Multiple encryption method for improving the security in communications

LAB University of Applied Sciences
Bachelor of Engineering, Information and Communication Technology
Faculty of technology
2021
Sergio Barrachina Rico

Abstract

Author(s)	Publication type	Completion year
Barrachina Rico, Sergio	Thesis, UAS	2021
	Number of pages	
	60	

Title of the thesis

Multiple encryption method for improving the security in communications

Degree

Bachelor of Engineering, Information and Communication Technology

Name, title and organisation of the thesis supervisor

Ismo Jakonen, Senior Lecturer, Media Technology

Name, title and organisation of the client

Abstract

The frequent use of technology to send and share some personal and confidential data needs to have secure techniques and methods to transmit it securely. For this purpose, the work aims to introduce and describe a method to improve security in communication and showing its implementation.

This method is based on two main principles: encrypting data multiple times using different algorithms and applying compression. Using different methods to encrypt data multiple times provides huge protection. In addition, the compression adds more security layers to this process and reduces the size of the information transmitted.

These features are described and analyzed to see their advantages and drawbacks and how are applied to the method defined. In addition, a great overview of encryption and cryptography is made in order to see the evolution and importance of these fields over the years and the current techniques and methods which are used nowadays to protect our information.

Keywords

Encryption, cryptography, compression, security.

Contents

1 Introduction

Encryption has been always present in our lives since the need of protecting data and information arose. Some data does not have to be known and has to be transmitted securely. For that purpose, some transformations are applied to make the data illegible to someone who is not involved in the communication process. This is the purpose of encryption.

Nowadays, encryption has become an essential aspect of our lives. We are sending every day and every time information through the internet in order to check the last posts in our social networks accounts or sending our bank account data to buy online. This information can be stolen by cybercriminals if this exchange of information is not made properly. This is the reason why encryption techniques, which are every time applied when some data is sent, are so important.

Cryptography is the discipline which studies the best ways and methods to protect the information exchanged during communications. The experts in this field are responsible for developing the best techniques to ensure the security and protection of our data. They try to find the best way to encrypt the data when this is transmitted.

Currently, communications are quite secure as there are lots of algorithms and methods that provide this security layer. However, after some time, some of them become insecure and are not used.

The goal of the thesis is to introduce a method to improve security in communications. This method applies multiple encryption methods and compression to the data which is going to be sent. This process aims to reduce the size of the data sent, in order to increase the speed in communications; and adding multiple security layers that complicate the decryption process for someone who is trying to get this data.

For this purpose, a great overview of cryptography and encryption will be given. Definition of some important terms, history and classification of encryption methods will be presented in order to get familiar with them. The purpose of this is to show the current state of the field to make a comparison with the solution suggested in the thesis.

Once all this background is given, the multiple encryption method will be presented. About this, its operating process and its main features will be described and analyzed to see its strengths and weaknesses. In addition, an implementation of this method will be shown.

2 Encryption

2.1 Cryptography: definition

Cryptography is the discipline which studies the principles, methods and means to transform data to hide its meaning and guarantee its integrity. Nowadays cryptography is a term associated to computer science, but it is not a matter that did not exist before the creation of computers. The need of protecting data has existed always so that a different person from the one who has to receive the information could not read the message or the information sent. (González-Tablas Ferreres et al. 2018a, 4.)

Currently, the use of ICT technologies for a big part of the activities we do in our daily life has increased the need of improving the protection of information. Most of our personal data is stored somewhere in the cloud and is transmitted through the internet. That is why cryptography is so important nowadays.

Cryptography has four main objectives:

- **Confidentiality:** Only the users authorized can access the information (Alcover Garau 2008, 1).
- **Integrity**: the original data or information is not altered after being sent to the recipient (Alcover Garau 2008, 1).
- **Authentication:** the parts involved in the communication must identify themselves (Alcover Garau 2008, 1).
- **No repudiation:** to avoid that the author of the information might deny that he/she is the author or has sent the data (Alcover Garau 2008, 1).

2.2 Encryption: definition

Encryption is the process to transform data or information into illegible series of characters using for that purpose a key. Algorithms designed for this objective protect the information making some changes in the original data using the key mentioned previously. Without the key, the real data transmitted cannot be known. (Ionos 2019.)

This technique allows securely sending information. The data can only be deciphered by the ones who have the key. The information intercepted during its transmission will be unreadable as it has been ciphered previously and needs to be decrypted in order to get the real content of the message. (Ionos 2019.)

2.3 History

As explained in the previous section, cryptography has a close relationship with computer science, but it existed already before the creation of the first computer. Cryptography is composed of the Greek words kryptós (hidden, secret) and graphos (to write), so it means hidden writing. (Ribagorda 2010.) So, in its definition, we can see that cryptography is not only related to computer science.

Cryptography had relevant importance in some military conflicts in order to send information avoiding enemies could read its content. However, one of the first examples found in history about hidden messages is the hieroglyphs from Ancient Egypt. There are non-standard hieroglyphs which provided more mystery and drama to the stories they were telling and also making them more difficult to understand. (Jesús Velasco 2014.)

In the Bible, there are references to Atbash. It was a substitution letter system used to cipher messages in 600 BC. It consisted of changing the first letter to the last one, the second to the second to last and so on. (Wikipedia 2015.)

In the 5th century BC, there are more examples of hidden messages in Sparta. They used an object called scytale (shown in figure 1). In this, a parchment was rolled around the stick (scytale) and the message was written along the scytale. To see the content of the message, the recipient had to use a stick with the same diameter as the scytale used to write it. (Jesús Velasco 2014.)

Figure 1. Spartan scytale (Wikipedia 2008)

Ancient Rome is the origin of one of the most well-known methods to encrypt information called Cesar cipher. It is based on the letters shift. Each one of the letters is replaced for another which is a fixed number of letters towards. It is known that Julio Cesar used an offset of 3 letters, so for example the letter A will become D in the ciphered message. (Jesús Velasco 2014.)

Moving towards in history, in the 9th century, it was born one of the most important advances in breaking ciphered messages to discover its content thanks to Al-Kini, the frequency analysis. It consists in measuring the frequency in which the letters or symbols appear in a text. In this way, the original symbol that does not appear in the ciphered message will be discovered owing to the probabilities calculated. (Jesús Velasco 2014.)

In the Renaissance, Leon Battista created a new method to cipher messages. It used different alphabets in order to protect the content of the information. In the previous techniques like Caesar cipher, the letter A was always substituted by the letter D (using an offset of three letters). So it was easy to decrypt. However, with this new method, the letter A maybe could be the letter D but other time might be the number 9 for example. In this way, the frequency analysis was not very effective. Therefore, Albertti was one of the creators of the polyalphabetic substitution methods. (Ribagorda Garnacho 2010.)

The cipher created by Albertti caused the birth of the disk cipher. It was composed of two concentric crowns. The inside one had the ciphered alphabet and was fixed, the external one had the alphabet printed without cipher and it could turn around its center. In this way, each letter of the alphabet matches another one of the ciphered alphabet. (Servos 2010.)

Over the years, cryptography will become more important especially in military conflicts. In the Crimea War, the United Kingdom took advantage thanks to Charles Babbage. Babbage worked on deciphering the Vignère codes, (which consists in a polyalphabetic substitution method) which were considered very hardy to decrypt. (Jesús Velasco 2014.)

In 1883, the Military Sciences Magazine from France published a treaty about the six basic principles that a cryptographic system should accomplish. They were written by Auguste Kerchoffs, linguist and cryptographer from the Netherlands. These principles will be explained in more detail later. (Jesús Velasco 2014.)

In the Second World War, cryptography became very important to change the direction of the conflict. Germany achieved a good position in the war thanks to the Enigma machine. It was invented in 1918. It was battery-powered and portable. Axis officials and forces used the Enigma machine to protect their communications during the military conflict. The

information was transmitted in morse code by wireless radio. This was easy to intercept, but its cipher made them incomprehensible for their enemies. (Jesús Velasco 2014.)

The Enigma machines (an example is shown in figure 2) used machines instead of ciphers or codebooks. It was able to produce 159 million million million combinations of cipher using an electromechanical cipher machine. First the message written to the machine is encoded with a movable mechanical rotor engraved with letters, and then by a plug board of electric circuits. In the next image, there is an example of Enigma machine. (The British Museum 2021.)

Figure 2. Enigma machine (Jszigetvari 2003)

Ciphering trusted on the initial position of the rotors, which could be removed, rotated and replaced. The text generated was illegible unless the recipient of the message knew the rotor and plug board settings on the day the ciphered text was produced. For security, Germans reset Enigma machines at midnight every night because the machine settings were the key in order to decipher the messages sent. (The British Museum 2021.)

Apart from Enigma, during Second World War the United States used another method that was applied in the First World War successfully. To encrypt their communication, the Americans used the language of the Native Americans. The United States Marine had

approximately 500 of natives that ciphered messages in their mother tongue to avoid the Japanese army could understand the messages they transmitted.

After the Second World War, cryptography experienced a great growth. In 1948, Claude Shannon, known as the creator of the information theory (it studies the quantification, storage and communication of information), published *A Communications Theory of Secrecy Systems*, a paper which talks about cryptography from the point of view of information theory. It is one of the starting points of the modernization of cipher methods to transform them into advanced mathematical processes. (Jesús Velasco 2014.)

The growth of computation made computers key elements of cryptography owing to their calculation capacity. This field became secret for most countries like the USA. From the mid-1950s to the mid-1970s, the NSA (National Security Agency) blocked publications or studies about cryptography. After this period, cryptography became a more public area and investigations started to be accessible for people out of the field. The first public advance in cryptography arrived on March 17th in 1975. IBM developed the cipher algorithm DES (Data Encryption Standard). In 2001, DES will be replaced by AES (Advanced Encryption Standard) that will become standard after five years of inspection. (Jesús Velasco 2014.)

These algorithms are examples of symmetric cryptography (it uses the same key for sender and receiver), which will be described later in more detail. In 1976, asymmetric cryptography was born thanks to Whitfield Diffie and Martin E. Hellman who wrote the paper *New Directions in Cryptography.* Nowadays, this kind of encryption is very used in the transactions we make on the internet. (Jesús Velasco 2014.)

As seen previously, cryptography has had a great impact and importance in history. Furthermore, currently, it is essential in most of the things we do in our daily life. This will be shown during the following sections, which will provide deep background and knowledge in cryptography and encryption.

3 Classic encryption methods

3.1 Transposition methods

The transposition methods consist in changing the position of the characters of a text. It will have the same characters that the original text after applying the technique. They are symmetric algorithms because each part of the communication, sender and receiver, use the same key in order to cipher and decipher the information. (Ramió Aguirre 2016b.)

There are different ways to make the transposition of the characters: by groups, series and columns or rows. These methods will be described in more detail in the following sections.

3.1.1 Groups

In the transposition by groups, groups of characters of the same length are taken. To make the transposition in each group, an order is defined to see in which position each character will be inside the group (Ramió Aguirre 2016b.). This will be explained better in the next example.

We have this message: ENCRYPTION THESIS

The text is divided in groups of 4 characters:

ENCR YPTI ONTH ESIS

The permutation order of the letter will be this: 2413. This means that the first letter of the ciphered text will be the second one in the original text, the second will be the fourth one and so on. Using this, the message will look like this:

NREC PIYT NHOT SSEI

3.1.2 Series

The transposition by series uses a similar perspective to make the changes in the positions of the characters. The ciphered message is the result of sorting the original one as chains of sub-messages. To do that, the technique uses different series of numbers that define the order of each sub-message. (González-Tablas Ferreres 2018b.)

Example

The message which is going to be changed is this: I AM WRITING THE THESIS.

The series that encrypt the message will be the next ones:

- S_1 = {2, 4, 6, 8, 10, 12, 14}
- S_2 = {1, 3, 5, 7, 9, 11, 13}
- S_3 = {15,19, 16, 18, 17}

Taking into account this, the order of the characters will be this:

2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15, 19, 16, 18, 17

So the encrypted message will look this way:

AWIIGHT IMRTNTE HSEIS

The good point of this method is that the message will be more difficult to decrypt for a person that does not know the key because the complexity has increased by using different series to define the order of the transpositions.

3.1.3 Columns/rows

The text which is going to be ciphered is distributed in a table with a determined number of rows and columns. Each cell of the table contains a character, and they are put on the table from left to right and from up to down. (González-Tablas Ferreres et al. 2018b, 13.)

Example

As an example, the same message will be used as in the previous section:

I AM WRITING THE THESIS

Next, the message will be rewritten in a table of five rows and four columns (5 x 4) in this way, which is shown in table 1.

I	A	M	W
R	I	T	I
N	G	T	H
E	T	H	E
S	I	S	X

Table 1. Message *I am writing the thesis shown* in a table of 5 rows and four columns.

As the whole message does not fill all the cells of the table, a character is added to complete it. In this case, it has been used the letter X but could be whichever.

Now, to cipher the message, this is going to be written by columns instead of following the rows. The ciphered message will be the next one:

I RN ESAIGTI MTT HSWIHEX

In this method, the key would be the dimension of the table in order to know how it has been encrypted. However, it can be also used as a key to encrypt. For example, the key could be the word TEAM. Each letter of the word identifies a column (if we use the columns to make the transpositions). To encrypt, we write the word changing the positions of the letters. In this way, the order of the columns has been changed and the encrypted message is written following the order of the columns. This is shown clearly in figure 3.

Example

Message = I AM WRITING THE THESIS

Key = TEAM → EAMT

T	**E**	**A**	**M**
I	A	M	W
R	I	T	I
N	G	T	H
E	T	H	E
S	I	S	X

A	**E**	**M**	**T**
M	A	W	I
T	I	I	R
T	G	H	N
H	T	E	E
S	I	X	S

Figure 3. Message *I am writing the thesis* written by columns before and after changing the order of the columns.

Ciphered message = MTTHS AIGTI WIHEX IRNES

3.2 Substitution methods

These methods encrypt the information by changing a single character or group of characters from another one or other ones. They are similar to the transposition techniques. The

difference is that this kind of methods, the characters are replaced by other different characters instead of changing only their positions in the original information or message. (González-Tablas Ferreres et al. 2018b, 17.) For example, if we have the word Goal and we use a substitution method, the ciphered word might be this: Rtgh. As can be seen, a transposition of the characters has not been made and these have been changed.

The substitution methods can be divided in two different types: monoalphabetic and polyalphabetic.

3.2.1 Monoalphabetic substitution

The monoalphabetic substitution consists of replacing each of the characters of the message that will be encrypted by one or more from the alphabet used to cipher the information. When each character is changed by another one, the methods are monographic. On the other hand, when a letter is replaced by two or more they are called polygraphic. (González-Tablas Ferreres et al. 2018b, 17.)

One of the most well-known examples of monographic techniques is the Caesar cipher, which has been mentioned previously. It consists in applying an offset of 3 letters to the whole alphabet. Using this method, the letter A becomes D in the ciphered message. This is showed clearly in the next image (figure 4).

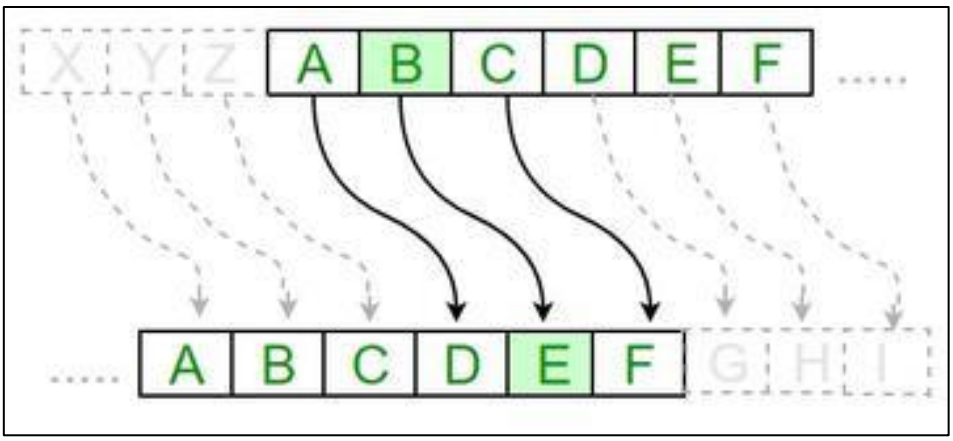

Figure 4. Diagram of Caesar cipher (Hebergement Webs)

Most of these ciphers can be defined with this equation:

$$E(m_i) = (am_i + b) mod\ n$$

a: decimation constant

b: displacement constant

m_i: letter (represented by a number) to cipher

n: number of letters in the alphabet

The operation mod which appears in the equation is the remainder of the division of two numbers, which is called modulo operation. For this expression, mod will be applied to the operation am_i+b divided by n.

