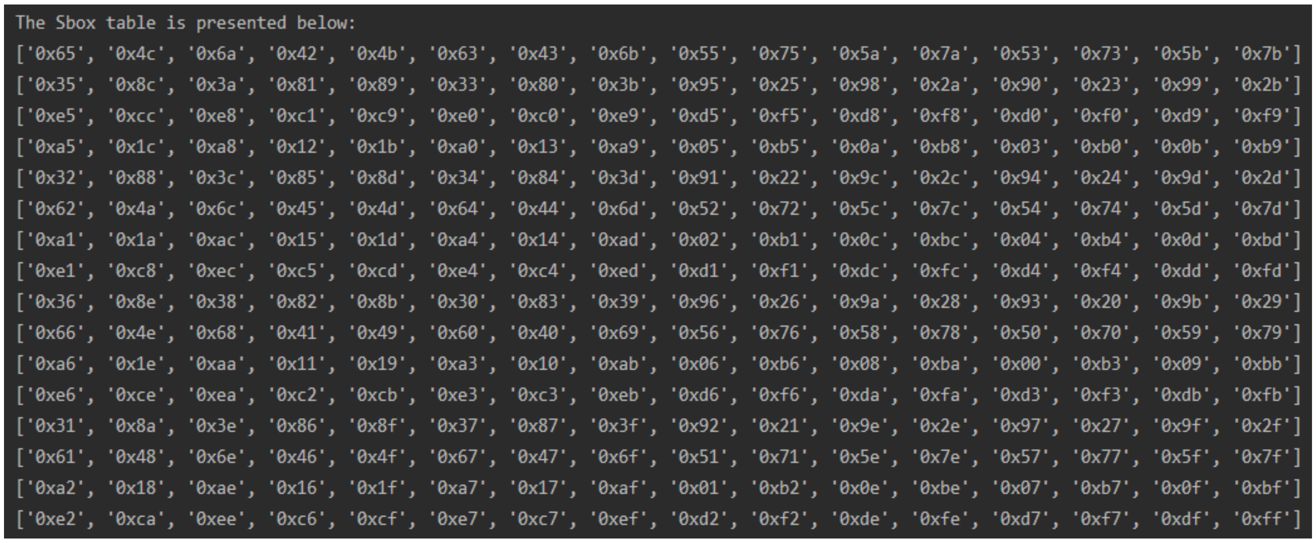
```
x = [int(x) for x in x]  # changing the sting bits into integers

''' Since in Python the 1st element in a list is in the 0 position,
    the mapping between the bits specified here and in the Skinny's
    documentation will be: x[0] -> x7, x[1] -> x6 and so forth '''

counter = 0
while counter <4:  # There are 4 iterations in the Sbox design
if counter != 3:  # The first 3 iterations
x[3] = xor(x[3], nor(x[0], x[1]))  # This equals to (x4 XOR (x7 NOR x6)) in the
documentation
x[7] = xor(x[7], nor(x[4], x[5]))  # Similarly to (x0 XOR (x3 NOR x2))
x = [x[5], x[6], x[0], x[1], x[3], x[7], x[4], x[2]]
# Above line equals to (x7, x6, x5, x4, x3, x2, x1, x0) --> (x2, x1, x7, x6, x4, x0,
x3, x5)


else:
''' This part is for the last iteration , where there is just a swap between x1 and
x2, after the XOR and NOR operations '''

x[3] = xor(x[3], nor(x[0], x[1]))
            x[7] = xor(x[7], nor(x[4], x[5]))
            x[5], x[6] = x[6], x[5]  # Swap between x1 and x2
counter += 1
x = [str(x) for x in x]  # Converting integer bits into strings bits
x = ("".join(x))  # Converting the list into 1 string of bits
x = "0x{:02x}".format(int(x, 2))  # Converting the binary into a hex number
return x





for w in range(0, 256):
    sbox_output_hex[w] = (format(w, '08b'))  # Converting the integers of the loop
into an 8-digit binary
sbox_output_hex[w] = skinny_aead_sbox(sbox_output_hex[w])  # Passing the binary into
```

```
the Skinny Sbox function
sbox_output_bin[w] = format(int(sbox_output_hex[w], 16), '08b') # Converting  hex
output of function into binary
for i in range(0, 256, 16):
print(sbox_output_hex[i:i+16])
# Printing 16 outputs per line to compare it with the table int the table of the
documentation

# Below is the output of the discrete Boolean functions created from the Sbox
for i in range(0, 256):
    boolean_function_1[i] = sbox_output_bin[i][0]
    boolean_function_2[i] = sbox_output_bin[i][1]
    boolean_function_3[i] = sbox_output_bin[i][2]
    boolean_function_4[i] = sbox_output_bin[i][3]
    boolean_function_5[i] = sbox_output_bin[i][4]
    boolean_function_6[i] = sbox_output_bin[i][5]
    boolean_function_7[i] = sbox_output_bin[i][6]
    boolean_function_8[i] = sbox_output_bin[i][7]

# The Truth Table of each Boolean Function is extracted in hexadecimal format
boolean_function_1 = hex(int("".join(boolean_function_1),2))
boolean_function_2 = hex(int("".join(boolean_function_2),2))
boolean_function_3 = hex(int("".join(boolean_function_3),2))
boolean_function_4 = hex(int("".join(boolean_function_4),2))
boolean_function_5 = hex(int("".join(boolean_function_5),2))
boolean_function_6 = hex(int("".join(boolean_function_6),2))
boolean_function_7 = hex(int("".join(boolean_function_7),2))
boolean_function_8 = hex(int("".join(boolean_function_8),2)




print('The 8 Boolean functions truth tables are presented below:')
print('1st Boolean function: ', boolean_function_1)
print('2nd Boolean function: ', boolean_function_2)
print('3rd Boolean function: ', boolean_function_3)
print('4th Boolean function: ', boolean_function_4)
print('5th Boolean function: ', boolean_function_5)
print('6th Boolean function: ', boolean_function_6)
print('7th Boolean function: ', boolean_function_7)
print('8th Boolean function: ', boolean_function_8)
```

As explained above, the purpose of this program is to simulate the Sbox used in the Skinny-AEAD Lightweight cryptographic Algorithm and extract the 8 Boolean functions described above.

So, the program is divided into these 2 parts. In order to proceed to the 2nd part the 1st one needs to be successful and have a correct result of the Sbox table. Since the authors provide this table in a hexadecimal form, a comparison is going to be made to verify that the program has the desirable result.