For the case of Caesar cipher, the equation will be this:

$$E(m_i) = (m_i + 3) mod\ n$$

The value of a in this cipher method is 1.

3.2.2 Polialphabetic substitution

The polyalphabetic substitution methods use different alphabets to encrypt the information, not only one as the monoalphabetic ones. In this way, a letter from the original message is not always replaced by the same one in the ciphered message. The substitution applied to each letter of the message changes depending on the position in the original message. (González-Tablas Ferreres et al. 2018b, 27.)

One of the most known methods of polyalphabetic substitution is the Vigenère Cipher. For encrypting the information with this method, it is used a key composed of different symbols of the alphabet: $K = \{k_0, k_1, k_2, \ldots, k_{d-1}\}$, where d is the length of the key. To define the process of this method, the next expression can be used: (González-Tablas Ferreres et al. 2018b, 29.)

$$E_k(m_i) = (m_i + k_{i\ mod\ n})\ mod\ n$$

where m_i is the i-th symbol of the original message and n the length of the input alphabet.

Example

To explain better how this method works, an example of use will be shown. The input alphabet will be the English one, composed of 26 letters (n = 26), the message which is going to be encrypted is MORE and the key used will be SUN ($k_1 = 18$, $k_2 = 20$, $k_3 = 13$), so the value of d will be 3. The process of encryption will take place in this way:

M: E (M) = 12 + 18 mod 26 = 4 → E

O: E (O) = 14 + 20 mod 26 = 8 → I

R: E (R) = 17 + 13 mod 26 = 4 → E

E: E (E) = 4 + 18 mod 26 = 22 → W

After applying the method to the word MORE, the cipher will give as a result EIEW.In order to make easier these operations, it is used the Vigenère square. There is an example of it in the next picture, figure 5.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 5. Vigenère square (Hammond 2015)

The rows represent the letters that compose the word or the message which is going to be encrypted. On the other hand, the columns are used for the key selected. The process in this case is simple. The whole word has to be iterated character by character with the letter of the key which is related to every symbol of the word or the message. For encrypting the word MORE with the key SUN the process is as follows:

Letter M, k_1 = S: Row S, Column M → E

Letter O, k_1 = U: Row U, Column O → I

Letter R, k_1 = N: Row N, Column R → E

Letter E, k_1 = S: Row S, Column E → W

After finishing the process, the result is the same as following the mathematical expression.

4 Symmetric cryptography

4.1 Definition

Symmetric cryptography consists in encrypting the information to be sent using a secret key which is only known by the parts involved in the communication: sender and receiver. Nobody except these two parts knows this information as the key is used for encrypting and decrypting. In this way, the receiver can read the data sent by the sender. In the next image (figure 6), the process followed by these methods is shown. (González-Tablas Ferreres 2018c, 9.)

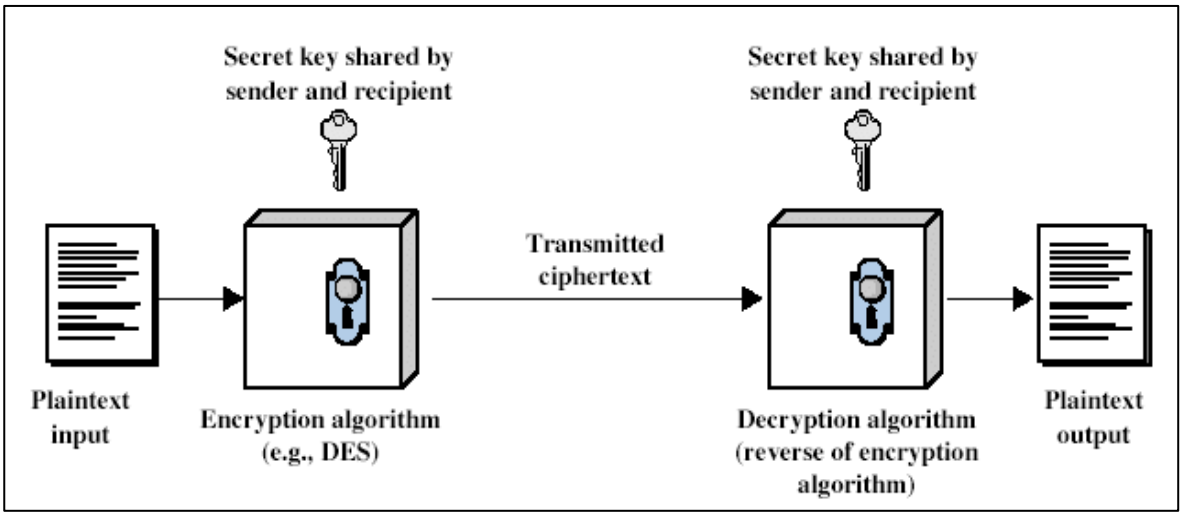

Figure 6. Diagram of symmetric algorithms functioning (Stallings et al. 2008)

In the picture above, the process followed is shown clearly. The data (in this case is plain text) is encrypted using an algorithm like DES. This algorithm, for making the encryption process, uses a secret key that, as it has been mentioned before, is shared only by sender and recipient. Once the algorithm has finished its task, the ciphered data is sent. When it gets the recipient, the decryption algorithm takes place. That means making the reverse process of the encryption algorithm using the key. After this, the information is readable by the receiver. (González-Tablas Ferreres 2018c, 4.)

About the security of this kind of algorithms, it is mainly based on the key. That means, if the key is sent to another entity external to the communication, this will be able to read the data sent. This the main problem of these methods. Anyway, this will be described in more detail later.

4.2 Block ciphers

Inside the symmetric algorithms, we have two types. One of these is the block ciphers. These kinds of methods take the plain text or the data to be encrypted in blocks of the same size. For each block of data processed, it is produced another one of equal size but with the content ciphered. To sum up, the block ciphers divide the information which will be encrypted in blocks of the same size and the cipher is applied to each one of these blocks. The size of these blocks is typically 64, 128 or 256 bits. (Stallings & Brown 2012a, 64.)

As these algorithms work with fixed-size blocks, sometimes when the plain text is divided into different parts, the size of each one of them is not equal to the size of the block. For example, if we have a total size of information of 167 bits and the size of the block is 64, the result will be 2 blocks of 64 bits and one of 39 bits. To solve that, it is applied padding to the block that does not have the same size as the established size of the block. The way this padding is made depends on the encryption algorithm used. For instance, in DES the block with bits missing is filled with 0 until getting a 64 bits block.

4.2.1 Modes of operation

For each block to be ciphered, a different mode of operation is applied depending on the application in which the encryption method will be used. In this way, each mode of operation will achieve one objective or another. For example, some of these modes are focused on encryption, whereas others try to provide data integrity. There are several modes of operation modes for the block cipher. The following are the most known modes of operation.

Electronic Codebook

Once the plain text is divided into blocks of the same size, each one of these blocks is ciphered independently from the others. As a result of this mode, it is obtained a sequence of ciphered blocks which has been generated using the same key for the ciphering to each one of them. To make the deciphering, it is applied the inverse process to each one of the blocks. In this way, the ciphering and deciphering can be described mathematically with these expressions: (Alcover Garau 2008b, 1.)

$$c_j = E_k(m_j)$$

where c_j represents each one of the ciphered blocks, E_k the algorithm used and m_j each one of the blocks in which has been divided the whole plain text which will be encrypted.

$$m = D_k(c_j)$$

where D_k represents the deciphering algorithm.

The ECB operation mode has some weaknesses. The number of ciphered text blocks is the same as the number of non-ciphered blocks. In this way, it is easier to detect patterns and figure out the original content of the data. In addition, a possible attacker can replace some blocks ciphered with the same key. Therefore, the data integrity is violated making the changes in the plain text possible and also making that the receiver of the information does not notice that. (Alcover Garau 2008b, 2.)

Cipherblock Chaining Mode (CBC)

The input of this method is the result of making the XOR operation (bitwise operation which gives 1 as a result when the two bits involved in the operation are different and 0 when they are the same) between the block which is going to be ciphered and the previous ciphered block. As in the mode explained previously, the same key is used for each block. To decipher, the deciphering method is applied for each block. The result of this operation and the previous ciphered block are the input of an XOR operation which produces the original block of plain text. (Stallings & Brown 2012a, 642.)

For the first block, the way of operating is a little bit different as there are no previous ciphered blocks. In this case, it is used an initialization array, which will be identified as IV. Therefore, the first XOR operation for the cipher is made between the IV array and the first block of plain text; and the second XOR for the deciphering is made between IV and the output of the deciphering algorithm. The process of this operation mode is shown in the next diagram (figure 7). (Stallings & Brown 2012a, 643.)

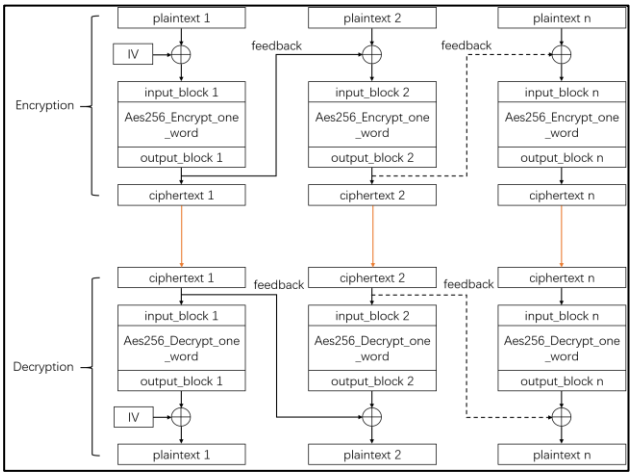

Figure 7. Diagram of the CBC operation mode (XILINX)

The initialization array IV needs to be known by each part of the communication: sender and receiver. This is mandatory as IV is used for both ciphering and deciphering. As this element is key in communication, it needs to be protected to make the process as secure as possible. Therefore, this has to be treated as a secret key for the encryption algorithms as it is essential. For that purpose, the initialization array is sent using ECB encryption. (Stallings & Brown 2012a, 643.)

This operation mode improves the ECB mode as now the cipher of a block depends on the previous one and the key used. In this way, equal blocks are not related to the same original plain text block as these have been ciphered in a different way.

Cipher Feedback (CFB)

This operation mode transforms any block cipher into a stream cipher. The data is encrypted in smaller units than the size of the block. (William Stallings, Lawrie Brown 2012, 644.) For example, if the block size is 64 bits the unit that will be used can be 8 bits. This size has to be always lower than the size of the block. The process followed by this mode is the next one.

The first thing that needs to be taken into account is encryption. Its input is a shift register which is, in the beginning, an initialization vector (IV) with a size of b bits. For this case, it will be used 64 bits. This process will give as output an encrypted register, from which will be taken the most significant s bits (the leftmost ones) and the others will be discarded. For this case, it can be established 8 as s. These 8 bits are XORed with the unit of plain text which is used in the current stage of the process and has a size of 8 bits too. As a result, it is obtained the ciphered text for this unit of plain text. The ciphered text is transmitted and the process continues as it is shown in the next picture (figure 8). (Stallings & Brown 2012a, 645.)

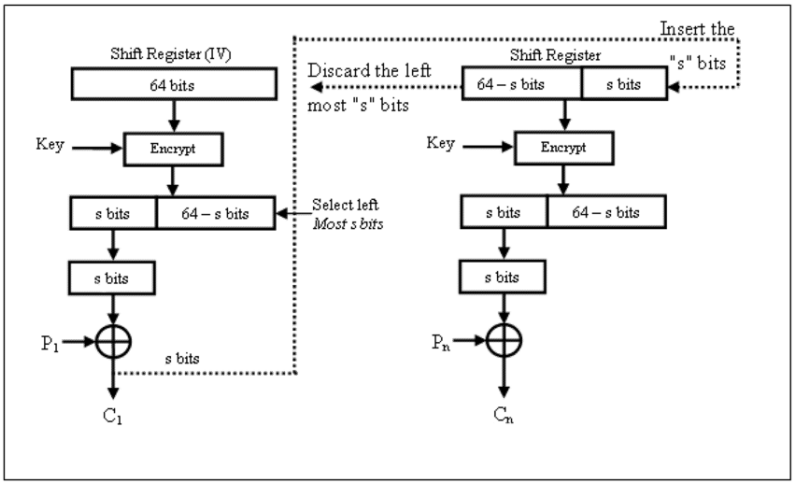

Figure 8. Diagram of the CFB mode (Yas Alsultanny 2007)

Next, the same steps are made again for the following unit of text. The input for the encryption is, as before, the shift register. However, the content will not be the same. The data of the register is shifted s bits (8 for this example). In this way, the most significant bits in the register (the first ones starting from the left) will be the bits discarded before when the encryption was applied, and the less significant bits are the ones calculated in the previous XOR operation. (Stallings & Brown 2012a.)

After that, the process continues as it has been described before. The content of the shift register is encrypted, the first 8 bits are selected and the 56 bits remaining are discarded. These 8 bits and the second unit of plain text will be subjected to an XOR operation, giving, as a result, the second text ciphered unit. The method will be repeated until finishing the encryption of the whole message.

About the decryption, the process followed is quite similar. The XOR operation is made with the ciphered text instead of plain text. On the other hand, for reading the original message, a decryption algorithm is not applied as in the other operation modes. Encryption is also used in this process.

Counter mode (CTR)

For this method, it is used a counter. The counter could be any value that is not repeated in a long time and if it is possible, used only once. This value is commonly known as nonce, which means that the number is used only for a single use, like for the encryption in a single communication. It needs to be generated randomly to ensure security. (Stallings & Brown 2012a, 645.)

This nonce is normally concatenated with another number which is incremented one by one every time that a new block of plain text is ciphered. In this way, these two values combined become the input for the CTR process. In a real case, if the size of the block used for the encryption is 128 bits, the first 64 bits correspond to the nonce and the other 64 remaining to the number which is incremented. (Stallings & Brown 2012a, 645.)

This counter is the input for encryption. Once this task is made, the encrypted counter is XORed with the block of plain text. When this operation is made, the ciphered block is obtained. (Stallings & Brown 2012a, 645.)

For decryption, the process is quite similar. In the XOR operation, it is used the ciphered block instead of the plain text one. The same values for the counter are used for the decryption of each ciphered block. The process is shown in the following image (figure 9). (Stallings & Brown 2012a, 645.)

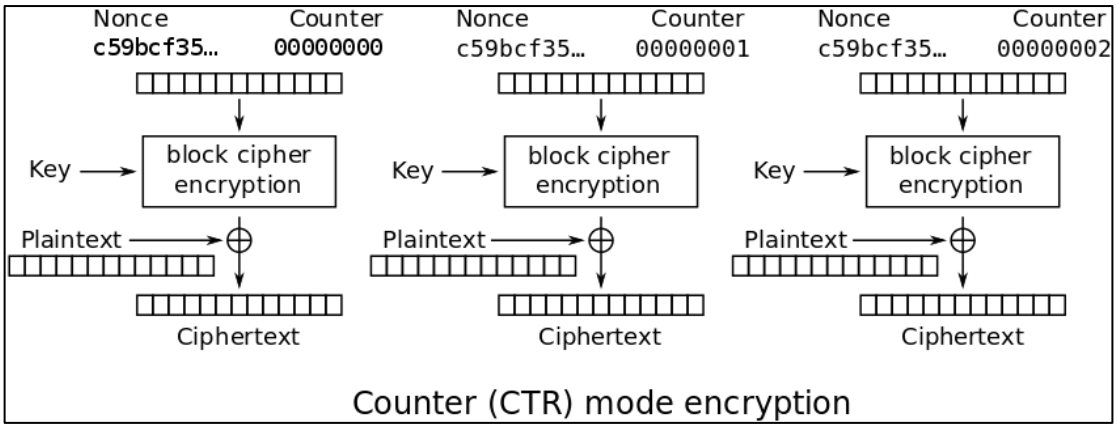

Figure 9. Diagram of the CTR mode (Nakov 2018)

As each block can be encrypted independently because the encryption process does not need the previous ciphered block, this method can be parallelized. This means that while one ciphering is taking place, others can be too. Therefore, the process might take less time in being finished than other operation modes. (Stallings & Brown 2012a, 645.)

4.2.2 Feistel cipher structure

Block cipher algorithms use a similar scheme to make the encryption of the data, like, for example, the DES algorithm. This structure was defined by the cryptograph Horst Feistel, who was working in the IBM company. The structure works as follows.

The input received is a plain text block of w bits which is divided in two, producing two halves of w/2 bits, left and right half; and also a key K. These two halves are processed during a determined number of rounds, which will be n. In each round, referred to as i, there are two input elements, L_{i-1} and R_{i-1}, which will be the left and the right half of the original input block respectively obtained in the previous round. In addition, in each round is used a subkey K_i, generated from the input key K. The subkeys Ki are generated from a subkey algorithm generator using as input from it the key K. (Stallings & Brown 2012a, 627.)

Each round has the same structure following the same steps each time. First, the left half of the block L_i is subjected to a substitution. The substitution is made using an F function called the round function (which depends on the K_i subkey of the round) applied to the right half R_i. Then, an XOR operation is applied to the output of the F function and the L half of the data processed. After this substitution, a permutation takes place, which consists in the exchange of the two halves. (Stallings & Brown 2012a, 627.)

This process is the same for each round, only changing the subkey. For deciphering, the steps are the same. The only thing that varies a little bit is the order of the subkeys. If we have n rounds, the first subkey used is the n round subkey, the second one the n-1 round subkey and so on. In this way, the order of the subkeys used is the opposite of the one used in the ciphering. In the next picture (figure 10), both processes, encryption and decryption are shown.

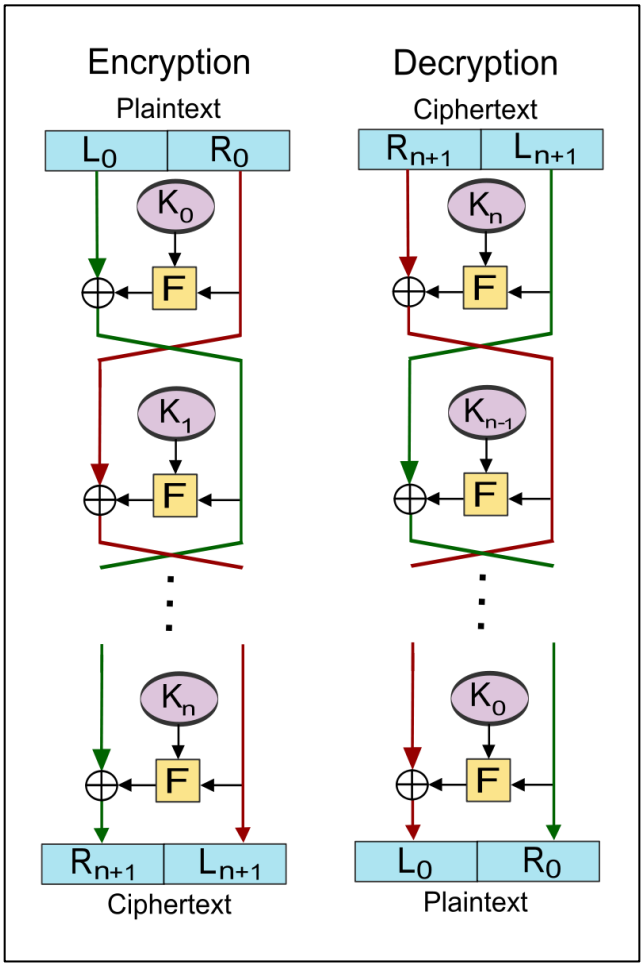

Figure 10. Diagram of Feistel cipher structure (Amiriki 2011)

The Feistel structure is the typical one used in block cipher algorithms and it shows the main factors that define this kind of encryption methods. Therefore, as a recap, the main aspects were mentioned but not defined at all. These are the main issues that have to be taken into account when block cipher methods are defined:

- **Block size**. It refers to the size, in bits, of the block which is used as the input for the encryption. With larger blocks the complexity increases. Normally, the block size used in current algorithms is 128 bits. (Stallings & Brown 2012a, 627.)

- **Key size**. The same as block size but related to the key used in these algorithms to make the encryption. Like the blocks, the typical size used for the key is 128 bits. Larger keys mean that complexity increases. Nevertheless, the encryption and decryption processes are slower. (Stallings & Brown 2012a, 627.)
- **Number of rounds.** The rounds are one of the most important factors of these algorithms. Applying a single round in the encryption does not provide much security. However, many of them increase it a lot. There are normally 16 rounds in these methods. (Stallings & Brown 2012a, 627.)
- **Subkey generator algorithm.** As has been described previously, for each round a subkey is generated from the key received at the beginning as input for the method. Complex algorithms increase the security of the methods for cryptoanalysis. (Stallings & Brown 2012a, 627.)
- **Round function.** This is the function applied to the right half of the block used at the beginning of the process of each round. As in the case of the subkey generator algorithms, the bigger complexity makes the method more secure. (Stallings & Brown 2012a, 627.)

4.2.3 DES (Data Encryption Standard)

The Data Encryption Standard Algorithm (DES) was developed by IBM as a request from the National Bureau of Standards (NBS) of the USA, currently known as the National Institute of Standards (NIST); which published a tender to create a cipher algorithm to protect the data. This algorithm was chosen as a standard to cipher confidential information. (Sánchez Arriazu 1999a, 1.)

DES algorithm is a block cipher method. It uses a block size of 64 bits. Currently, this algorithm is not considered as secure as it has been broken. The reason is that the key size is only 56 bits, which means that current computers can test all the possibilities of the key and figure out which one was used during the encryption process. The key is not exactly 64 as the block size because 8 bits of the key are used for parity. This means that these 8 bits are used to check data integrity to see if the data has been modified. (Sánchez Arriazu 1999a, 1.)

The method can be divided into two main phases: key processing and data block processing. The first phase is responsible for obtaining the different keys that will be used in the processing of the data block. The second one protects the data sent by applying different transformations to the data which will be encrypted. (Sánchez Arriazu 1999a, 5.)

Key processing

First of all, a 64 bits key is introduced as input for the key processing. After that, a permutation is applied to the key. That means the bits will change their original position once this transformation is made. The permutations are made using the following table (figure 11), which determines the new positions. (Sánchez Arriazu 1999a, 5.)

$PC-1$							
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

Figure 11. Permuted Choice 1 table (Agarwal & Marsh 2015a)

Following the table, the first bit of the key after applying the permutation will be the number 57, the second one the number 49 and so on. Once all the positions have been changed, the less significant bit (the rightmost one) of each byte that composes the whole key is removed and the key is reduced to 56 bits. The other bits are used for checking data integrity as has been said previously. (Sánchez Arriazu 1999a 5.)

Next, the new key is divided into two halves, left and right. The left part has the most significant bits of the key, which are the leftmost ones. These two halves are the starting point to calculate the 16 subkeys that will be used later in the encryption of the data block.

Following the process, each half will be shifted some bits to the left, depending on the round. The rounds will be 16 in order to obtain the 16 subkeys. For rounds 1, 2, 9 and 16, each half will be shifted one bit. On the other hand, in the other rounds, the two parts of the key will be shifted two bits. If all shifted bits are added together, a total of 28 bits will be obtained, which is the number of bits of each half. Therefore, after making the 16 rounds, the final key will be the same as the initial before as all the bits have been shifted. (Sánchez Arriazu 1999a, 5.)

Once the shift is applied to each half, a new permutation, called Permuted Choice 2 (PC-2), is made. This permutation takes the two halves, left and right, concatenated, and changes the position of the bits as in PC-1. This time, the result is a key composed of 48 bits as the other 8 bits are removed. The permutation is made following the next table (figure 12). (Sánchez Arriazu 1999a, 6.)

PC – 2							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Figure 12. Permuted Choice 2 table (Divya Agarwal & Katie Marsh 2015b)

Once this permutation is applied, the subkey for the encryption is generated. This process will be repeated 15 times more in order to obtain the required 16 subkeys.

Data block processing

After all the subkeys have been generated, the encryption process can start. The first step is to get a block of 64 bits, as it is the size that DES works with. This step is necessary as not all the blocks will have the size required. Once the 64 bits are reached, the encryption process can start. (Sánchez Arriazu 1999a, 6.)

First of all, the block is subjected to an initial permutation, similar to the ones described previously for the subkey generating process. Next, the block is divided into two parts, left (L) which contains the most significant bits, and right (R); each one of 32 bits. After that, the expansion process takes place for the right part. This means that the R block will be expanded to 48 bits from the initial 32 bits. For this task, some bits will be repeated in order to reach the required amount. This expansion is made following the next table (figure 13). (Sánchez Arriazu 1999a, 6.)

Expansion (E)

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

Figure 13. Expansion table (Sánchez Arriazu 1999b)

Therefore, the R block will be subjected to a permutation of some bits and some of them will appear more than one time after applying this transformation. In the table is shown that the first bit will be the number 32, the second one will be the one which was the first before and so on. In addition, bit number 1 is repeated and will appear again in the new right block as the last one. (Sánchez Arriazu 1999a, 6.)

Next, once the right half block is expanded, an XOR operation will take place between the result of the expanded block and the subkey used for this round. As a result, it will be obtained a new block of 48 bits, which will be divided into six smaller blocks, each one of six bits. (Sánchez Arriazu 1999a, 7.)

After this operation, each one of the eight blocks will be subjected to a substitution operation. The substitution of the bits is defined in the S boxes. The S boxes are composed of eight matrixes in total, each one applied to each one of the eight blocks of six bits. These matrixes are made up of four rows and 15 columns. To determine the row, the first bit (b_0) and the last one (b_5) are used. On the other hand, the remaining bits (b_1, b_2, b_3, b_4) are consulted. (Sánchez Arriazu 1999a, 7.)

In order to understand this better, an example will be given. In this case, the input for the first S box is 110111. The bits b_0 and b_5 are used to know the row, which are 1 and 1 for this example. The number 11 is three in decimal, so the row, in this case, is the third one. To know the columns, the remaining bits (b_1, b_2, b_3, b_4) are 1011. This number is 11 in decimal, so the column is the 11^{th} one. Once these two numbers are known, the first S box (that can be seen on the next page in figure 14) is checked and it says that the number for row three and column number 11 is 14. Therefore, a 4-bit number is obtained as the highest one which appears in the S boxes is the number 15, which is the highest

number that can be obtained using four bits. These S boxes will be applied to each one of the blocks of 6 bits. (Sánchez Arriazu 1999a, 7.)

Fila	Columna																S-Caja
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Figure 14. S boxes DES algorithm (Sánchez Arriazu 1999b)

Once the eight S boxes have been applied to each one of the blocks, all of them are concatenated to form one 32-bit block. Then this block is subjected to a new permutation which changes the positions of the bits in the block as has been seen in previous phases of the algorithm. Next, another XOR operation is made, in this case between the result of the previous permutation and the left block L.

The result of this operation will be the new R right block and block L will be, at this point, the previous R block. In other words, once the XOR operation is made, R and L exchange their positions. (Sánchez Arriazu 1999a, 8.)

This whole process (key expansion, S boxes, permutation and XOR operations) is repeated for each one of the 16 subkeys generated previously and it is only applied to the R right half block of the initial input of the algorithm. Once the rounds have finished, another permutation (which is the inverse of the initial Permutation IP) is applied to the concatenation of the right block (R(16)) and the left block (L(16)) obtained in the last round. Finally,

the data obtained after the IP^{-1} permutation is the ciphered block. To sum up, the process of a single round is shown in the next picture (figure 15). (Sánchez Arriazu 1999a, 8.)

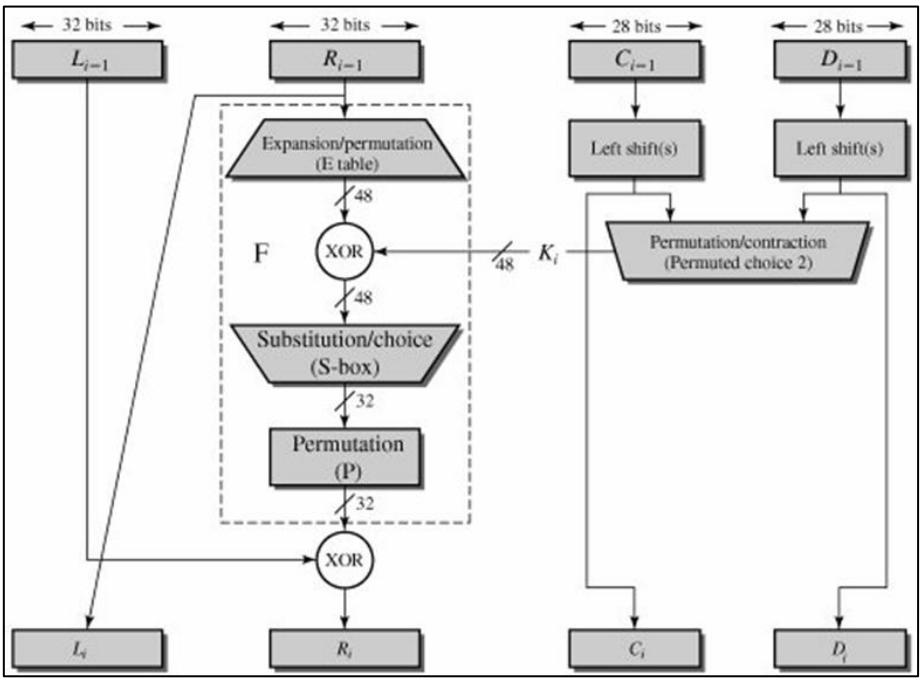

Figure 15. Diagram of a DES round and the subkey generation. (Stallings & Brown 2012b)

4.2.4 Triple Data Encryption Standard (TDES)

Triple Data Encryption Standard (TDES) is a variation of the DES algorithm. It mainly consists in applying the DES algorithm three times. It was created because of the vulnerabilities found in DES, which are mainly related to key size. As it only uses a 64-bit block, from which only 56 bits are used, this algorithm is weak against brute force attacks as has been explained previously. (Stallings & Brown 2012a, 630.)

With this new implementation of DES, the security increases as now the key size is higher. Therefore, brute force attacks are weaker against this algorithm. Looking at the key size, there are two kinds of TDES: TDES with two keys (key size of 112 bits) and TDES with three keys (key size of 168 bits). (Stallings & Brown 2012a, 43.)

Note that as not all the bits from the key are used to encrypt, the key size which is used at the end is lower because the last bit of each byte that forms the whole key is used for parity. Therefore, from the first implementation of TDES, 16 bits are removed (64 bits/key x 2 keys = 128, 128/8 = 16, 128 -16 = 112 bits) and from the second one 24 bits are removed

(64 bits/key x 3 keys = 192, 192/8 = 24, 192 - 24 = 168 bits). (Stallings & Brown 2012a, 44.)

The process of encryption in TDES algorithm is quite simple as it is based on the DES algorithm. The algorithm receives a 64-bit block of data as input and three iterations of DES are executed following this structure: (Karthik & Muruganandam A. 2014.)

- Encryption with the first key
- Decryption with the second key
- Encryption with the third key

For the 112 bit-key, the process is the same (it can be seen in figure 16). However, the key used in the encryption stages varies a bit as in the encryption phases the same key is used. On the other hand, the decryption process for both key sizes is the same. The only difference with the encryption is that the sequence is as follows: (Karthik & Muruganandam 2014.)