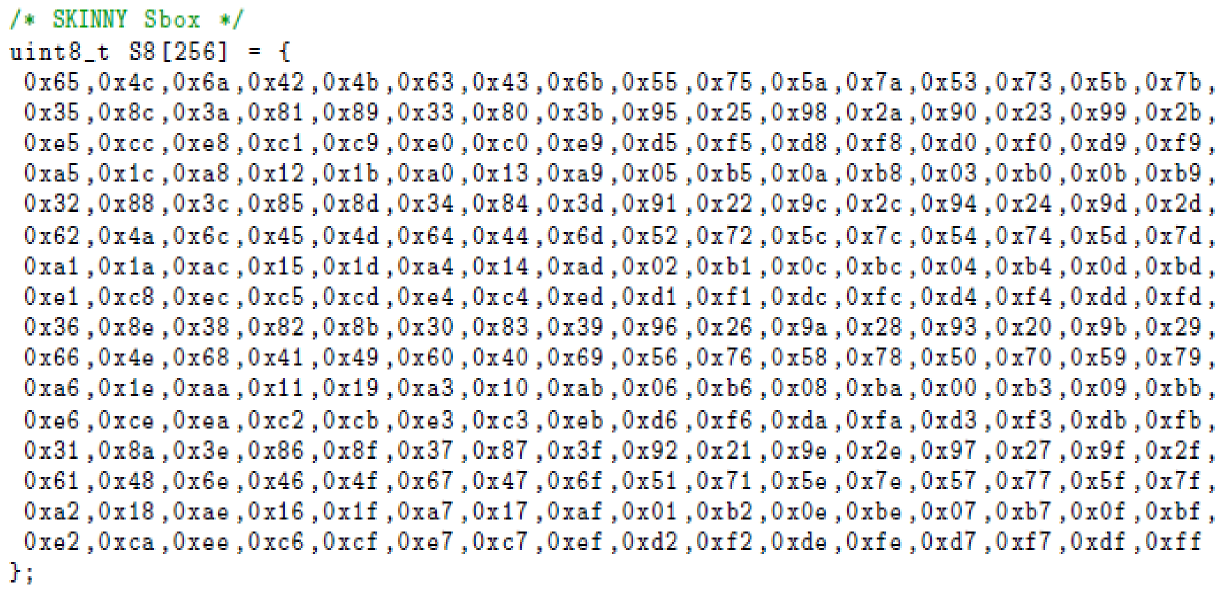
Below the output table of the Sbox is presented from the simulation created using Python.



```
The Sbox table is presented below:
['0x65', '0x4c', '0x6a', '0x42', '0x4b', '0x63', '0x43', '0x6b', '0x55', '0x75', '0x5a', '0x7a', '0x53', '0x73', '0x5b', '0x7b']
['0x35', '0x8c', '0x3a', '0x81', '0x89', '0x33', '0x80', '0x3b', '0x95', '0x25', '0x98', '0x2a', '0x90', '0x23', '0x99', '0x2b']
['0xe5', '0xcc', '0xe8', '0xc1', '0xc9', '0xe0', '0xc0', '0xe9', '0xd5', '0xf5', '0xd8', '0xf8', '0xd0', '0xf0', '0xd9', '0xf9']
['0xa5', '0x1c', '0xa8', '0x12', '0x1b', '0xa0', '0x13', '0xa9', '0x05', '0xb5', '0x0a', '0xb8', '0x03', '0xb0', '0x0b', '0xb9']
['0x32', '0x88', '0x3c', '0x85', '0x8d', '0x34', '0x84', '0x3d', '0x91', '0x22', '0x9c', '0x2c', '0x94', '0x24', '0x9d', '0x2d']
['0x62', '0x4a', '0x6c', '0x45', '0x4d', '0x64', '0x44', '0x6d', '0x52', '0x72', '0x5c', '0x7c', '0x54', '0x74', '0x5d', '0x7d']
['0xa1', '0x1a', '0xac', '0x15', '0x1d', '0xa4', '0x14', '0xad', '0x02', '0xb1', '0x0c', '0xbc', '0x04', '0xb4', '0x0d', '0xbd']
['0xe1', '0xc8', '0xec', '0xc5', '0xcd', '0xe4', '0xc4', '0xed', '0xd1', '0xf1', '0xdc', '0xfc', '0xd4', '0xf4', '0xdd', '0xfd']
['0x36', '0x8e', '0x38', '0x82', '0x8b', '0x30', '0x83', '0x39', '0x96', '0x26', '0x9a', '0x28', '0x93', '0x20', '0x9b', '0x29']
['0x66', '0x4e', '0x68', '0x41', '0x49', '0x60', '0x40', '0x69', '0x56', '0x76', '0x58', '0x78', '0x50', '0x70', '0x59', '0x79']
['0xa6', '0x1e', '0xaa', '0x11', '0x19', '0xa3', '0x10', '0xab', '0x06', '0xb6', '0x08', '0xba', '0x00', '0xb3', '0x09', '0xbb']
['0xe6', '0xce', '0xea', '0xc2', '0xcb', '0xe3', '0xc3', '0xeb', '0xd6', '0xf6', '0xda', '0xfa', '0xd3', '0xf3', '0xdb', '0xfb']
['0x31', '0x8a', '0x3e', '0x86', '0x8f', '0x37', '0x87', '0x3f', '0x92', '0x21', '0x9e', '0x2e', '0x97', '0x27', '0x9f', '0x2f']
['0x61', '0x48', '0x6e', '0x46', '0x4f', '0x67', '0x47', '0x6f', '0x51', '0x71', '0x5e', '0x7e', '0x57', '0x77', '0x5f', '0x7f']
['0xa2', '0x18', '0xae', '0x16', '0x1f', '0xa7', '0x17', '0xaf', '0x01', '0xb2', '0x0e', '0xbe', '0x07', '0xb7', '0x0f', '0xbf']
['0xe2', '0xca', '0xee', '0xc6', '0xcf', '0xe7', '0xc7', '0xef', '0xd2', '0xf2', '0xde', '0xfe', '0xd7', '0xf7', '0xdf', '0xff']
```

Picture 24: Sbox table from Python program

Now a screenshot is shown from the table as presented in the Skinny-AEAD documentation.

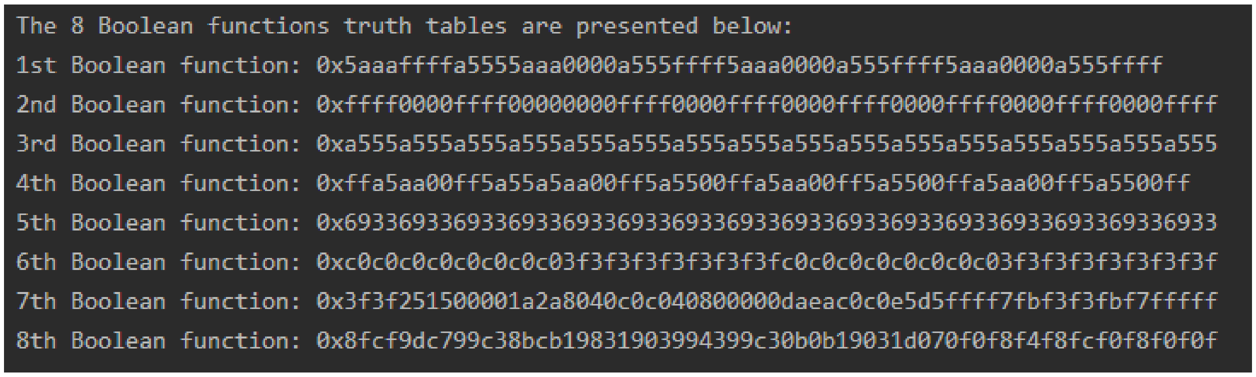


```
/* SKINNY Sbox */
uint8_t S8[256] = {
 0x65,0x4c,0x6a,0x42,0x4b,0x63,0x43,0x6b,0x55,0x75,0x5a,0x7a,0x53,0x73,0x5b,0x7b,
 0x35,0x8c,0x3a,0x81,0x89,0x33,0x80,0x3b,0x95,0x25,0x98,0x2a,0x90,0x23,0x99,0x2b,
 0xe5,0xcc,0xe8,0xc1,0xc9,0xe0,0xc0,0xe9,0xd5,0xf5,0xd8,0xf8,0xd0,0xf0,0xd9,0xf9,
 0xa5,0x1c,0xa8,0x12,0x1b,0xa0,0x13,0xa9,0x05,0xb5,0x0a,0xb8,0x03,0xb0,0x0b,0xb9,
 0x32,0x88,0x3c,0x85,0x8d,0x34,0x84,0x3d,0x91,0x22,0x9c,0x2c,0x94,0x24,0x9d,0x2d,
 0x62,0x4a,0x6c,0x45,0x4d,0x64,0x44,0x6d,0x52,0x72,0x5c,0x7c,0x54,0x74,0x5d,0x7d,
 0xa1,0x1a,0xac,0x15,0x1d,0xa4,0x14,0xad,0x02,0xb1,0x0c,0xbc,0x04,0xb4,0x0d,0xbd,
 0xe1,0xc8,0xec,0xc5,0xcd,0xe4,0xc4,0xed,0xd1,0xf1,0xdc,0xfc,0xd4,0xf4,0xdd,0xfd,
 0x36,0x8e,0x38,0x82,0x8b,0x30,0x83,0x39,0x96,0x26,0x9a,0x28,0x93,0x20,0x9b,0x29,
 0x66,0x4e,0x68,0x41,0x49,0x60,0x40,0x69,0x56,0x76,0x58,0x78,0x50,0x70,0x59,0x79,
 0xa6,0x1e,0xaa,0x11,0x19,0xa3,0x10,0xab,0x06,0xb6,0x08,0xba,0x00,0xb3,0x09,0xbb,
 0xe6,0xce,0xea,0xc2,0xcb,0xe3,0xc3,0xeb,0xd6,0xf6,0xda,0xfa,0xd3,0xf3,0xdb,0xfb,
 0x31,0x8a,0x3e,0x86,0x8f,0x37,0x87,0x3f,0x92,0x21,0x9e,0x2e,0x97,0x27,0x9f,0x2f,
 0x61,0x48,0x6e,0x46,0x4f,0x67,0x47,0x6f,0x51,0x71,0x5e,0x7e,0x57,0x77,0x5f,0x7f,
 0xa2,0x18,0xae,0x16,0x1f,0xa7,0x17,0xaf,0x01,0xb2,0x0e,0xbe,0x07,0xb7,0x0f,0xbf,
 0xe2,0xca,0xee,0xc6,0xcf,0xe7,0xc7,0xef,0xd2,0xf2,0xde,0xfe,0xd7,0xf7,0xdf,0xff
};
```

Picture 25: Sbox table from Skinny-AEAD documentation

It is clear that the tables match each other, so the calculation of the 8 Boolean functions follows.

The Boolean functions have been named as boolean_function_1, boolean_function_2, up to boolean_function_8 and the truth tables are going to be presented in a hexadecimal form. This way they will be more legible and they can be imported into SageMath in this form as well.



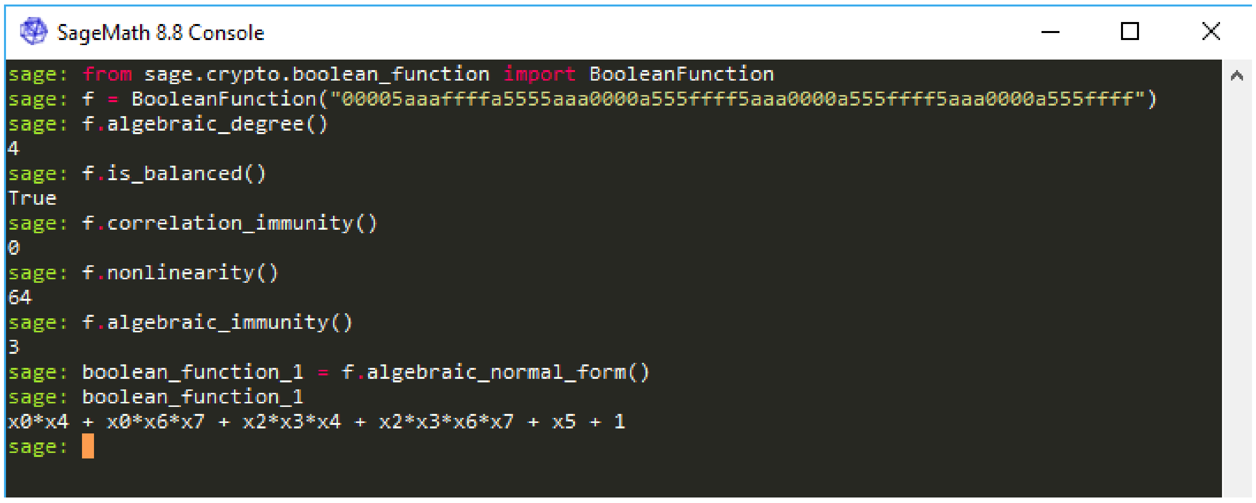
```
The 8 Boolean functions truth tables are presented below:
1st Boolean function: 0x5aaaffffa5555aaa0000a555ffff5aaa0000a555ffff5aaa0000a555ffff
2nd Boolean function: 0xffff0000ffff00000000ffff0000ffff0000ffff0000ffff0000ffff0000ffff
3rd Boolean function: 0xa555a555a555a555a555a555a555a555a555a555a555a555a555a555a555a555
4th Boolean function: 0xffa5aa00ff5a55a5aa00ff5a5500ffa5aa00ff5a5500ffa5aa00ff5a5500ff
5th Boolean function: 0x6933693369336933693369336933693369336933693369336933693369336933
6th Boolean function: 0xc0c0c0c0c0c0c03f3f3f3f3f3f3f3fc0c0c0c0c0c0c03f3f3f3f3f3f3f3f
7th Boolean function: 0x3f3f251500001a2a8040c0c040800000daeac0c0e5d5ffff7fbf3f3fbf7fffff
8th Boolean function: 0x8fcf9dc799c38bcb19831903994399c30b0b19031d070f0f8f4f8fcf0f8f0f0f
```

Picture 26: Truth tables of the 8 Boolean functions

The length of the hexadecimal characters is not equal for every Boolean function; this is due to the fact that the leading zeros have been skipped in some of them. However, they are going to be added to the SageMath tool as it can only process truth tables with length that are a power of 2. In this case the length needs to be 64 hexadecimal characters, so 256 bits.

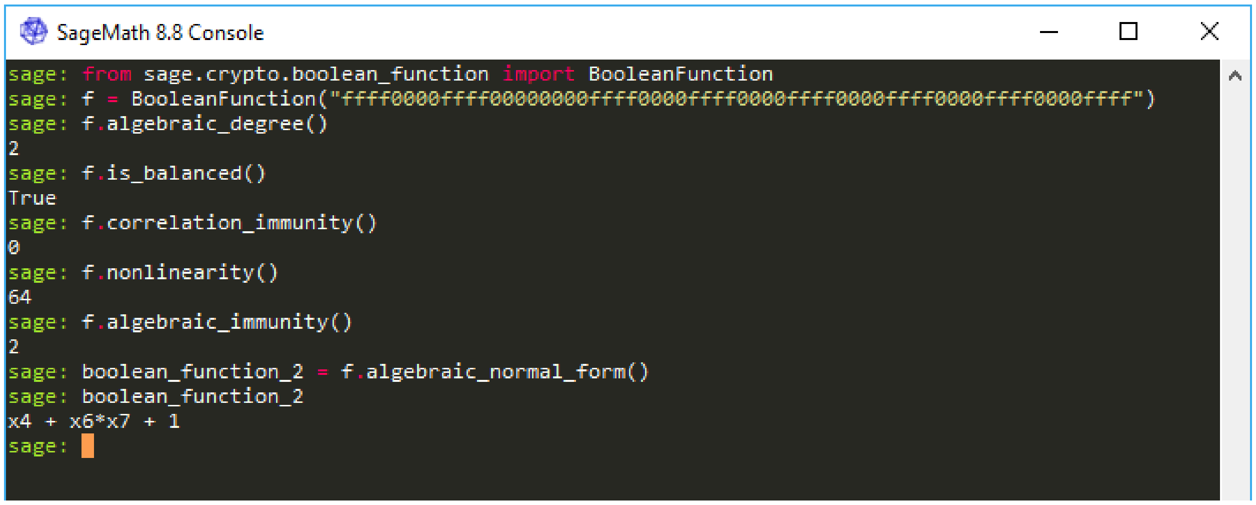
The Boolean functions are inserted into the SageMath tool. No details are going to be given for the commands since they were analyzed elaborately in the previous evaluations.