- Decryption with the third key
- Encryption with the second key
- Decryption with the first key

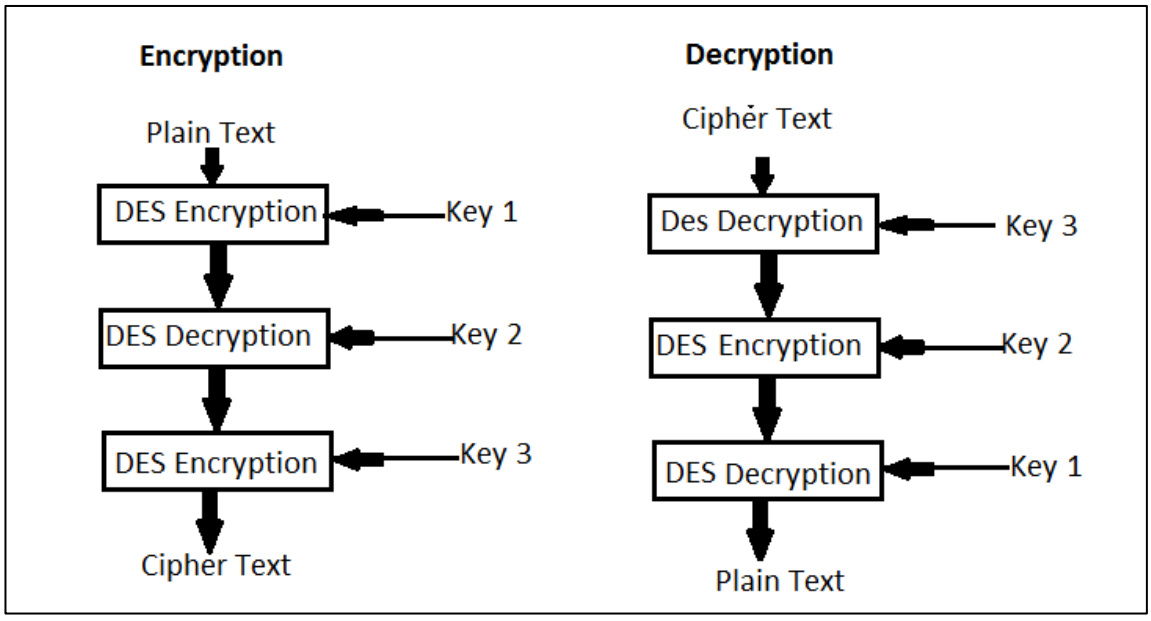

Figure 16. Diagram of TDES encryption and decryption (Nevon Projects)

4.2.5 Advanced Encryption Standard (AES)

Advanced Encryption Standard is one of the most used and secure encryption algorithms nowadays. Its development started as the search for a new successor to the DES algo-

rithm, which showed some drawbacks and weaknesses that made them insecure. The search was started by the NIST (National Institute of Standards and Technology). The institution organized a contest to find the best algorithm. The chosen one was the Rijndael algorithm, which became the new AES encryption standard in 2001. (Boxcryptor.)

AES is a block cipher as DES. It uses 128-bit blocks of data input for the encryption and larger keys of three different sizes: 128, 192 and 256 bits. The algorithm does not follow the Feistel structure, as it is a substitution-permutation network, being these operations the main ones in the encryption process. (Stallings & Brown 2012a, 631.)

One of the main features of his algorithm that differentiates from DES is the state array, which is represented by a square matrix of bytes. The state array is modified in each phase of the algorithm and stores the result of the different operations that are made during the process. Once the algorithm has finished at the final stage, the result is copied to an output matrix that contains the ciphered block. Therefore, as the block size is 128 bits, the state array will be a 4 x 4 matrix, (four rows and four columns) with 16 bytes in total. Here it is shown the state array, representing with the letter B each one of the bytes. (Stallings & Brown 2012, 631.)

$$\begin{pmatrix} B_{15} & B_{11} & B_7 & B_3 \\ B_{14} & B_{10} & B_6 & B_2 \\ B_{13} & B_9 & B_5 & B_1 \\ B_{12} & B_8 & B_4 & B_0 \end{pmatrix}$$

As mentioned above, multiple operations are applied to the state array. These operations are one of the main features of this algorithm as they define how the data is encrypted. There are four main operations, which will be described later in more detail: (Stallings & Brown 2012a, 631.)

- **AddRoundKey**. An XOR operation of the block with the key of the round (Stallings & Brown 2012a, 631).
- **ByteSub.** It makes a byte-by-byte substitution of the block using a table (Stallings & Brown 2012a, 631).
- **ShiftRow.** A row by row permutation (Stallings & Brown 2012a, 631).
- **MixColumns.** A transformation of the four bytes of each column by multiplying them with a fixed matrix (Stallings & Brown 2012a, 631).

The encryption process of this algorithm starts with an Add Round Key operation. After that, the four operations are executed following this order: Substitute bytes-Shift rows-Mix columns-Add round key. These are made during a defined number of rounds, depending on the size of the key. For a 128-bit key 10 rounds are made, for 192 bits 12 and for 256

bits 14 rounds. The sequence is the same in all the rounds except for the last round. In this, the sequence made is the next one: Substitute bytes-Shift rows-Add round key. (González-Tablas Ferreres et al. 2018c.) After finishing all the rounds the ciphered block of 128 bits is obtained. To sum up, the process followed by the algorithm is shown in the next image (figure 17).

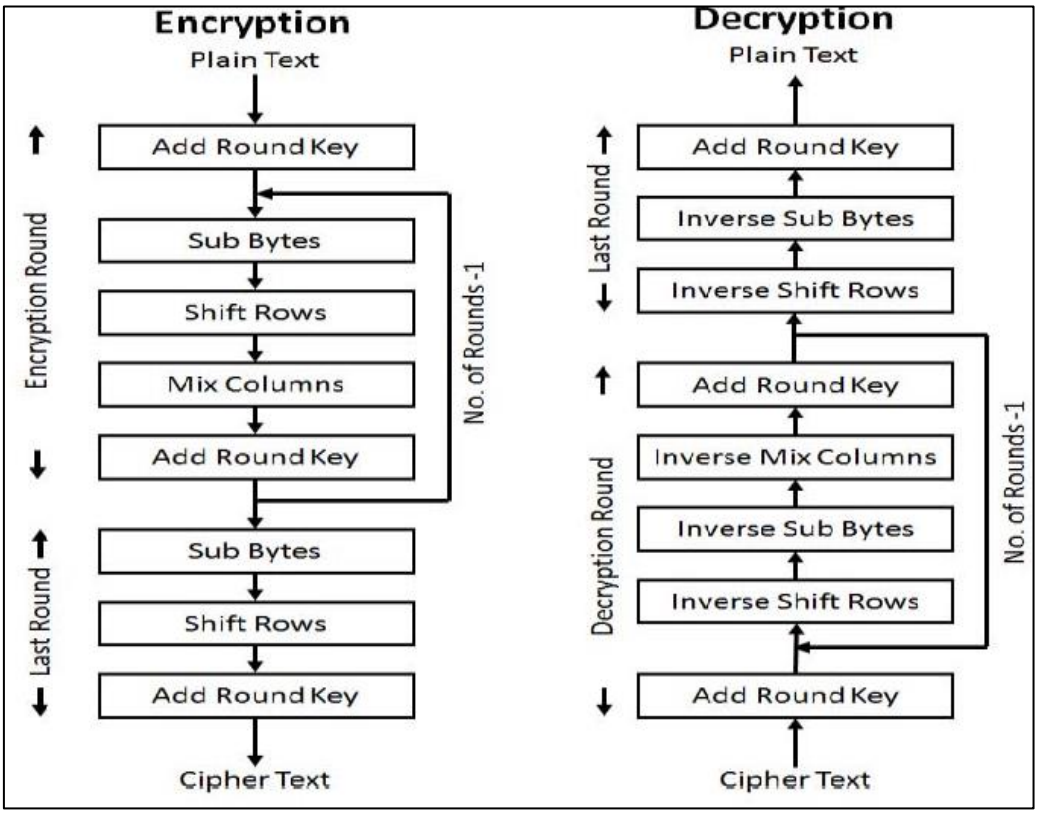

Figure 17. AES encryption and decryption diagram (Moustafa Mamdouh et al. 2017)

As shown above, the decryption process of a ciphered block is quite similar to encryption. The operations applied in this case are the inverse ones and are made in the opposite order. For example, the add round key operation is the same. However, the Shift rows one has the Inverse Shift rows operation in the decryption phase.

Substitute byte transformation

Substitute byte is a simple transformation of each byte of the state array. The bytes that will replace the ones currently in the state array are defined in the S-Box table shown below. The S-Box is a square matrix of 16 rows and columns containing the 256 possible permutation values of the bytes. To determine the values corresponding to each byte in the state matrix, the rightmost 4 bits are used to indicate the column and the leftmost 4 bits are used to indicate the row. For example, the byte 8F represented in hexadecimal

will be substituted by 73. This substitution will be applied to all the bytes of the state array. An example of S-Box is shown in figure 18. (Stallings & Brown 2012a, 634.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 18. S-box for AES algorithm (Vazhayil 2015)

The inverse substitute byte transformation does the same operation. It looks in another table and gives the inverse value. In the previous example, the value 8F was substituted by 73. For the inverse substitution, the input value 73 will give as output 8F in order to recover the previous value when the decryption takes place. (Stallings & Brown 2012a, 643.)

Shift row transformation

The shift row transformation consists in shifting to the left a given number of bytes depending on the row of the state matrix. All rows, except the first one, are subjected to byte shifts, so this one is not altered. For the second row, a left shift of one byte is applied. For the third row, the shift is 2 bytes. Finally, for the fourth row, the number of bytes shifted is 4. To explain this operation better here there is an example. (Stallings & Brown 2012a, 634-636.)

$$\begin{pmatrix} 56 & 8F & 12 & 45 \\ DA & 1C & 4E & 9A \\ 0F & FF & BD & 09 \\ 7C & 80 & 9D & 1C \end{pmatrix} \rightarrow \begin{pmatrix} 56 & 8F & 12 & 45 \\ 1C & 4E & 9A & DA \\ BD & 09 & 0F & FF \\ 1C & 7C & 80 & 9D \end{pmatrix}$$

The inverse right shift transformation executes the same operation. In this case, the shift applied to the last three rows is made to the right. As a result, the state array will be obtained before applying the normal shift transformation. (Stallings & Brown 2012a, 636.)

Add round key transformation

In this operation, all the bits from the state array are subjected to an XOR operation with the bits of the round key. For the first round, the operation will be made with the input of the algorithm and the initial key. For the next rounds, the XOR will be applied to the round subkey and the output of the mix columns transformation. Finally, for the last round, the operation is made between the subkey of the last round and the output of the shift rows transformation. Below there is an example of the add round transformation. (Stallings & Brown 2012a, 637.)

$$\begin{pmatrix} 47 & 40 & A3 & 4C \\ 37 & D4 & 70 & 9F \\ 94 & E4 & 3A & 42 \\ ED & A5 & A6 & BC \end{pmatrix} xor \begin{pmatrix} AC & 19 & 28 & 57 \\ 77 & FA & D1 & 5C \\ 66 & DC & 29 & 00 \\ ED & A5 & A6 & BC \end{pmatrix} = \begin{pmatrix} EB & 59 & 8B & 1B \\ 40 & 2E & A1 & C3 \\ F2 & 38 & 13 & 42 \\ 1E & 84 & E7 & D2 \end{pmatrix}$$

As the bytes are represented with hexadecimal notation is difficult to see how the XOR is applied as it is a bitwise operation. To see it better, the XOR between the bytes 40 (0100 0000 in binary) and 19 (0001 1001) will be described. When an XOR operation is performed between two bits, when these have the same value (1 and 1 or 0 and 0), the result is 0. On the other hand, when the values are different (1 and 0 or 0 and 1) the result is 1. Therefore, the result of the operation 01000000 XOR 00011001 is 01011001, which in hexadecimal is 59.

For the decryption process, the same round key transformation is applied. In this case, there is not any inverse operation. The reason is that the XOR operation is its inverse itself. If the operation is applied to 19, the byte of the round key, and 59, the byte obtained as a result of the previous XOR, the result will be 40, the value from the state array that had been transformed previously. (Stallings & Brown 2012a, 637.)

Mix columns transformation

The mix columns transformation is performed in each column of the state array. Each byte of the column is transformed into a new value by multiplying the column by a fixed matrix. Here there is an example.

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix}$$

The values represented by S and S' are the ones from column c of the state array. The difference between S and S' is that S is the value before making the multiplication and S' the value after operating. The multiplication is similar to the normal matrix multiplication. The columns from the first matrix and the rows of the second matrix need to be the same value. In this case, the first matrix has four columns and the second one has four rows, so multiplication is possible. (González-Tablas Ferreres et al. 2018c.)

To calculate each one of the values, the operation starts multiplying the first number of the first row of the matrix at the left with the first number of the first columns of the matrix at the right. The operation follows making the same for the next values and adding the result to the previous multiplications. For calculating $S'_{0,c}$, the sequence will be as follows:

$$S'_{0,C} = 02 \cdot S_{0,C} + 03 \cdot S_{1,C} + 01 \cdot S_{2,C} + 01 \cdot S_{3,C}$$

However, in the case of this Mix columns function, the multiplications and additions are not done in this way as they are applied to numbers with different properties that have different definitions for these operations. Therefore, the add operation is substituted by an XOR. (González-Tablas Ferreres et al. 2018c.)There are some other variations in the multiplication. Nevertheless, this will not be explained in more detail as it is outside the scope of this thesis.

Key Expansion

The AES key expansion will create keys for each of the rounds executed in the algorithm from the key introduced by the user. The bytes of the key are grouped in words of 32 bits, which are four bytes. These values will be copied to a linear array called W, whose length will be determined by the following formula: (González-Tablas Ferreres et al. 2018c.)

$$W = Nb * (Nr + 1)$$

Where Nb represents the number of words of the block which is going to be ciphered; and Nr is equal to the number of rounds executed in AES, which depends on the size of the key. If a 128 bit-key is used for the encryption, Nb will be equal to four. Owing to this key size, the number of rounds will be 10. Therefore, with these values, the size of the W array can be calculated. (González-Tablas Ferreres et al. 2018c.)

$$W = 4 * (10 + 1) = 44$$

This will be the size of the array for a 128-bit block and a 128-bit key. Next, the 4 4-byte words from the key will be copied to the first four positions of the array (W[0-3]). After that, the remaining 40 subkeys for the round need to be calculated. For that purpose, a specific algorithm is executed, which can be seen in the following image (figure 19).