Apart from the properties of the Boolean functions, the algebraic forms were extracted as well. The results of all the functions will be presented below and afterwards, the properties of each function will be assessed as well.



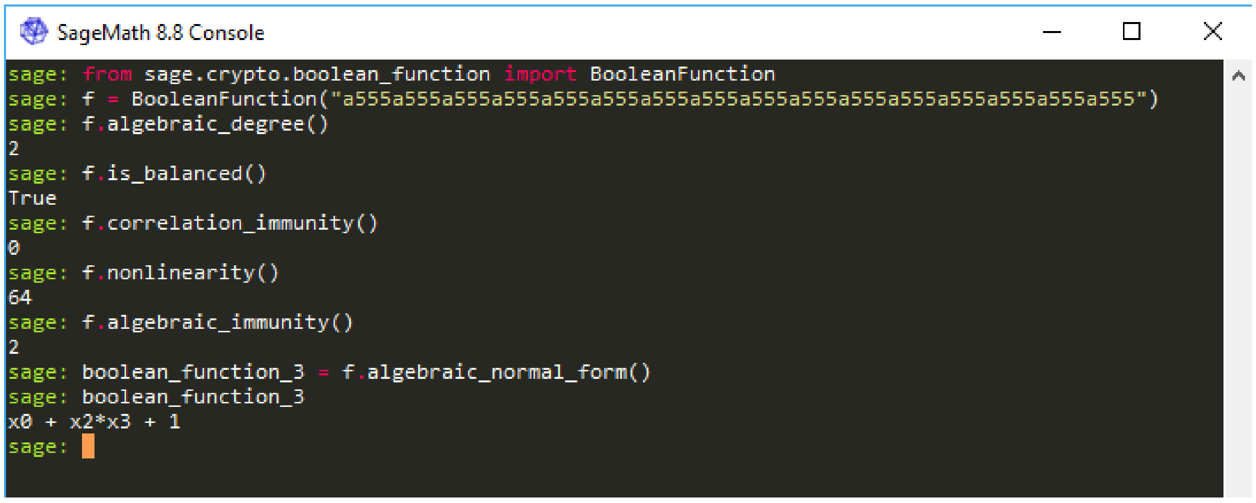
```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("00005aaaffffa5555aaa0000a555ffff5aaa0000a555ffff5aaa0000a555ffff")
sage: f.algebraic_degree()
4
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
3
sage: boolean_function_1 = f.algebraic_normal_form()
sage: boolean_function_1
x0*x4 + x0*x6*x7 + x2*x3*x4 + x2*x3*x6*x7 + x5 + 1
sage: 
```

Picture 27: Properties of Skinny's 1st Boolean function



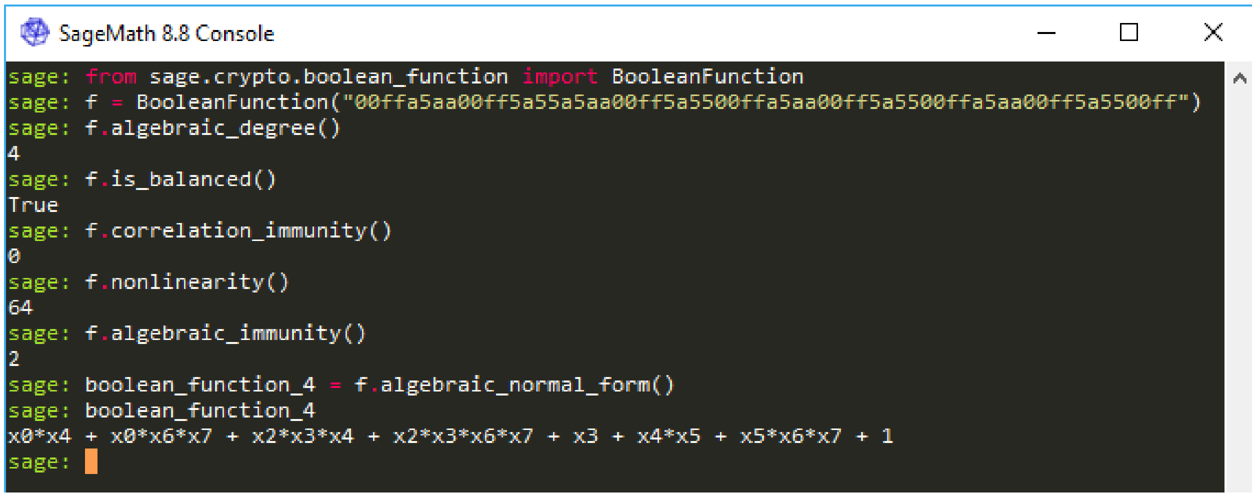
```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("ffff0000ffff00000000ffff0000ffff0000ffff0000ffff0000ffff0000ffff")
sage: f.algebraic_degree()
2
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
2
sage: boolean_function_2 = f.algebraic_normal_form()
sage: boolean_function_2
x4 + x6*x7 + 1
sage: 
```

Picture 28: Properties of Skinny's 2nd Boolean function



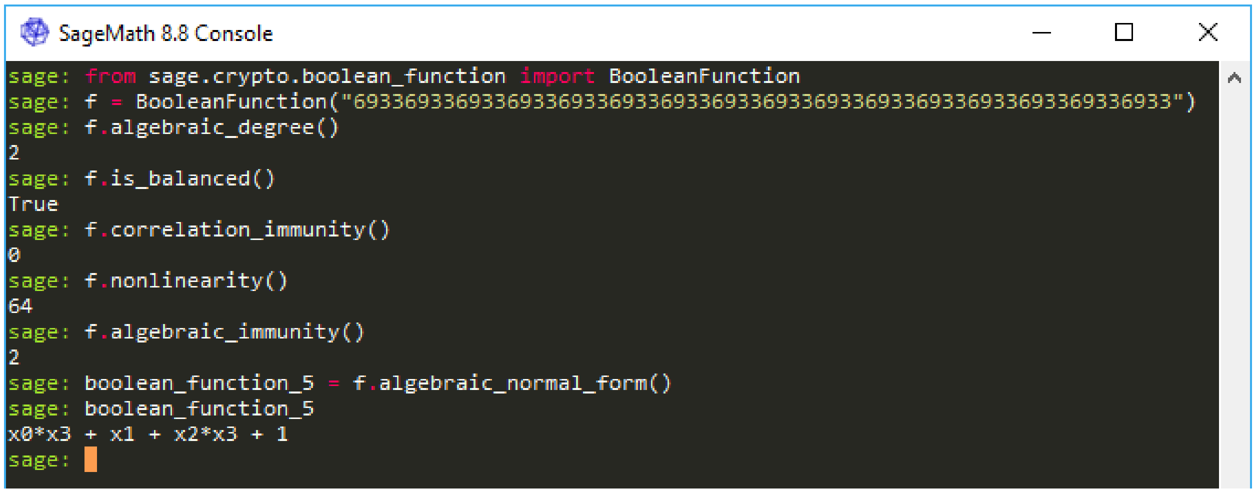
```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("a555a555a555a555a555a555a555a555a555a555a555a555a555a555a555a555")
sage: f.algebraic_degree()
2
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
2
sage: boolean_function_3 = f.algebraic_normal_form()
sage: boolean_function_3
x0 + x2*x3 + 1
sage: 
```