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++)    w[i] = (key[4*i], key[4*i+1],
                                       key[4*i+2],
                                       key[4*i+3]);
    for (i = 4; i < 44; i++)
    {
     temp = w[i - 1];
     if (i mod 4 = 0)    temp = SubWord (RotWord (temp))
                                ⊕ Rcon[i/4];
     w[i] = w[i-4] ⊕ temp
    }
}
```

Figure 19. Implementation of the AES key expansion algorithm (Ather Owais 2019)

As shown above, the first four positions contain the words from the input key. Once this is finished the other values are calculated. First, the value from the previous position from the W array is copied to the variable temp. Therefore, if the current position (represented in the code by the variable i) is four, the value in temp will be the content of the W in the third position (W[3]). (BrainKart.)

Next, it is checked if the current position is divisible by four, making the module operation. The modulo operation calculates the remainder of a division. Therefore, if the remainder is zero when a number is divided by four, this number is divisible by four. For example, if the number four is divided by four, the remainder will be zero and the condition will be fulfilled. However, this is not the case for five, for example, as the remainder will be one if it is divided by four. (BrainKart.)

If this condition is fulfilled, the value of temp will change. Its new value will be the result of applying the Substitute byte transformation described previously to the result of the operation RotWord, which is applied to temp. The RotWord function consists in a left shift of one byte as the ones performed in the Shift rows transformation. Next, the result of this operation is subjected to an XOR with Rcon, which is a constant value that changes every round. (BrainKart.)

4.3 Stream ciphers

The stream ciphers divide the input message or data typically in bytes. However, sometimes the division is made by bits too. This is the main difference with block ciphers, which encrypt using blocks of data.

As these types of ciphers are symmetric methods, they use a key for both encryption and decryption. However, the difference with other methods is that the key is not used in the way it is introduced to the stream cipher. First, the key is subjected to a transformation made by a pseudorandom bit generator. It produces a stream of eight bits (one byte) numbers that seem to be random. (Stallings & Brown 2012a, 46.)

This random stream of numbers, which is called keystream is subjected to an XOR operation with the content which is going to be ciphered. The operation is made byte-by-byte each time. Therefore, if the stream that is going to be ciphered is 101110110, the ciphering will be as follows. (Stallings & Brown 2012a, 46.)

The plaintext which is going to be ciphered is 10110110 and the keystream obtained in the random generator is 01101100. To make the encryption, the XOR operation is performed, giving as a result 101110110. For decryption, the process is the same. The XOR operation is made between the ciphered text and the keystream, giving as a result the plain text before the ciphering.

4.3.1 Golomb postulates

To encrypt the information using a stream cipher, the keystream generated needs to be or at least seem as random as possible. The mathematician Solomon Golomb suggested the randomness properties that a keystream needs to have to be considered secure.

- **Postulate 1.** The number of 0s in the keystreams needs to be equal to the number of 1s. As maximum, it only can exist a difference of one. For example, it is acceptable that there are seven 1s and eight 0s in a keystream. (Ramió Aguirre & Muñoz Muñoz 2015, 2.)
- **Postulate 2.** The runs of the keystream have to follow a geometric distribution. In other words, half of the runs have a length of one, one-fourth have length two, one-eighth have length three and so on. Runs are sequences of equal numbers as, for example, 1111 or 00000. (Ramió Aguirre & Muñoz Muñoz 2015, 2.)
- **Postulate 3.** The out-of-phase autocorrelation function should have the same number of hits as misses for all shifts applied to the sequence, from 1 to n-1 (n is the number of bits in the sequence). The autocorrelation function compares bit by

bit both sequences, the original one and the one when the shift is applied. This comparison is based on counting the number of hits, which are the number of equal bits in both sequences; and the number of misses, which are the number of bits that have changed in the new sequence. The autocorrelation function is defined by the following expression. (Ramió Aguirre & Muñoz Muñoz 2015, 2.)

$$\boldsymbol{AC(K) = \frac{A - F}{T}}$$

The A value represents the number of hits, the letter F the number of misses, T the number of bits of the sequence and K the bits which have been shifted. Therefore, the value of this expression needs to be constant for every value of K. (Ramió Aguirre & Muñoz Muñoz 2015, 2.)

These conditions must be fulfilled by a sequence of bits. However, they are not enough and some other properties are needed for having a sequence as random as possible. For example, another desirable property for the bit sequences is having a period as long as possible. The period is the number for which the sequence, called s, is N-periodic. The s sequence is N-periodic if $s_i = s_{i+N}$, where i is a number from 1 to length of s -1, which refers to each bit of s. (Ramió Aguirre & Muñoz Muñoz 2015.)

Suppose that the sequence s is as follows: s = 0, 1, 1, 0, 1,1, 0, 1, 1, …. It can be seen that $s_i = s_{i+3}$, as $s_1 = s_4$ (0 = 0), $s_2 = s_5$ (1 = 1), $s_3 = s_6$ (1 = 1), and so on. Therefore, this is an example of a N-period sequence, in this case, 3-periodic. (Mitra.)

4.3.2 RC4 algorithm

The RC4 algorithm was developed by Ron Invest in 1987 for the company RSA security, which is focused on encryption and encryption standards. The algorithm was kept a secret. However, some years later, in 1994, the algorithm is published in the Cyberpunk email list, making it public. (Estrella Gutiérrez 2006.)

The RC4 algorithm is a stream cipher that has a variable-length key, which varies from 1 to 256 bytes. The messages in this algorithm are ciphered applying an XOR operation between the message and the keystream which needs to be generated. The keystream is generated using two algorithms: KSA and PRGA (Stallings & Brown 2012a.)

The KSA algorithm uses the S array, which is first initialized from 0 to 255, being all its elements byte values. Then, the key used for this algorithm is copied in an array called T, whose length is 256 bytes as S. If the length of the key is lower than 256 bytes, this is repeated on the array until reaching the length of the array. Typically, the key length is 128 bits (16 bytes). (Ramió Aguirre & Muñoz Muñoz 2016a.)

Next, the permutations from S array will be performed. For that, the T array will be used to calculate the positions which have to be permuted in the S array. The output of this process will be the input for the next stage which is keystream generation, the PRGA algorithm. The permutation of S will be made following this algorithm.

```
KSA algorithm {
        j = 0;
        for i = 0 to 255 {
                j = (j + S[i] + T[i]) mod 256;
                swap (S[i], S[j]);
        }
}
```

The variable i will be a simple iterator that contains the current position from the S array. The j value will contain the position with which the current position is going to be swapped. In other words., the value in the current position of the S array (i) will be the value from the j position. (Ramió Aguirre & Muñoz Muñoz 2016a.)

This j position is calculated by applying the operation modulo 256 to the result of adding the value of j, the value from S in position i, and the value from T in position i too. The modulo 256 operation is made in order to obtain a number from 0 to 255. The reason is that as the length of the array is 256, the first and last positions are 0 and 255 respectively and the modulo operation ensures that the results will be inside that range. Once this is made for each one of the positions of the S array, the PRGA algorithm can be executed. (Ramió Aguirre & Muñoz Muñoz 2016a.)

The Pseudo-Random Generation Algorithm (PRGA) uses the S array which has been generated by KSA. PRGA is the responsible of generating the keystream which will be XORed with the message to generate the ciphered block. The keystream produced will have the same size as the message which is going to be encrypted. The process for generating it is described in the algorithm. (Ramió Aguirre & Muñoz Muñoz 2016a.)

```
PRGA algorithm {
        i = j = k = 0;
        while (k < L){
                i= (i + 1) mod 256;
```