Picture 29: Properties of Skinny's 3rd Boolean function



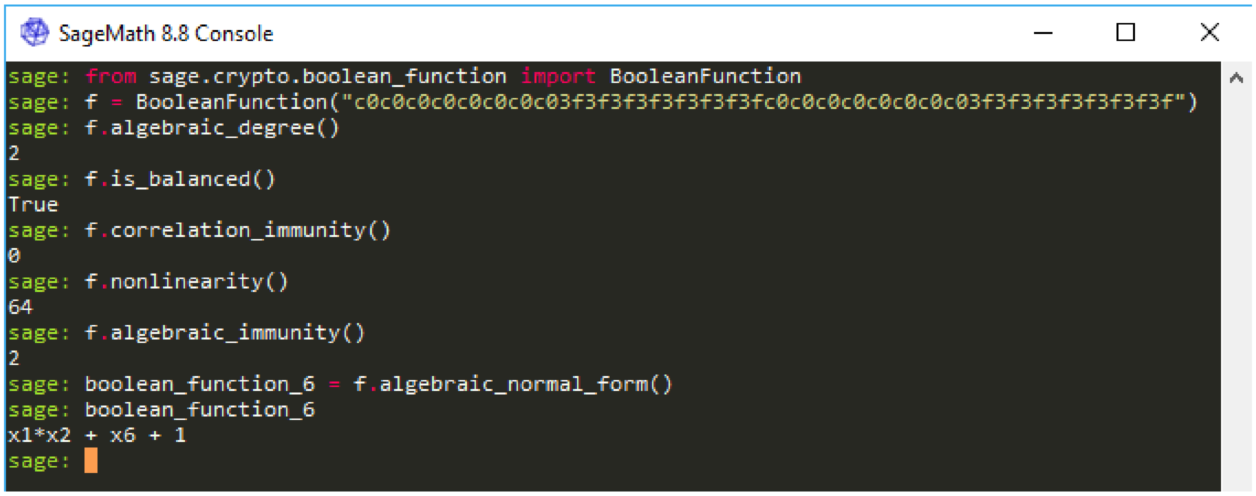
```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("00ffa5aa00ff5a55a5aa00ff5a5500ffa5aa00ff5a5500ffa5aa00ff5a5500ff")
sage: f.algebraic_degree()
4
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
2
sage: boolean_function_4 = f.algebraic_normal_form()
sage: boolean_function_4
x0*x4 + x0*x6*x7 + x2*x3*x4 + x2*x3*x6*x7 + x3 + x4*x5 + x5*x6*x7 + 1
sage: 
```

Picture 30: Properties of Skinny's 4th Boolean function



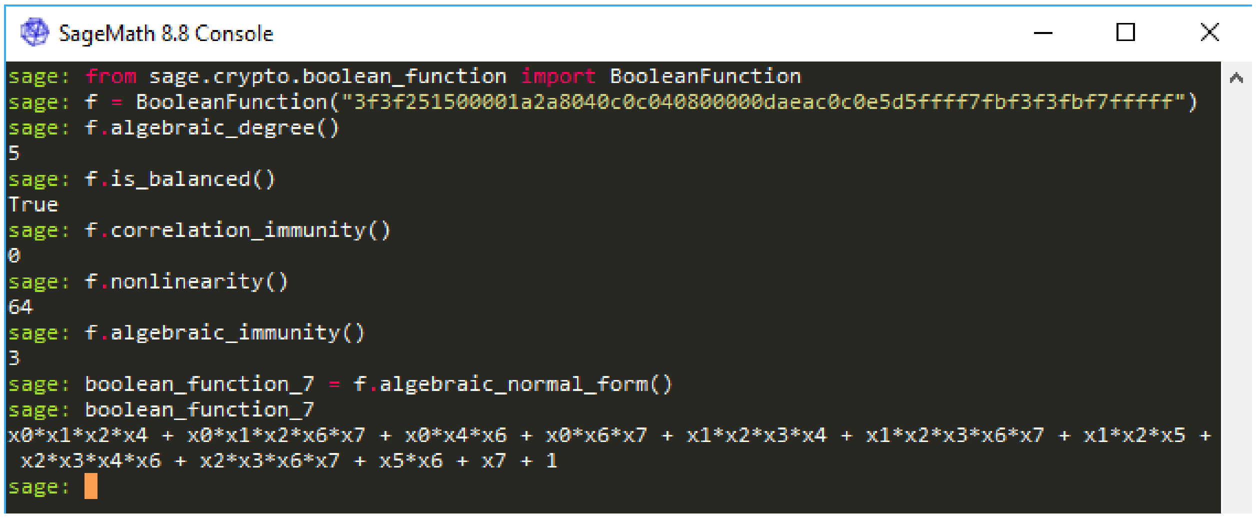
```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("6933693369336933693369336933693369336933693369336933693369336933")
sage: f.algebraic_degree()
2
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
2
sage: boolean_function_5 = f.algebraic_normal_form()
sage: boolean_function_5
x0*x3 + x1 + x2*x3 + 1
sage:
```

Picture 31: Properties of Skinny's 5th Boolean function



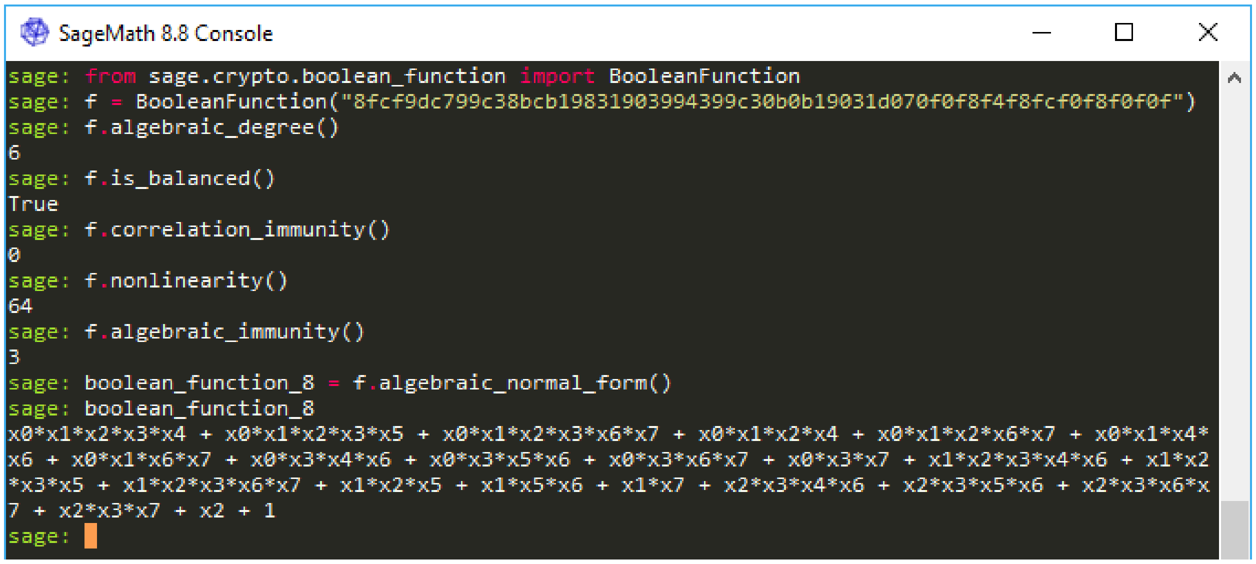
```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("c0c0c0c0c0c0c0c03f3f3f3f3f3f3f3fc0c0c0c0c0c0c0c03f3f3f3f3f3f3f3f")
sage: f.algebraic_degree()
2
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
2
sage: boolean_function_6 = f.algebraic_normal_form()
sage: boolean_function_6
x1*x2 + x6 + 1
sage:
```