```
                    j= (j+ S[i]) mod 256;
                    swap(S[i], S[j]);
                    t = (S[i] + S[j]) mod 256;
                    Copy value from S[t] to the final ouput;
                    k++;
            }
}
```

As shown above, first some variables are initialized to 0. These will be counters that will contain the values of the positions that will be swapped (i and j) and the values of the position from the final keystream, in order to know where the new byte has to be located once is calculated. After that, the steps to calculate the bytes of the keystream are executed while the value of k is lower than L, which is the length of the message which is going to be ciphered and the length of the keystream. (Ramió Aguirre & Muñoz Muñoz 2016a.)

The positions i and j are calculated by making similar module operations like the ones described in KSA algorithm. When the values are known, the data from the positions i and j from the S arrays are swapped. Then, the t value is calculated, which will be the position where the value from S that will be taken to generate the keystream is located. It is calculated by applying the modulo 256 operation to the result of adding the values of the positions i and j from S. Next, the value t is copied to the output (the keystream) and the k variable is incremented by one. If its value is lower than the length of the keystream (or the length of the message), then the process described is repeated until the whole keystream is generated. (Ramió Aguirre & Muñoz Muñoz 2016a.)

Once this process is finished, the message is ciphered by making an XOR operation with the keystream obtained. The process for decryption is the same, but making the XOR of the keystream with the ciphered text. (Ramió Aguirre & Muñoz Muñoz 2016a.)

5 Asymmetric cryptography

5.1 Definition

In the previous section, it has been seen that symmetric cryptography uses a single key for both encryption and decryption. In addition, this is shared with both sender and receiver involved in the communication. On the other hand, in asymmetric cryptography, a pair of keys is used. This pair of keys is generated by sender and receiver, so each one of them has two keys, one public and one private. (Ionos 2019.)

The public key of each one of the parts involved in the communication is shared in order to make possible the exchange of information. Nevertheless, this is not the case for the private key. This key is protected and it is not known by anyone else. If the private is from the sender, this is only known by him/her. The same situation happens to the private key of the receiver. This is the main strength of asymmetric cryptography. (Ionos 2019.)

In the communication, when the sender, which will be called user 1, wants to send a message or data to the receiver, user 2, user 1 encrypts the information using the public key from user 2. By using the public key from user 2, the message can only be decrypted with the private key from the same user. The process is shown in the next image (figure 20). (Ionos 2019.)

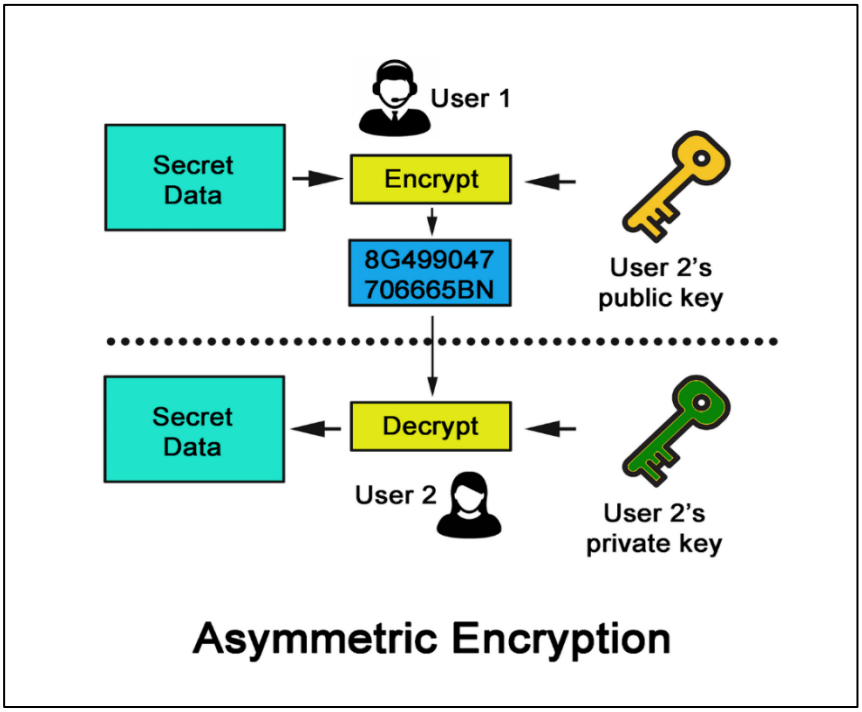

Figure 20. Communication process in asymmetric cryptography (Sheets 2019)

In the example, the advantages of asymmetric cryptography are shown. Anyone can use the public key from user 2 and only this user can see the information encrypted with that key as it can only be decrypted with his private key. Therefore, the problem of sharing and distributing the keys is not present here as it was in symmetric cryptography. The public key can be seen by everybody so there is no need of having a very secure channel to send this information. (Ionos 2019.)

On the other hand, there are some drawbacks in this kind of cryptographic methods related to identity. User 2 can not be sure who sent the message. It could exist the possibility that another user, which will be user 3 for this case, had the public key from user 2 and sent the message instead of user 1 with some bad intentions like sending malicious software. (Ionos 2019.)

Another case can happen. For example, user 1 can not be sure at all that the public key was from user 2. User 3 might create a key to pass it off as the real one in order to capture messages sent between user 1 and user 2. This is the reason why this kind of cryptography is a way of checking the authenticity and identity of the parts involved in the communication. There are two ways to do that task: digital certificates and digital signatures. (Ionos 2019.)

Digital certificates are used to check the authenticity of the public keys. On the other hand, digital signatures are related to identifying the sender of the message. For that purpose, the author of the message uses his/her private key to generate a signature. This signature is verified with the public key from the sender once the message gets to the receiver (Ionos 2019.). This will be explained in more detail in the next sections.

5.2 RSA algorithm

The RSA algorithm was the first asymmetric cipher algorithm. Its name has the origin from the mathematicians Rivest, Shamir and Adleman, who created it in 1977. It can be used for both encryption of data and generating digital signatures. It is considered one of the best public key algorithms. As described previously for asymmetric cryptography, public key algorithms are based on using a pair of keys: the public key, used for encrypting; and the private key used for decrypting. (Ionos 2019.)

The idea about using this pair of keys in the communication process comes from the cryptographs Withfield Diffie and Martin Hellman, who disclosed, in 1976, the exchange keys protocol Diffie Hellman (DH). This protocol allows establishing a key using a non-secure channel for its transmission. (Ionos 2019.)

For the algorithm, the investigators used unidirectional mathematical functions. Its calculation is not a difficult task. However, the inverse of it is complicated as it requires a big amount of computational operations that take a lot of time to execute. This is one of the main characteristics of RSA.

The RSA algorithm is based on the multiplication of large prime numbers. The multiplication itself is easy to perform. Nevertheless, there are not algorithms that decompose, in an effective way, a number in its prime factors. (Quirantes 2018.) This problem is described in the following example.

Suppose that we take the numbers 14629 and 30491. The result of multiplying them is 446052839. This number can be decomposed in four divisors: number one, itself and the two numbers which have been used to obtain the value. For the RSA only these two numbers are used for encryption and decryption. (Quirantes 2018.)

The RSA algorithm starts with the choice of these two numbers which are not public. These will be called p and q. As seen previously, these values are multiplied to give as a result a new number, which will be called n and will be public. For generating the keys, another value is used, which can be called F number. The F number is calculated in this way: (Quirantes 2018.)

$$F = (p - 1) * (q - 1)$$

The public key will be composed of the number n and another value called e. This value e and F will be relative prime numbers. These numbers are relative primer numbers if they do not have common divisors between them, except of 1 of course. For example, 8 and 15 do not have common divisors different from 1, as 3, 5 and 15 are not divisors of 8. The same situation happens to 15, as 2, 4 and 8 are not divisors of this value. (Quirantes 2018.)

On the other hand, the private key, called d, is a number that fulfills e*d mod F = 1. In this case, it is said that d is the inverse of e mod F. To ensure that this is fulfilled, numbers e and F are relative primer numbers. (Quirantes 2018.)

Once the keys are generated, the cipher and decipher of the message sent can be done. The message will be represented with M and the ciphered one with C. To generate C and M, the following expressions are used. (Quirantes 2018.)

$$C = M^e mod\ n$$

$$M = C^d mod\ n$$

In this way, the messages can be encrypted and decrypted. In the first expression, it can be seen that is used the number e which is part of the public key. On the other hand, the number d is used in the decryption, which is part of the private key. (Quirantes 2018.)

5.2.1 Advantages

As shown above, the RSA algorithm is one of the most secure algorithms used nowadays for encrypting information. Decomposing a number in its prime factors is a computationally complex task. There are no algorithms that can properly execute this task. In addition, the length of the keys used provides more security to this method. RSA can use large sizes as 1024 or 2048 bits. Therefore, figuring the key out by making a brute force attack testing all the possibilities is not effective owing to these key sizes. (Patil et al. 2016.)

Another feature that provides better protection is the fact of using a pair of keys. Although one of them is public because it is needed to make the encryption process of the information sent, the other key is kept secret. Therefore, in this algorithm, there is no problem related to sharing keys in an insecure channel as in some symmetric algorithms like AES or DES. (Ionos 2019)

5.2.2 Disadvantages

On the other hand, this algorithm has some disadvantages compared to the other methods previously explained. RSA is computationally slower than symmetric algorithms. The key sizes and the operations executed affect its performance for both encryption and decryption processes. (Patil et al. 2016)

In the next image (figure 21), the algorithm RSA is compared with other encryption methods like DES, TDES, AES and Blowfish. The results show that RSA is the slowest algorithm. The different algorithms have been applied to encrypt files with different sizes.

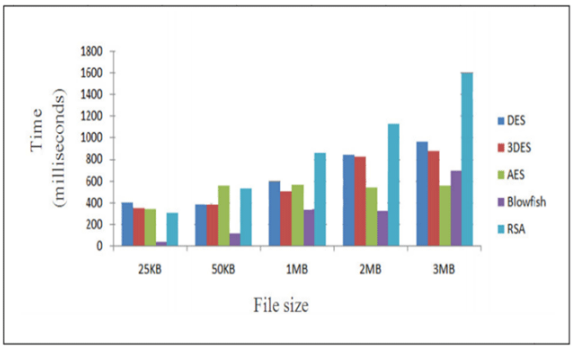

Figure 21. Comparison of encryption speed for different file sizes using different encryption algorithms (Patil et al. 2016)

5.3 Hash functions

One of the most important aspects of asymmetric cryptography is knowing the parts involved in the communication. When a message is received by someone, this person needs to know who is the author of the message as the public key used to encrypt maybe is being used by another user who is not involved in the communication. Hash functions help in this task. (González-Tablas Ferreres 2018d.)

Hash functions generate an identifier for a block of data which is sent as input to the function. The result obtained is called digest. This digest has a fixed length. However, the input data does not have any requirements and its size is variable. (González-Tablas Ferreres et al. 2018d.)

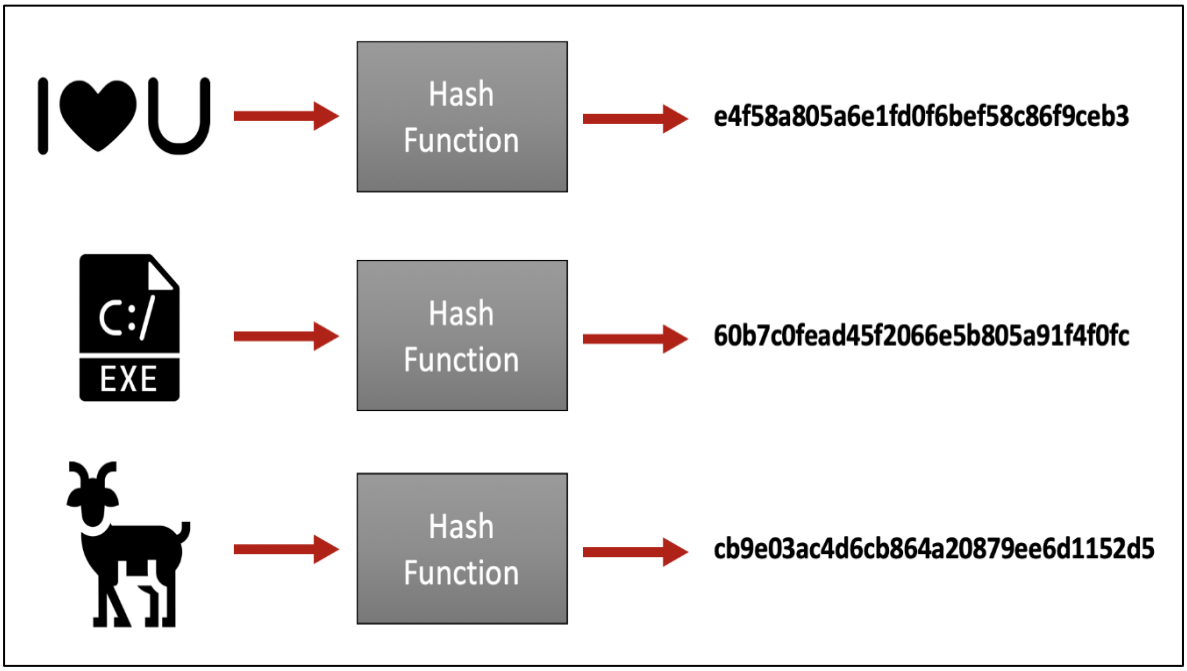

Figure 22. Overview of a Hash Function (Roccia 2019)

As shown in the previous image (figure 22), the hash function generates a digest for each kind of data. This value is unique and identifies the data introduced. In addition, is shown that all the digests produced have the same length. The lengths of the digests are not always the same and vary depending on the hash function or hash algorithms which has been applied to the data.

A hash function has a defined space of digests or hashes for a given function, which is calculated using the expression 2^n. For this formula, n is the number of bits of the output

of the hash function. This means, with a higher number of bits, the space of hashes increases. (González-Tablas Ferreres et al. 2018d.)

Although has been said that every hash function generates a unique identifier for any data given as input, collisions may occur. A collision takes place when the same hash value is generated from different input data given a hash function. This can be described mathematically in the following way: H(M) = H(M'). In this expression, M and M' are input data and H() is the hash function which is applied to a determined input. (González-Tablas Ferreres et al. 2018d.)

As said previously, a hash function aims to generate an identifier for every kind of data introduced as input to the function. Not any function can be considered a hash function as some requirements have to be fulfilled.

- The function has to be applicable to input messages of any length (González-Tablas Ferreres et al. 2018d).
- The output has the same length for any input data (González-Tablas Ferreres et al. 2018d).
- **Diffusion**. The hash has to change at least half of the bits approximately if, at least, a single bit is modified. (González-Tablas Ferreres et al. 2018d.)
- **Deterministic.** When a function is applied to a block of data, the output has to be always the same. (González-Tablas Ferreres et al. 2018d.)
- **Efficient.** The hash calculation should be fast for hardware and software implementations. (González-Tablas Ferreres et al. 2018d.)
- **Preimage resistant.** For any given hash value h, it is computationally impossible to find a message M' whose hash matches the first one (H(M) = h). (González-Tablas Ferreres et al. 2018d.)
- **Second preimages resistant**. It is computationally impossible to find a message M' for a given message M, where both hashes are the same. (González-Tablas Ferreres et al. 2018d.)
- **Collision resistant.** It is computationally impossible to find two messages, M and M', whose digests are the same. (González-Tablas Ferreres et al. 2018d.)

When something is said to be computationally impossible, this means that there are no algorithms or techniques to search collisions that are more efficient than brute force at-

tacks. In addition, if the digest spaces are big enough, it can be said that the probability to find a collision is 0 in a reasonable time. (González-Tablas Ferreres et al. 2018.)

Digital signatures

One of the uses of hash functions is shown in the digital signatures. As explained previously, digital signatures are one of the mechanisms used to authenticate the user who sent the message received in the communication. This feature is needed in asymmetric cryptography as the public key can be obtained by another user who is not involved in the communication between sender and receiver.

In this authentication process, the hash functions are an essential element. Normally, when a message is sent and signed a hash function is applied to the data in order to generate a digest. This digest is generated by the receiver once he or she receives the message. The digest is calculated using the same function or algorithm used by the sender. After that, these values are compared. If the digests are the same, the message is validated. Otherwise, the receiver is not sure about the identity of the sender. The signing is described below in figure 23 (El Khatabi 2019.)

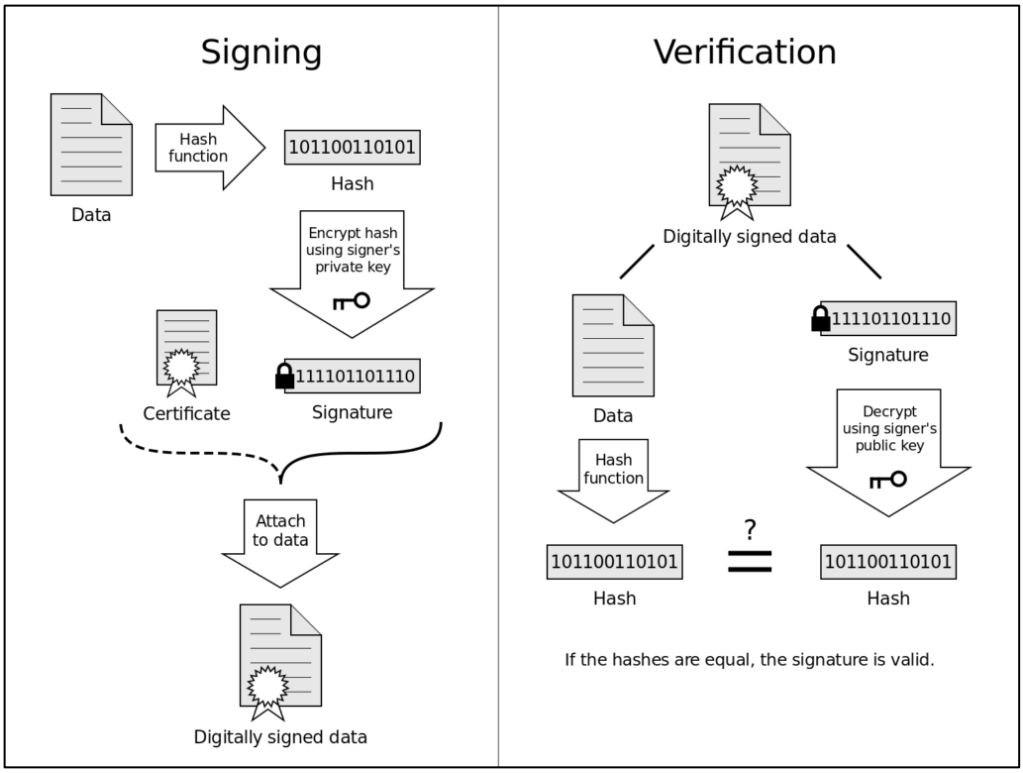

Figure 23. Diagram of signing and verification processes in digital signatures (CRS4)

When the sender wants to send some data, first a hash function is applied to generate the corresponding digest. This digest is ciphered with the private key from the sender. This encrypted digest is the signature of the sender. This will be used to check the identity of the person who sent the message. Once this is done, the data is sent, attaching the signature (encrypted hash) and a digital certificate. (El Khatabi 2019.)

The digital certificate identifies the one who has the certificate. These certificates contain the public key of the owner of the certificate and are signed by a Certificate Authority (CA). A CA is an organization that validates the identity of an entity (person, website, etc) and generates a pair of public and private keys. Furthermore, these organizations can associate a public key that already exists to the entity that made a request and provided this public key. After the identity is validated, a digital certificated signed by the CA is given to the applicant. (El Khatabi 2019.)

Back to the signature process, once the message reaches the receiver, the verification process takes place. The data is deciphered and divided into two parts: the data itself and the signature. For the data, the hash function is applied and it generates a digest. For the signature, this is deciphered using the public key from the signer, the sender. This public key is included in the certificate sent in the data. (El Khatabi 2019.)

After applying the public key to the signature, another digest is obtained. Both digests, this and the one which comes from the data of the message received, are compared. If both are equal, the signature is verified successfully as there have not been modifications in the data received. (El Khatabi 2019.)

6 Case: Multiple encryption method

6.1 Description

The case is based on introducing and defining a method that will consist in encrypting the data which is going to be sent multiple times using different algorithms. Most of these algorithms will be symmetric algorithms. As seen in previous chapters, this means that these will use the same key for both encryption and decryption. These methods will be used following a sequence that will be known by the sender and receiver involved in the communication. The method is shown in the following picture.

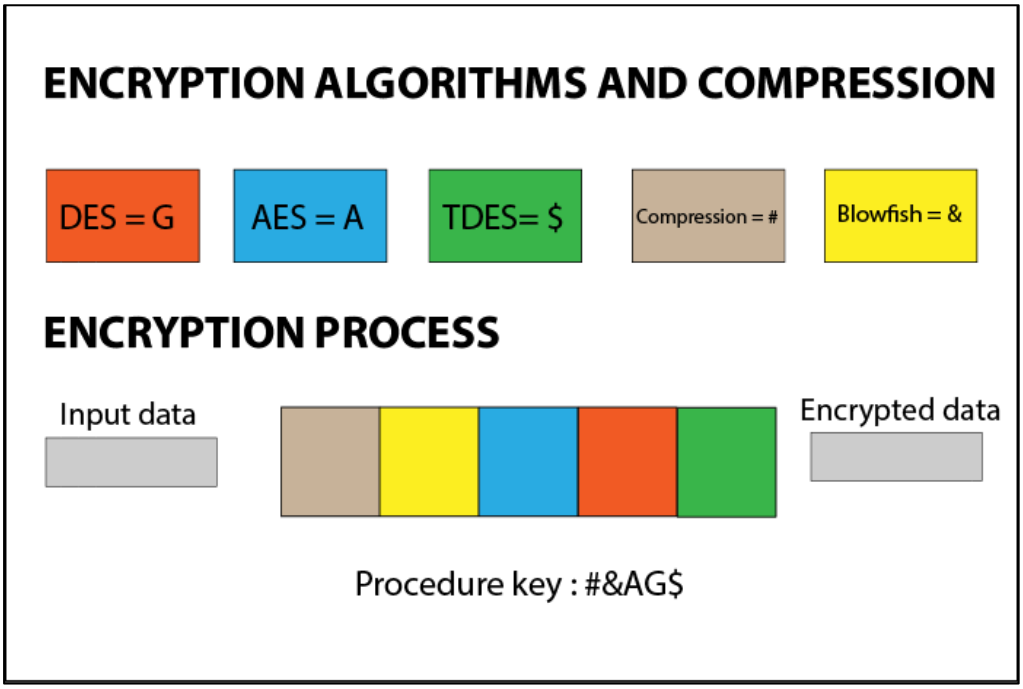

Figure 24. Diagram of the multiple encryption method.

The order of how the algorithms have been used will be told to the receiver by a key that will indicate the algorithm used and when it has been applied during the encryption This is indicated in the previous image (figure 24). Thanks to this key, the data received will be readable for the user that receives the message as the meaning of the key will be known before the data is received by him/her.

In addition, the method will include another feature that will provide more security to it. This feature is compression. The compression will reduce the total size of the information sent and will make more complicated the decryption for someone who is not involved in

the communication and wants to read the information exchanged between sender and receiver.

The reason of this is that the original data will suffer some modifications during the encryption, so this will complicate the task of cryptanalysis. Most times the data is analyzed in order to figure out how it has been encrypted. However, as the compressed data has some variations compared to the original, from that, conclusions can not be drawn so easily.

The key will be composed of a sequence of bits that will indicate the algorithms, the compression and the order of how this has been applied. Each element will be identified in the key by some bits that will be represented as characters to make the description easier. The same will be applied to the compression in order to know if it has been applied and at which point of the encryption.

6.1.1 Compression

Compression is one of the main features on which this method is based. Compression can be defined as a process that receives some data as input and provides, as output, data whose original size has been reduced. The input data can be one single element, like a file, or multiple elements like more than one file. Compression allows reducing the amount of space taken up in disks or other storage devices, so it is commonly used in order to save more space.

Therefore, the compression applied to the exchange of information provides faster communications. If the files or the messages sent to someone are small, the time that this data lasts to get to the sender will be decreased. As the amount of information that has to be transmitted is less than the original without compression, the communication process takes less time.

In order to justify why compression is used in the encryption process, it is important to know first how it works. Compression, as explained previously, is the process of reducing the size of the data. One of the bases of compression is redundancy, which refers to the repetition of some content in the data which wants to be compressed. (Morales Sandoval 2003.)

The action of reducing the amount of information is important nowadays. Currently, some technologies provide faster internet connections than in the past. For example, anyone can have an internet connection of 100 Mbps thanks to optic fiber. However, there are some places when this technology is not available or maybe someone can not afford it.

Therefore, it is always a good practice to try to adapt to the worst conditions and be as optimal as possible.

For analyzing the compression, some parameters are taken into account to measure the performance. The first one is the compression ratio, which indicates how much the size of the compressed file has been reduced from the original one. This parameter is calculated using the following expression.

$$Compression\ ratio = \frac{Bytes\ of\ compressed\ file}{Bytes\ of\ original\ file}$$

To see better how it works, an example will be given. There is a file of 580 bytes which is going to be compressed. After the compression, the size of the file is 400 bytes. If the compression is applied for these values, the result will be 0.69 approximately. This means that the compression gives, as a result, a file whose size is 0.69 times the size of the file without compression. In other words, it can be said that after the compression, the size of the file is 69 % of the original one.

On the other hand, the second parameter used to measure the performance of the compression is the compression factor. The compression factor indicates by how much the size of a compressed file has been reduced compared to the original one. It is calculated using the following expression.

$$Compression\ factor = \frac{Bytes\ of\ original\ file}{Bytes\ of\ compressed\ file}$$

As made previously for the compression ratio, an example will be given for this parameter too. The file has an original size of 600 bytes and the size after compression is 400 bytes. With these values, the compression factor will be 1.5. This means that the size of the file has been reduced 1.5 times.

Compression methods

The compression methods can be divided in two categories: lossy and lossless compression. A compression is lossless when the original file or data can be generated again from the compressed version. It is commonly used, for example, for compressing text as the bits from it can not be lost as the text needs to be understandable once is decompressed. (Miguel Morales Sandoval 2003.)

The lossy compression refers to those methods that, after applying the compression, this data is not an exact version of the original one, only an approximation. This happens because some content that is not considered relevant is removed from the data compressed.

This kind of compression is commonly used for image, audio and video compression applications. The compressing is made by removing some irrelevant information. (Morales Sandoval 2003.)

For this case, the compression used will be lossless as the encryption will be introduced and explained for files that can not accept losses of data like file texts. However, for further versions of the method, this can be defined as it can be applied too for this multiple encryption method.

6.1.2 Algorithms used

As described previously, this method is knowns as multiple encryption method. The method will encrypt multiple times the data before this is sent to the receiver. For that purpose, some existing algorithms are used for this process. One of the advantages of this method is that most of the algorithms can be applied in this method. Although an algorithm is less secure than another used in the process, as many algorithms have been applied an attacker does not know at which moment or in which stage has been performed in the encryption of data.

For example, the DES algorithm has its main weakness in its key size as only 56 bits are useful because the other bits are used for parity. Currently, this algorithm is not being used because of its security problems. This is the reason why TDES and AES were developed as has been explained in previous chapters.

Although these problems, if another encryption method like AES is performed after, but the attacker does not know the key used to encrypt with this algorithm or does not know if this algorithm has been executed after, the decryption process becomes very difficult without the knowledge of these parameters.

Even though some problems are shown by some algorithms, these can be applied in this method as they are used together with other algorithms considered secure as AES is nowadays. This is the reason why some algorithms like DES are used in this method. However, the list of algorithms used is shown as an example in order to explain better the method. The algorithms mentioned later can be changed or replaced by other ones. Nevertheless, for future real implementations, the best option will be to choose the most secure algorithms that are developed and used currently. These are the algorithms used for exemplifying the method: DES, AES, Triple DES and RSA.

6.1.3 Operating process

As described previously, the multiple encryption method will use different algorithms in order to make the encryption process. In addition, compression is added to the process in order to make the cryptanalysis more complicated for a possible attacker. The encryption is made in a sequence known for both sender and receiver. The sender, of course, knows the way how the data has been encrypted as has been made by himself. However, the receiver needs to know this sequence to make the inverse process.

The process of encryption is known for both parts thanks to the procedure key. The procedure key will consist of different bits that will indicate which algorithm has been used and when has been applied. All the information is represented in bits by computers, but in order to make a clear explanation, the key will be composed of different characters that will identify each algorithm used.

Once the algorithms used are known by the receiver, the process of the decryption is very easy to perform. The key will be read by the receiver from left to right, which is the same that read it from the most significant bits to the less significant ones. Therefore, as the algorithms and the order in which they have been executed are known, the only thing that has to make the receiver is executing the inverse processes of each encryption algorithm following the inverse order of the sequence in the procedure key.

In addition, if compression is performed, this is also indicated in the procedure key. Compression will be, in terms of this method, another technique or algorithm performed in the encryption process. However, it provides some interesting features and advantages described previously that are very useful for this case.

About the algorithms used during the process of encryption, these use a single key to perform both encryption and decryption as seen in previous chapters. To make easier the process, a single key could be used for all the symmetric algorithms that are executed during the encryption process. However, this can be a problem.

It is very recommendable to use different keys for each algorithm performed during the process of encryption. Reusing the same key for different algorithms can help an attacker to find some patterns in the ciphered data that can give him/her clues about the way the information has been protected. Therefore, the different keys for the algorithms used will be shared among the parts involved in the communication. This data will be transmitted using RSA which is an asymmetric cryptographic algorithm.

Asymmetric cryptography uses a pair of keys: one public and one private. For sending the key of each symmetric algorithm, this will be encrypted with the public one from the receiver. After that, this will be decrypted with the private key from the receiver too as this one can only be used for this purpose. If the information is encrypted with the public key, it needs to be decrypted with the private one of the same owner. In this way, the keys will be transmitted securely and both parts will know all the necessary information to encrypt and decrypt during the communication.

6.1.4 Example of use

Once the method has been described, an example of use will be given. User 1 wants to send a message to user 2. The information needs to be protected during the communication process so the multiple encryption method is used for that purpose. To execute it properly, the algorithms that can be used during encryption are communicated to both sender and receiver; and its identification too.

The identification of each algorithm will be a single character represented by a byte. In addition, the compression will be also be identified by a byte to know if it has been used and when during the encryption. Therefore, to make the communication possible, the characters have to be known by both sender and receiver. The characters which identify each algorithm and the compression are defined and communicated to the receiver.

For the sharing process of these values (and also the keys used by the symmetric algorithms performed in this method) RSA algorithm is used. As explained in previous sections, both sender and receiver generate a pair of keys, one public and one private. The algorithms that will be available for the encryption and their identifiers will be encrypted using the public key from the receiver (user 2) and this information will be decrypted by user 2 using his/her private key.

The advantage of using RSA in this stage is that the information about the algorithm will be only known by user 2 as this data can only be decrypted with the private key of user 2. Therefore, only user 2 can see the information. An example of how this information can be sent is shown here.

- DES, key_des → K
- TDES, key_tdes → @
- AES, key_aes →P
- Compression → I

Internally, the method will work with bits or bytes. The use of characters helps to make the explanation easier and clearer. This is the reason why this is shown in this way. In addition, the symmetric keys by each one of the algorithms are indicated as they are also needed for the decryption.

Once this information gets to user 2, the exchange of messages or data can be made. User 1 wants to send the message ‘Hello user 2’. For that, user 1 encrypts this data using the following procedure key: IP@K. This key means that the encryption has followed these steps. First, the message has been compressed (I). After that, the algorithms AES, TDES and DES have been executed to encrypt (P@K). The result is an intelligible message that has to be deciphered.

When this message reaches user 2, this one reads the procedure key and executes the inverse sequence. First, the algorithms DES, TDES and AES are executed. In this case, the same algorithms are used but applying their processes to decrypt. After that, the output of these processes is the compressed data which has to be uncompressed at this point. Once all the operations are performed user reads the message ‘Hello user 2’.

6.2 Example of implementation

In order to show how this method works and to demonstrate that this can be implemented to be used in real systems, a little application has been developed. For the implementation, the programming language which has been used is JavaScript. JavaScript is one of the most used programming languages in the world and is the one used for web development. This is the reason why this language has been chosen for developing this application.

In JavaScript, some lybraries have some functions implemented that perform the encryption algorithms described in previous sections. One of these libraries is CryptoJS, which is the one used for this implementation of the application. Its main features are the speed of executing the encryption methods and the ease to use them.

On the other hand, JavaScript has also some methods that implement the compression of text in order to reduce the amount of data transmitted. In this case, the compression function in the implementation of this method is lz-string. Lz-string is a implementation of the LZW (Lempel Ziff Welch) compression algorithm.

This technique identifies phrases that are repeated or recurrent in the text. These phrases are replaced by codes, which take less space and reduces the size of the file. In this way,

iff there are some phrases or words that appear a lot in the text, the codes that substitute them will decrease the amount of the data transmitted. (Sharda 2021.)

Finally, Boostrap has been used for the user interface. Bootstrap is a framework used for front-end development. That means it is only focused on the way that user interacts with an application and tries to provide the best user experience possible. It contains a lot of graphic components which makes the task of creating user-friendly interfaces easier.

6.2.1 Source code

Using all the technologies explained and described previously, an implementation of the multiple encryption method has been developed. For this one, the algorithms DES, Triple DES and AES have been used. Each one of these algorithms use a different key to encrypt. In addition, LZW compression is used to try to reduce the amount of data transmitted.

The implementation consists in two main functions: encrypt and decrypt. The encrypt function takes the text which is going to be ciphered and the procedure key which indicates the sequence of encryption followed. The decryption function performs the inverse process in order to get the original data sent. Figure 25 shows how the encryption function is implemented.