Picture 32: Properties of Skinny's 6th Boolean function



```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("3f3f251500001a2a8040c0c040800000daeac0c0e5d5ffff7fbf3f3fbf7fffff")
sage: f.algebraic_degree()
5
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
3
sage: boolean_function_7 = f.algebraic_normal_form()
sage: boolean_function_7
x0*x1*x2*x4 + x0*x1*x2*x6*x7 + x0*x4*x6 + x0*x6*x7 + x1*x2*x3*x4 + x1*x2*x3*x6*x7 + x1*x2*x5 +
 x2*x3*x4*x6 + x2*x3*x6*x7 + x5*x6 + x7 + 1
sage: 
```

Picture 33: Properties of Skinny's 7th Boolean function



```
sage: from sage.crypto.boolean_function import BooleanFunction
sage: f = BooleanFunction("8fcf9dc799c38bcb19831903994399c30b0b19031d070f0f8f4f8fcf0f8f0f0f")
sage: f.algebraic_degree()
6
sage: f.is_balanced()
True
sage: f.correlation_immunity()
0
sage: f.nonlinearity()
64
sage: f.algebraic_immunity()
3
sage: boolean_function_8 = f.algebraic_normal_form()
sage: boolean_function_8
x0*x1*x2*x3*x4 + x0*x1*x2*x3*x5 + x0*x1*x2*x3*x6*x7 + x0*x1*x2*x4 + x0*x1*x2*x6*x7 + x0*x1*x4*
x6 + x0*x1*x6*x7 + x0*x3*x4*x6 + x0*x3*x5*x6 + x0*x3*x6*x7 + x0*x3*x7 + x1*x2*x3*x4*x6 + x1*x2
*x3*x5 + x1*x2*x3*x6*x7 + x1*x2*x5 + x1*x5*x6 + x1*x7 + x2*x3*x4*x6 + x2*x3*x5*x6 + x2*x3*x6*x
7 + x2*x3*x7 + x2 + 1
sage: 
```

Picture 34: Properties of Skinny's 8th Boolean function

It is easily observable that most of the properties are identical to most Boolean functions.

All of the functions are balanced, which means that their truth table has the same number of ones (1s) and zeros (0s) and as a result these functions are possibly vulnerable to correlation attacks. Also, the nonlinearity is 64 in every Boolean function. Since all of them contain 8 variables, the maximum possible nonlinearity according to the Bent function is $2^{8-1} - 2^{8/2-1}$, which is equal to 120. As a result this is not ideal and vulnerabilities could exist in linear attacks. The correlation immunity is 0, but as with the Boolean function of the Grain algorithm, since the keystream generation is not directly connected to these functions, this vulnerability is of no concern.

Last but not least, the algebraic immunity is equal to 2 in the first 6 functions and equal to 3 in the last 2. This is obvious by checking the algebraic degree and the complexity of the algebraic form of the Boolean functions.

In addition, all the functions use 8 variables, so the maximum possible algebraic immunity they may attain is n/2 = 8/2 = 4. Consequently, the last two (2) functions are more robust against algebraic attacks from the others, since their value is closer to the maximum.

6. CONCLUSION

Based on the results from the practical part in Chapter 5 some conclusions are going to be drawn regarding the security level of the Lightweight cryptographic algorithms related to the Boolean functions utilized.

At first, it should be clarified that it is extremely challenging to design a Boolean function that can satisfy all the cryptographic properties and be immune to all the related cryptanalytic attacks. Nevertheless, this is not always negative and does not give the adversary a way to break the system or the algorithm all the times. This is due to the fact that the Boolean functions can be utilized in some parts of the algorithm implementation that is unrelated with the generation of the keystream. As a result, even if the Boolean function of a cryptographic algorithm is compromised, it may be infeasible for the attacker to extract the secret key or the plaintext from the ciphertext.

To be more specific using an example, in the 5th chapter most of the Boolean functions were vulnerable to correlation attacks due to the correlation immunity property result, which was zero (0). However, these Boolean functions were not utilized in the process of the keystream creation, so this is not necessarily a "weak point" that could give raise to a successful attack. The Grain algorithm exists for many years and despite this vulnerability there have not been cryptanalytic attacks able to exploit this and break the algorithm.

However, this is not the case in many cryptographic algorithms, as they use Boolean functions as a vital part for the keystream generation. The less cryptographic properties satisfied by these functions, the higher the possibility to compromise the algorithm.

In any case, due to the fact that some cryptographic properties are not being met in several Boolean functions being used in lightweight algorithms, it is of great interest to further focus on whether this lack of cryptographic criteria can be effectively exploited.

To sum up, all these ascertainments establish a clear direction for the research community to investigate even more the issue and clarify if further actions need to be taken, so as to increase the level of security of the Boolean function utilized in the cryptographic algorithms.

TABLE OF TERMINOLOGY

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Lightweight Algorithm	Αλγόριθμος Χαμηλών Απαιτήσεων
Conventional	Συμβατική
Cryptography	Κρυπτογραφία
Stream Cipher	Κρυπτογραφικός Αλγόριθμος Ροής
Block Cipher	Κρυπτογράφηση ανά μπλοκ - ομάδες
Boolean Function	Λογική Συνάρτηση
Confidentiality	Εμπιστευτικότητα
Data Integrity	Ακεραιότητα Δεδομένων
Non-Repudiation	Βεβαίωση Ταυτοπροσωπίας
Authentication	Ταυτοποίηση
Internet of Things	Διαδίκτυο των Πραγμάτων
Plaintext	Απλό Κείμενο (Αναγνώσιμο)
Ciphertext	Κρυπτογραφημένο Κείμενο
Keystream	Ψευδοτυχαία Ακολουθία bits
Vulnerabilities	Ευπάθειες
Eavesdrop	Υποκλέβω

Exploit	Εκμεταλλεύομαι
Unauthorized	Μη Εξουσιοδοτημένος
Adversary	Αντίπαλος
Balancedness	Εξισορρόπηση
Nonlinearity	Μη Γραμμικότητα
Avalanche Criterion	Κριτήριο Χιονοστιβάδας
Correlation Immunity	Ανοσία Συσχέτισης
Algebraic Immunity	Αλγεβρική Ανοσία
Truth Table	Πίνακας Αληθείας

ABBREVIATIONS - ACRONYMS

NKUA	National and Kapodistrian University of Athens
NIST	National Institution of Standards and Technology
ISO	International Organization for Standardization
SHA	Secure Hash Algorithm
MAC	Message Authentication Code
AES	Advanced Encryption Standard
DES	Data Encryption Standard
RC4	Rivest Cipher
RSA	Rivest-Shamir-Adleman
DSA	Digital Signature Algorithm
RFID	Radio Frequency Identification
IoT	Internet of Things
AEAD	Authenticated Encryption with Associated Data
COA	Ciphertext Only Attack
CPA	Chosen Plaintext Attack
KPA	Known Plaintext Attack
CCA	Chosen Ciphertext Attack

IV	Initialization Vector
RAM	Random Access Memory
ROM	Read Only Memory
SCA	Side Channel Attack
CRYPTREC	Cryptography Research and Evaluation Committees
IDE	Integrated Development Environment

REFERENCES

[1] Alyson Brown, “The Basics of Cryptography”, Towards data science, 28 Jan 2019; https://towardsdatascience.com/the-basics-of-cryptography-80c7906ba2f7 [Accessed 12/12/2019]

[2] Margaret Rouse, “Cryptography”, Search Security, Sep 2018; https://searchsecurity.techtarget.com/definition/cryptography [Accessed 12/12/2019]

[3] Finjan Team, “What is Non-Repudiation?”, Finjan blog, 27 Feb 2017; https://blog.finjan.com/what-is-non-repudiation/ [Accessed 12/12/2019]

[4] Jerrold Siegel, “Cryptography”, umls.edu (University of Missouri – St.Louis),

http://www.umsl.edu/~siegelj/information_theory/projects/des.netau.net/Cryptography%20and%20goals.html [Accessed 12/12/2019]

[5] Rafael Almeida, “Symmetric and Asymmetric Encryption”, hackernoon, 9 Dec 2017; https://hackernoon.com/symmetric-and-asymmetric-encryption-5122f9ec65b1 [Accessed 12/12/2019]

[6] Austin Chamberlain, “Applications of Cryptography”, University College London, 12 Mar 2017, https://blogs.ucl.ac.uk/infosec/2017/03/12/applications-of-cryptography/ [Accessed 12/12/2019]

[7] Jayanthi Manikandan, “Basics of Cryptography”, Info Security Institute, 7 Apr 2018, https://resources.infosecinstitute.com/basics-of-cryptography-the-practical-application-and-use-of-cryptography/#gref [Accessed 12/12/2019]

[8] Sreeja Rajesh, Varghese Paul, Varun G. Menon, and Mohammad R. Khosrav,” A Secure and Efficient Lightweight Symmetric Encryption Scheme for Transfer of Text Files between Embedded IoT Devices”, Molecular Diversity Preservation International, 24 Feb 2019

[9] Wolf SSL, “What is a Block Cipher”, 19 Dec 2014, https://en.wikipedia.org/wiki/Block_cipher#cite_note-NIST-modes-22 [Accessed 12/12/2019]

[10] Maneesh Singh, “Difference between Block Cipher and Stream Cipher”, GeeksforGeeks, 20 May 2019,https://www.geeksforgeeks.org/difference-between-block-cipher-and-stream-cipher/ [Accessed 12/12/2019]

[11] John Carl Villanueva, “Stream Ciphers vs Block Ciphers”, JSCAPE, 12 May 2015, https://www.jscape.com/blog/stream-cipher-vs-block-cipher [Accessed 12/12/2019]

[12] Jerrold Siegel, “History of DES”, umls.edu (University of Missouri – St.Louis), 13 May 2013, http://www.umsl.edu/~siegelj/information_theory/projects/des.netau.net/des%20history.html [Accessed 12/12/2019]

[13] Margaret Rouse, “Data encryption Standard (DES)”, Search Security, Nov 2014,https://searchsecurity.techtarget.com/definition/Data-Encryption-Standard [Accessed 12/12/2019]

[14] Jasmine Henry, “3DES is officially Being Retired”, Cryptomathic, 3 Aug 2018, https://www.cryptomathic.com/news-events/blog/3des-is-officially-being-retired [Accessed 12/12/2019]

[15] Jon Callas, “Triple DES: How strong is the data encryption standard? “, Search Security, 31 May 2017,https://searchsecurity.techtarget.com/tip/Expert-advice-Encryption-101-Triple-DES-explained [Accessed 12/12/2019]

[16] Margaret Rouse, “Advanced Encryption Standard (AES)”, Search Security, Mar 2017, https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard [Accessed 12/12/2019]

[17] Tutorials Point, “Advanced Encryption Standard”, 24 Jun 2015,

https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm [Accessed 12/12/2019]

[18] Wikipedia, "RC5", 15 Dec 2019, https://en.wikipedia.org/wiki/RC5 [Accessed 12/12/2019]

[19] Wikipedia, "Blowish (cipher)", 5 Dec 2019, https://en.wikipedia.org/wiki/Blowfish_(cipher) [Accessed 12/12/2019]

[20] Sally Jarkas, "RC4 Encryption Algorithm", Geek for Geeks, 15 Sep 2015, https://www.geeksforgeeks.org/rc4-encryption-algorithm/ [Accessed 12/12/2019]

[21] Wikipedia, "Salsa20", 8 Dec 2019, https://en.wikipedia.org/wiki/Salsa20[Accessed 12/12/2019]

[22] Viva Differences, "What Is The Difference Between Block Cipher & Stream Cipher?", 26 May 2016 https://vivadifferences.com/what-is-the-difference-between-block-cipher-stream-cipher/

[Accessed 12/12/2019}

[23] Tutorials Point, "Attacks on Cryptosystems", 24 Jun 2015,

https://www.tutorialspoint.com/cryptography/attacks_on_cryptosystems.htm [Accessed 12/12/2019]

[24] Wikipedia, "Brute-force Attack", 9 Nov 2019, https://en.wikipedia.org/wiki/Brute-force_attack [Accessed 12/12/2019]

[25] Wikipedia, "Man in the Middle", 22 Nov 2019, https://en.wikipedia.org/wiki/Man-in-the-middle_attack [Accessed 12/12/2019]

[26] Imperva, "What is Social Engineering", 3 Aug 2016, https://www.imperva.com/learn/application-security/social-engineering-attack/ [Accessed 12/12/2019]

[27] Margaret Rouse, "Side-Channel Attacks", May 2019, https://searchsecurity.techtarget.com/definition/side-channel-attack [Accessed 12/12/2019]

[28] Chris Kowalczyk,"Ciphertext-Only (Known Ciphertext) Attack", Crypto-IT, 15 Nov 2013, http://www.crypto-it.net/eng/attacks/known-ciphertext.html [Accessed 12/12/2019]

[29] Wikipedia, "Dictionary Attack", 15 Dec 2019, https://en.wikipedia.org/wiki/Dictionary_attack [Accessed 12/12/2019]

[30] Gireesh Pandey, "Cryptographic Characteristics of Boolean Functions", Defense Research and Development Organization, Oct 2012

[31] Wikipedia, "Balanced Boolean Function", 17 Dec 2018,

https://en.wikipedia.org/wiki/Balanced_boolean_function [Accessed 12/12/2019]

[32] Thomas W. Cusick, "Correlation Immunity", Science Direct, 2017,

https://www.sciencedirect.com/topics/mathematics/correlation-immunity [Accessed 12/12/2019]

[33] Citizendium, "Algebraic attack", 25 Jun 2010, http://en.citizendium.org/wiki/Algebraic_attack [Accessed 12/12/2019]

[34] Alex Biryukov and Leo Perrin, "State of the Art in Lightweight Symmetric Cryptography", University of Luxembourg and Inria Paris, Jun 2017

[35] Kerry A. McKay, Larry Bassham, Meltem Sonzem Turan and Nicky Mouha, "Report on Lightweight Cryptography", National Institute of Standards and Technology, Mar 2017