```
function encrypt(proc_key, algorithmsList) {
    var text = document.getElementById('file-content').value;
    var enc_text = "";
    var e_text = text;
    for (var i = 0; i < proc_key.length; i++) {
        switch (proc_key[i]) {
            case 'I':
                enc_text = CryptoJS.AES.encrypt(e_text, algorithmsList[proc_key[i]].key).toString();
                break;
            case 'C':
                enc_text = CryptoJS.DES.encrypt(e_text, algorithmsList[proc_key[i]].key).toString();
                break;
            case 'V':
                enc_text = CryptoJS.TripleDES.encrypt(e_text, algorithmsList[proc_key[i]].key).toString();
                break;
            case '$':
                enc_text = LZString.compressToUTF16(e_text);
                break;
        }
        e_text = enc_text;
    }
    return enc_text;

}
```

Figure 25. Source code of the encryption function of the method.

This function takes the text and encrypts it. As it can be seen, for each character of the procedure key it applies an encryption algorithm or compression. For I AES is performed, for C DES, for V Triple DES and for $ the compression is applied. All these mehods are implemented by CryptoJS library as it has been said previously and compression is made using lz-string.

Furthermore, for the encryption methods, some parameters are needed to execute the functions. These parameters are the text that is going to be encrypted and a key used for the algorithm to encrypt. For each algorithm, a different key is used. All this information, encryption algorithms and keys, is known before in order to make the decryption process by the receiver once the procedure key is transmitted. This is necessary as if this data is not known the ciphered text will not be decrypted. For performing the decryption, the process is almost the same but applying the decryption operations of the algorithms and making the sequence indicated by the procedure key in the opposite order. Therefore, the same information about the algorithms is required for this case too. The implementation of the decryption function is shown in figure 26.

```
function decrypt(encrypt, proc_key, algorithmsList) {
    var dec_text = encrypt;
    var bytes;

    for (var i = proc_key.length - 1; i >= 0; i--) {
        switch (proc_key[i]) {
            case 'I':
                bytes = CryptoJS.AES.decrypt(dec_text, algorithmsList[proc_key[i]].key);
                dec_text = bytes.toString(CryptoJS.enc.Utf8);
                break;
            case 'C':
                bytes = CryptoJS.DES.decrypt(dec_text, algorithmsList[proc_key[i]].key);
                dec_text = bytes.toString(CryptoJS.enc.Utf8);
                break;
            case 'V':
                bytes = CryptoJS.TripleDES.decrypt(dec_text, algorithmsList[proc_key[i]].key);
                dec_text = bytes.toString(CryptoJS.enc.Utf8);
                break;
            case '$':
                dec_text = LZString.decompressFromUTF16(dec_text);
                break;
        }
    }
    return dec_text;
}
```

Figure 26. Source code of the decryption function of the method.

The information about the algorithms that will be used in the communication will be sent previously and stored for current and future communications. In this way, this data will be

available and the access will be fast. For this case, only the implementation of the method has been made and not the communication of the procedure key and the information of the algorithms and compression used. However, as explained in previous sections, this would be made by applying RSA or another asymmetric cryptographic method in order to keep the communication secure. In this version of the method, this information can be accessed as shown in figure 27. In further implementations will be sent encrypted and then decrypted as explained previously.