[36] Dan Mitchell, "Internet of Things (IoT) What it is and why it matters", Statistical Analysis System Institute, https://www.sas.com/el_gr/insights/big-data/internet-of-things.html [Accessed 12/12/2019]

[37] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe, “PRESENT: An Ultra-Lightweight Block Cipher”, Ruhr-University Bochum, Technical University Denmark and France Telecom R&D, Sep 2007

[38] SONY, “Clefia: The 128-bit Block Cipher”, 2007 https://www.sony.net/Products/cryptography/clefia/ [Accessed 12/12/2019]

[39] Zheng Gong1, Svetla Nikova1 and Yee Wei Law, “KLEIN: A New Family of Lightweight Block Ciphers”, University of Twente, Katholieke Universiteit Leuven and University of Melbourne, Jun 2011

[40] Hell Martin and Johansson Thomas, “Security Evaluation of Stream Cipher Enocoro-128v2”, Lund University, 2010

[41] Christophe De Canniere and Bart Preneel, “Trivium”, Katholieke Universiteit Leuven, 2006

[42] Martin Hell, Thomas Johansson, Alexander Maximov, “A Stream Cipher Proposal: Grain-128”, Lund University, Aug 2006

[43] National Institute of Standards and Technology, “DRAFT Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process”, 27 Aug 2018

[44] National Institute of Standards and Technology, “Lightweight Cryptography”, 26 Apr 2018, https://csrc.nist.gov/Projects/lightweight-cryptography/ [Accessed 12/12/2019]

[45] Wikipedia, “SageMath”, 9 Dec 2019, https://en.wikipedia.org/wiki/SageMath [Accessed 12/12/2019]

[46] Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sonnerup, Hirotaka Yoshida, “Grain-128AEAD - A lightweight AEAD stream cipher”, Lund University FHNW, AIST, 2019

[47] Rusydi H. Makarim and Yann Laigle Chapuy, “Boolean Functions”, SageMath, 13 Oct 2016, http://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/boolean_function.html [Accessed 12/12/2019]

[48] Hongjun Wu and Tao Huang, “TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms”, Division of Mathematical Sciences Nanyang Technological University, 27 Sep 2019

[49] Christof Beierle, Jeremy Jean, Stefan Kolbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim, “Skinny-AEAD and Skinny-Hash”, University of Luxembourg, ANSSI, Cybercrypt A/S, Ruhr Bochum University, Nanyang Technological University, Singapore, NTT Secure Platform Laboratories, Rambus Cryptography, 2019

[50] Picture 1: Symmetric Cryptography, Kelley Robinson, “What is Public key Cryptography? ”, Twilio Blog, 21 Sep 2018, https://www.twilio.com/blog/what-is-public-key-cryptography [Accessed 30/12/2019]

[51] Picture 2: Asymmetric Cryptography, Kelley Robinson, “What is Public key Cryptography?”, Twilio Blog, 21 Sep 2018, https://www.twilio.com/blog/what-is-public-key-cryptography [Accessed 30/12/2019]

[52] Picture 3: Block Cipher, Yogesh Prasad, “Block Cipher”, Hackers Interview, 8 Jul 2018 http://blog.hackersinterview.com/cryptography/block-cipher/ [Accessed 30/12/2019]

[53] Picture 4: Stream Cipher, CryptoSmith, “Stream Ciphers”, 7 Jun 2007, https://cryptosmith.com/2007/06/07/stream-ciphers/ [Accessed 30/12/2019]]

[54] Picture 5: Passive Attacks, Nihad Hassan, “What is an Active Attack vs Passive Attack Using Encryption? “, Venafi, 5 Mar 2019, https://www.venafi.com/blog/what-active-attack-vs-passive-attack-using-encryption [Accessed 30/12/2019]

[55] Picture 6: Active Attacks, Tech Differences, “Difference Between Active and Passive Attacks”, 2 Feb 2018, https://techdifferences.com/difference-between-active-and-passive-attacks.html [Accessed 30/12/2019]

[56] Picture 7: Brute-Force Attack, Manage Engine, “What is Brute-Force Attack?”, https://www.manageengine.com/log-management/cyber-security-attacks/what-is-brute-force-attack.html[Accessed 30/12/2019]

[57] Picture 8: Tove Marks, “Man In the Middle Attacks: Here’s what You need to Know”, VPN Overview, 21 Jun 2019, https://vpnoverview.com/internet-safety/cybercrime/man-in-the-middle-attacks/ [Accessed 30/12/2019]

[58] Picture 9: Social Engineering Attack, Chris Reaburn, “What Your Business Needs to Know About Social Engineering Attacks”, Nextiva Blog, 26 Feb 2019, https://www.nextiva.com/blog/social-engineering-attacks.html [Accessed 30/12/2019]

[59] Picture 10: Dictionary Attack, Hiba Sh, “Dictionary Attack: everything You Need To Know”, The VPN Guru, https://thevpn.guru/dictionary-attacks-explained/ [Accessed 30/12/2019]

[60] Picture 11: Side-Channel Attacks, Daniel Genkin, Adi Shamir, Eran Tromer, “ RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”, Tel Aviv University, https://m.tau.ac.il/~tromer/acoustic/ [Accessed 30/12/2019]

[61] Picture 12: Ciphertext Only Attacks (COA), Hackers Arise, Cryptography Basics, Part 2: Attack Models for Cryptanalysis, 19 Sep 20 https://www.hackers-arise.com/post/2019/04/30/cryptography-basics-part-2-attack-models-for-cryptanalysis [Accessed 30/12/2019]

[62] Picture 13: Chosen-Plaintext Attack (CPA), Hackers Arise, Cryptography Basics, Part 2: Attack Models for Cryptanalysis, 19 Sep 2020, https://www.hackers-arise.com/post/2019/04/30/cryptography-basics-part-2-attack-models-for-cryptanalysis [Accessed 30/12/2019]

[63] Picture 14: Known-Plaintext Attack (KPA), Johannes Gutenberg Universitat Mainz, “Cryptology: Known Plaintext Attack”, 17 Dec 2017, https://www.staff.uni-mainz.de/pommeren/Cryptology/Classic/1_Monoalph/knownplain.html

[Accessed 30/12/2019]

[64] Picture 15: Chosen Ciphertext Attack (CCA), Hackers Arise, Cryptography Basics, Part 2: Attack Models for Cryptanalysis, 19 Sep 2020 https://www.hackers-arise.com/post/2019/04/30/cryptography-basics-part-2-attack-models-for-cryptanalysis [Accessed 30/12/2019]

[65] Picture 16: Algebraic Expression of Boolean Function, Aslesha, “Equal Boolean Functions”, DocSity, 27 Apr 2013, https://www.docsity.com/en/equal-boolean-functions-discrete-mathematics-lecture-slides/317478/ [Accessed 30/12/2019]

[66] Picture 17: Truth Table of Boolean Function, Aslesha, “Equal Boolean Functions”, DocSity, 27 Apr 2013, https://www.docsity.com/en/equal-boolean-functions-discrete-mathematics-lecture-slides/317478/ [Accessed 30/12/2019]

[67] Picture 18: Security in IoT Devices, Cheap SSL Security, “IoT Security: Understanding PKI’s Role in Securing Internet of Things”, https://cheapsslsecurity.com/blog/iot-security-understanding-pki-role-in-securing-internet-of-things/ [Accessed 30/12/2019]

[68] Picture 19: SageMath, SageMath, http://www.sagemath.org/ [Accessed 30/12/2019]