```
var procedure_key = ['$', 'I', 'C', 'V'];
var algorithms = {
    'I': {
        name: 'AES',
        key: "mbjdscjdbjvd"
    },
    'C': {
        name: 'DES',
        key: "bpkxeq452389z"
    },
    'V': {
        name: 'TripleDES',
        key: "scbjdaxzzexw"
    },
    '$': {
        name: 'lz-compression'
    }
};
```

Figure 27. Procedure key and information of encryption algorithms and compression.

6.2.2 User interface

The user interface has been developed using Bootstrap framework and it is very simple to use for the user. It consists of some text areas which display the content which is going to be encrypted, the encrypted text once the method has been applied and the data after applying the decryption.

About the interaction with the application, the user can upload a text file and the content is shown in the text area. Next, the user can encrypt the data by pressing the Encrypt button and the application will show the encrypted text after the method has been executed.

Furthermore, to show that the text can be deciphered from the ciphered one, the user can also show the original text again by pressing the Decrypt button. Then, the original data

will be displayed from the ciphered one. In the next image (figure 28) there is an example of how this application works.

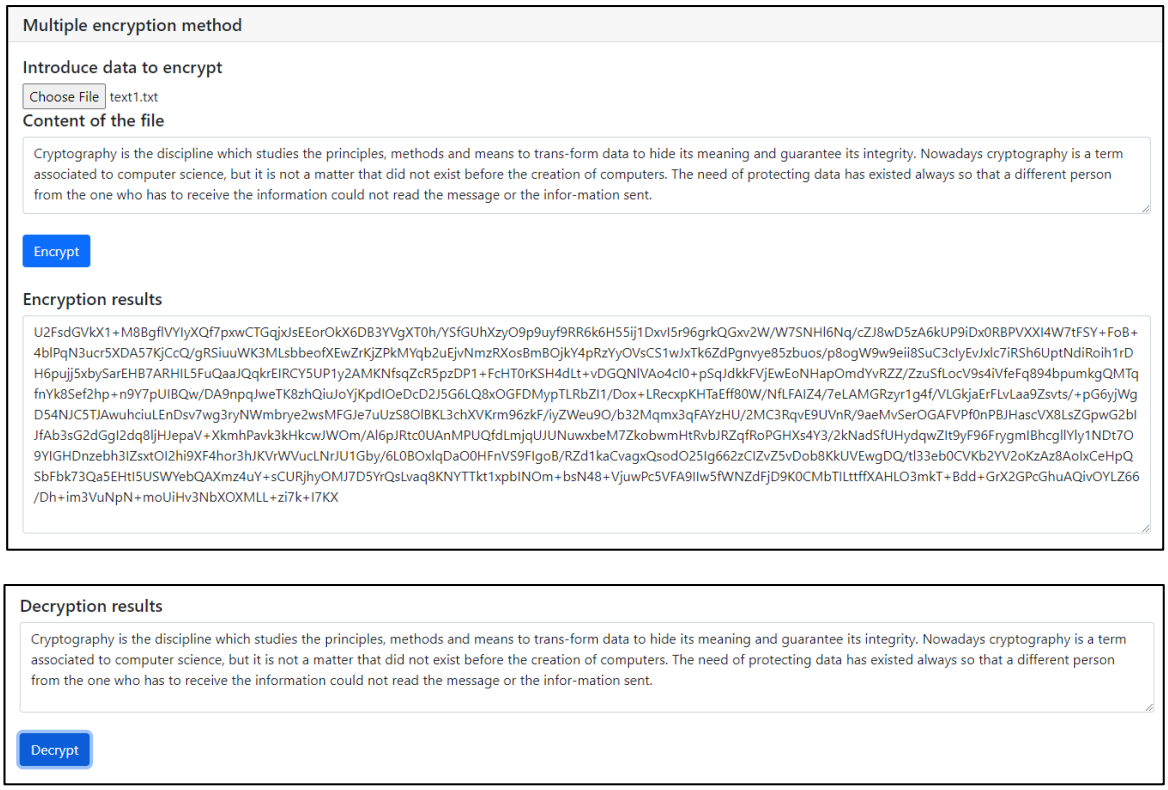

Figure 28. User interface of the application.

In the image above a text has been uploaded to the application and the content of it is shown. Next, the content of the file appears encrypted once the Encrypt button has been clicked. Finally, the original content is shown again after pressing the Decrypt button.

6.3 Analysis

After showing the implementation of this method, an analysis of the main features of the method will be made. As seen previously, the main aspects are the multiple encryption of the data and the compression of it. Some advantages and drawbacks will be commented in order to see the strengths and weaknesses of the method and which things can be improved to achieve better performance.

6.3.1 Applying multiple encryption algorithms

Applying multiple encryption algorithms before sending some information provides a big security layer to our data. Imagine that some data has been sent to a user and the encryption method used for that purpose is, for example, DES. DES algorithm is not an insecure algorithm itself. However, its main problem is its key length is only 56 bits as has been discussed previously. That means, it can be broken making brute force attacks to figure out the key that has been used during the encryption.

The advantage of this multiple encryption method is that even if an attacker knows that this algorithm has been used, he or she is not aware of when it has been applied. Therefore, if the result of the encryption of this method is after encrypted again using AES, the attacker has to overcome another security layer. To sum up, each layer applied adds more protection to the data sent during the communication. The complexity of decryption is very big for someone who knows neither the algorithms applied nor the keys used for the symmetric algorithms.

When the different algorithms are applied, different keys have to be used. Encrypting the data using a single key for all the algorithms applied is not the best way to perform this technique. Some clues and patterns can be given to possible attackers if the same key is used every time.

6.3.2 Compression in the encryption process

Another feature that includes this method is compression. As described previously, compression allows reducing the amount of data transmitted during communication. This means that the communications can be faster. However, despite the benefits of this process, this might not be performed so multiple times to the data which is going to be encrypted.

The compression is based on looking for redundancy in data in order to keep as much of the content the same as possible. For example, imagine that this message is going to be compressed: ‘I want to study computer science because I like computers’. When the compression method is applied, it finds that the word ‘computer’ is repeated so it can be replaced by other characters like ‘c9’. Now, the message is shorter as the word ‘computer’ has been substituted by ‘c9’.

When encryption is applied to data, repetitions and redundancy are less frequent. It is more difficult for compression methods to find redundancy in order the reduce the size of the data. This means that maybe the best option to apply the compression is before the multiple encryptions are executed in order to get the maximum benefit from the compression. It can be applied after encryption of course, but may not be very effective. The benefits of applying compression first can be seen in the following image (figure 29).

Compression method	Test set size	Compr. Size	Compr. + DES	Compr. + 3DES	Compr. + AES	Compr. + RC4
RLE	3229989	2796235	2796304 (+0,0021%)	2796304 (+0,0021%)	2796368 (+0,0041%)	2796235 (+0%)
Huffman	3229989	1833144	1833224 (+0,0025%)	1833224 (+0,0025%)	1833296 (+0,0047%)	1833144 (+0%)
Arithmetic	3229989	1215909	1215992 (+0,0026%)	1215992 (+0,0026%)	1216080 (+0,0053%)	1215909 (+0%)
LZW	3229989	1743266	1743344 (+0,0024%)	1743344 (+0,0024%)	1743424 (+0,0049%)	1743266 (+0%)

Figure 29. Cost in bytes of applying compression to data and then encryption (Bruno Carpieneri 2018a)

The image above is a table that shows the size in bytes after applying compression and encryption. Encryption and compression have been tested for different algorithms. The files used are a set of 17 elements that are text and binary files. These files were created by Tim Bell, Ian Witten, and John Clearly. (Carpieneri 2018c)

Therefore, it shows, as it was expected, that after compressing a file its size is reduced. Then, when the encryption is executed after the compression, the size remains more or less the same. Is shown that size has increased a little, but it is more or less the same as the rise is practically 0. With these results we can conclude that the combination of compression and encryption is working as the size of the files has not increased so much.

On the other hand, other results are shown when compression is applied after encryption. When the processes are applied in this order, sometimes the size of the files, once they are encrypted, is increased when these files are compressed as we can see in the following image.

Encryption method	Test set size	Encrypt. Size	Encrypt. + Huffman	Encrypt. + AC	Encrypt. + LZW	Encrypt. + RLE
RC4	3229989	3229989	3235861 (+5872 b.)	6376898 (+3146909 b.)	4587226 (+1357237 b.)	3242600 (+12611 b.)
DES	3229989	3230064	3235939 (+5950 b.)	6370696 (+3140707 b.)	4587115 (+1357126 b.)	3242550 (+12561 b.)
3DES	3229989	3230064	3235932 (+5943 b.)	6373622 (+3143633 b.)	4588580 (+1358591 b.)	3242570 (+12581 b.)
AES	3229989	3230128	3235985 (+5996 b.)	6370180 (+3140191 b.)	4586952 (+1356963 b.)	3242619 (+12630 b.)

Figure 30. Cost in bytes of applying encryption to data and then compression (Bruno Carpieneri 2018b)

The results in figure 30 show that the sizes of files are bigger after applying compression once the data is encrypted. These conclusions are the same for all the encryption methods tested. This means that is a better option to apply compression before encrypting data.

7 Summary

The importance of cryptography and encryption has been shown during the development of the thesis. Nowadays, new technologies are constantly used for chatting with friends or family and for buying online. All this data needs to be protected and transmitted in a secure way to not be obtained by anyone.

As seen in the previous sections, cryptography and encryption are not only related to computer science and telecommunications. Before the creation of these, encryption had been used in other ways. Cryptography was essential in order to decipher the Enigma machine. This allowed finishing Second World War earlier and avoiding lots of deaths owing to this conflict.

The development of these sectors over the years has provided us nowadays different algorithms that encrypt our information when this is sent through the internet. However, after some time some of them get insecure as cybercriminals or even cryptographs discover some weaknesses in these methods.

The purpose of this thesis was to introduce a multiple encryption method, adding compression to the data in order to add multiple security layers to the information. Therefore, the data would be very complicated to decipher.

This method has been described in detail. After that, a possible implementation has been developed to show better how this method works. This was the first version of the multiple encryption method, so it can be still improved a lot. For further developments, the communication of the algorithms and their keys would be implemented using RSA providing a secure way to share this information.

Furthermore, other encryption libraries will be needed as the one used had some problems. As seen previously, when the text was encrypted several times, the length of it increased despite applying also compression. Therefore these issues need to be fixed.

On the other hand, an analysis of the main features of the method has given some conclusions. The multiple encryptions make the cryptanalysis very difficult to decipher the information encrypted. As explained previously, the encryption has to be performed using a different key for each symmetric algorithm applied. If not, some clues may be given to decipher.

Regarding compression, this technique allows reducing the size of the data transmitted and makes more difficult the decryption for the entity that has neither the procedure key nor the keys used for the symmetric algorithms. However, applying compression after

encryption might not be very effective as redundancy in data is not found. Therefore, the performance of compression is lower.

The method described can be the first stage of a real implementation of this technique to provide more security in the current communications. The computational capacity is growing more every time and the current algorithms can become insecure in some years. In addition, quantum computing will provide very powerful computers that can be a threat to the existing encryption systems. This is the reason why some investigation is needed in this field to try to provide more secure methods and techniques as might be the one explained in the thesis.

References

Agarwal, D. & Marsh, K. 2015a. PC-1 Permutation table. Referenced on 2 May 2021. Available at https://www.projectrhea.org/rhea/images/thumb/6/65/KS1.png/250px-KS1.png

Agarwal, D. & Marsh, K. 2015b. PC-2 Permutation table. Referenced on 2 May 2021. Available at https://www.projectrhea.org/rhea/images/thumb/a/a6/KS4.png/250px-KS4.png

Alcover Garau, P. M. 2008a. Tema Criptografía. Conceptos generales. Content of a course which is about security in networks. Universidad Politécnica de Cartagena. Retrieved on 28 January 2021. Available at https://ocw.bib.upct.es/pluginfile.php/5303/mod_resource/content/1/Introduccion.pdf

Alcover Garau, P. M. 2008b. TEMA Modos de operación en el cifrado por bloques. Course lecture. Universidad Politécnica de Cartagena. Retrieved on 4 April 2021. Available at https://ocw.bib.upct.es/pluginfile.php/5310/mod_resource/content/1/MODOS_DE_OPERACION_CIFRADO_BLOQUES.pdf

Alsultanny, Y. 2007. Block diagram of the encryption of the CFB mode. Referenced on 4 April 2021. Available at https://www.researchgate.net/figure/Block-diagram-of-the-encryption-of-the-CFB-mode_fig1_255581885

Amiriki. 2011. Feistel cipher diagram. Referenced on 21 April 2021. Available at https://en.wikipedia.org/wiki/Feistel_cipher#/media/File:Feistel_cipher_diagram_en.svg

Boxcryptor. Cifrado AES-256. Boxcryptor. Retrieved on 3 May 2021. Available at https://www.boxcryptor.com/es/encryption/

BrainKart. AES Key Expansion. BrainKart. Retrieved on 4 May 2021. Available at https://www.brainkart.com/article/AES-Key-Expansion_8410/

Carpentieri, B. 2018a. Cost of encryption for one-dimensional data (compression + encryption). Referenced on 12 May 2021. Available at https://www.hindawi.com/journals/scn/2018/9591768/

Carpentieri, B. 2018b. Cost of encryption for one-dimensional data (encryption + compression). Referenced on 12 May 2021. Available at https://www.hindawi.com/journals/scn/2018/9591768/

Carpentieri, B. 2018c. Efficient Compression and Encryption for Digital Data Transmission. Hindawi. Retrieved on 12 May 2021. Available at https://www.hindawi.com/journals/scn/2018/9591768/

CRS4. Diagram illustrating how a simple digital signature is applied and verified. Referenced on 8 May 2021. Available at https://www.crs4.it/wp-content/uploads/2018/06/1500px-Digital_Signature_diagram.png

EcuRed contributors. 2018. Teoría de la información. EcuRed. Retrieved on 19 February 2021. Available at https://www.ecured.cu/index.php?title=Especial:Citar&page=Teor%C3%ADa_de_la_informaci%C3%B3n&id=3262083

El Khatabi, M. 2019. Digital Signatures are the Cybersecurity Vulnerability You Need to Stop Ignoring. CybelAngel. Retrieved on 8 May 2021. Available at https://cybelangel.com/blog/digital-signatures-are-the-cybersecurity-vulnerability-you-need-to-stop-ignoring/

Estrella Gutiérrez, S. 2006. Criptoanálisis del algoritmo RC4. Thesis. Universidad Nacional Autónoma de México. Retrieved on 5 May 2021. Available at http://132.248.9.195/pd2006/0603108/0603108.pdf

González-Tablas Ferreres, A. I. , de Fuentes García-Romero de Tejada, J.M. , González Manzano, L. , Martín González, P. 2018a. Introducción a la criptografía. Presentation for a cryptography and security course. Universidad Carlos III de Madrid. Retrieved on 22 January 2021. Available at http://ocw.uc3m.es/ingenieria-informatica/criptografia-y-seguridad-informatica/material-propio%20LorenaChema/L2_Intro_criptosistemas.pdf

González-Tablas Ferreres, A. I. , de Fuentes García-Romero de Tejada, J.M. , González Manzano, L. , Martín González, P. 2018b. Criptografía clásica y su criptoanálisis. Presentation for a cryptography and security course. Universidad Carlos III de Madrid. Retrieved on 26 February 2021. Available at http://ocw.uc3m.es/ingenieria-informatica/criptografia-y-seguridad-informatica/material-propio%20LorenaChema/L4_Criptoclasica.pdf

González-Tablas Ferreres, A. I. , de Fuentes García-Romero de Tejada, J.M. , González Manzano, L. , Martín González, P. 2018c. Criptosistemas Simétricos: Bloque. Presentation for a cryptography and security course. Universidad Carlos III de Madrid. Retrieved on 3 May 2021. Available at http://ocw.uc3m.es/ingenieria-informatica/criptografia-y-seguridad-informatica/material-propio-1/L5_Criptosistemas_simetricos.pdf-1

González-Tablas Ferreres, A. I. , de Fuentes García-Romero de Tejada, J.M. , González Manzano, L. , Martín González, P. 2018d. Funciones resumen. Presentation for a cryptography and security course. Universidad Carlos III de Madrid. Retrieved on 3 May 2021. Available at http://ocw.uc3m.es/ingenieria-informatica/criptografia-y-seguridad-informatica/material-propio%20LorenaChema/L9_Resumen.pdf

Hammond, J. 2015. Vignère square. Referenced on 15 February 2021. Available at https://s3-eu-west-1.amazonaws.com/scoop-cms/566e8c75ca2f3a5d5d8b45ae/1.PNG

Hebergement Webs. Caesar cipher diagram.jpg. Retrieved on 4 March 2021. Available at https://www.hebergementwebs.com/image/31/3184043ed6dd0b7b32a7b237b2449a4b.webp/criptografia-con-python---caesar-ciphercriptografia-con-python-caesar-cipher-0.webp

Ionos – Web hosting and cloud partner company. 2019. El encriptado informático: así se protege la comunicación. Retrieved on 29 January 2021. Available at https://www.ionos.es/digitalguide/servidores/seguridad/todo-sobre-los-metodos-de-encriptado/

ITCA-FEPADE. U6.1 Concepto de criptografía. ITCA-FEPADE. Retrieved on 22 January 2021. Available at https://virtual.itca.edu.sv/Mediadores/cms/u61_concepto_de_criptografa.html

Jesús Velasco, J. 2014. Breve historia de la criptografía. El Diario. Retrieved on 6 February 2021. Available at https://www.eldiario.es/turing/criptografia/breve-historia-criptografia_1_4878763.html

Jszigetvari. 2013. Enigma.jpg. Wikipedia. Retrieved on 9 February 2021. Available at https://upload.wikimedia.org/wikipedia/commons/a/ae/Enigma.jpg

Karthik, S & Muruganandam, A. 2014. Data Encryption and Decryption by Using Triple DES and Performance Analysis of Crypto System. International Journal of Scientific Engineering and Research (IJSER), 24-31.

Luringen. 2007. Skytale.png. Wikipedia. Retrieved on 6 February 2021. Available at https://en.wikipedia.org/wiki/Scytale#/media/File:Skytale.png

Mamdouh, M., Sadek, R. A. , El Ghoz, H. & El Far, Y. 2017. AES Encryption and Decryption Block Diagram. Referenced on 3 May 2021. Available at https://d3i71xaburhd42.cloudfront.net/e46071c7eb742bcd9798ddb29ff04db1163cd121/2-Figure1-1.png

Mitra, P. Golomb’s Randomness Postulates. Indian Institute of Technology Guwahati. Retrieved on 5 May 2021. Available at http://www.iitg.ac.in/pinaki/Golomb.pdf

Morales Sandoval, M. 2003. Notas sobre compresión de datos. Document about compression. Cinvestav Unidad Tamaulipas. Retrieved on 10 May 2021. Available at https://www.tamps.cinvestav.mx/~mmorales/documents/Compre.pdf

Nakov, S. 2018. Counter (CTR) mode encryption. Referenced on 5 April 2021. Available at https://cryptobook.nakov.com/symmetric-key-ciphers/cipher-block-modes

Nevon Projects. Diagram of TDES encryption and decryption. Retrieved on 3 May 2021. Available at http://nevonprojects.com/wp-content/uploads/2015/06/Triple-Des-image.png

Owais, A. 2019. AES key expansion algorithm. Referenced on 4 May 2021. Available at https://d107a8nc3g2c4h.cloudfront.net/images/blog/wp-content/uploads/AES.jpg

Patil, P., Narayankar, P., Narayan, D G, Meena SM. 2016. Encryption time vs. File size for DES, 3DES, AES, Blowfish and RSA. Referenced on 7 May 2021. Available at https://reader.elsevier.com/reader/sd/pii/S1877050916001101?token=83B19502B67425DC51D54F73DF61CABD454C7D94048534B58853BF733B6CB6B031A99D7B63F9EADE6546E479ACBB3F97&originRegion=eu-west-1&originCreation=20210507143228

Quirantes, A. 2018. Así funciona el algoritmo RSA. El Profe de Física. Retrieved on 7 May 2021. Available at https://elprofedefisica.naukas.com/2018/01/22/asi-funciona-el-algoritmo-rsa/

Ramió Aguirre, J. & Muñoz Muñoz, A. 2015. Píldora nº 32: ¿Qué son los postulados de Golomb?. Cryptography course content. Criptored. Retrieved on 5 May 2021. Available at http://www.criptored.upm.es/thoth/material/texto/pildora032.pdf

Ramió Aguirre, J. & Muñoz Muñoz, A. 2016a. Video on YouTube. Píldora formativa 35: ¿Cómo funciona el algoritmo RC4?. Universidad Politécnica de Madrid. Retrieved on 5 May 2021. Available at https://www.youtube.com/watch?v=G3HajuqYH2U

Ramió Aguirre, J. 2016b. Lección 7: Algoritmos de cifra por transposición o permutación. Universidad Politécnica de Madrid. Retrieved on 4 March 2021. Available at http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion7.html

Ribagorda Garnacho, A. 2010. Lección 1. Historia de la criptografía y su desarrollo en Europa. Retrieved on 8 February 2021. Available at http://www.criptored.upm.es/intypedia/video.php?id=historia-criptografia&lang=es

Roccia, T. 2019. Overview of Hash Function. Referenced on 7 May 2021. Availbale at https://miro.medium.com/max/700/1*SCHCrY-GeCGjSbVk7P8kaQ.png

Sánchez Arriazu, J. 1999a. Descripción del algoritmo DES (Data Encryption Standard). Reterieved on 21 April 2021. Available at http://www.satorre.eu/descripcion_algoritmo_des.pdf

Sánchez Arriazu, J. 1999b. S boxes DES algorithm. Retrieved on 3 May 2021. Available at http://www.satorre.eu/descripcion_algoritmo_des.pdf

Servos, W. 2010. The Alberti cipher. Trinity College. Retrieved on 29 January 2020. Available at http://www.cs.trincoll.edu/~crypto/historical/alberti.html

Sharda, Y. 2021. Unix's LZW Compression Algorithm: How Does It Work?. Hackernoon. Retrieved on 17 May 2021. Available at https://hackernoon.com/unixs-lzw-compression-algorithm-how-does-it-work-cp65347h

Sheets, D. 2019. Asymmetric encryption. Referenced on 5 May 2021. Available at https://img.militaryaerospace.com/files/base/ebm/mae/image/2019/06/Curtiss_PRIVATE_PUBLIC_KEY_1_b.5d13d214b19e6.png?auto=format&fit=max&w=1440

Stallings, W. & Brown, L. 2012a. Computer Security: Principles and Practice. Book. Pearson. Retrieved on 2 April 2021. Available at https://hagesjo.se/static/books/Computer_security.pdf

Stallings, W. & Brown, L. 2012b. Single round of DES algorithm. Book. Pearson. Retrieved on 3 May 2021. Available at https://hagesjo.se/static/books/Computer_security.pdf

Stallings, W., Brown, L.V., Bauer, M. & Howard, M. 2008. Simplified Model of Symmetric Encryption. Referenced on 2 April 2021. Available at https://hagesjo.se/static/books/Computer_security.pdf

The British Museum. Enigma cipher machine: About the object. Retrieved on 16 February 2021. Available at http://www.teachinghistory100.org/objects/about_the_object/enigma_cipher_machine

Vazhayil, A. 2015. S-box. Referenced on 3 May 2021. Available at https://captanu.files.wordpress.com/2015/04/aes_sbox.jpg

Wikipedia. 2015. Atbash. Retrieved on 8 February 2021. Available at https://en.wikipedia.org/wiki/Atbash

XILINX. CBC_working_mode.png. Referenced on 4 April 2021. Available at https://xilinx.github.io/Vitis_Libraries/security/2020.1/_images/CBC_working_mode.png