UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Performance Evaluation of Cryptographic Security Algorithms

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Kevin Alexis Contreras Delgado

Tutor:

Iza Paredes Cristhian Rene, PhD.

Urcuquí, Diciembre 2024

Autoría

Yo, **Kevin Alexis Contreras Delgado**, con cédula de identidad 1721088928, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Diciembre 2024.

[illegible]

Kevin Alexis Contreras Delgado

CI:1721088928

Autorización de publicación

Yo, **Kevin Alexis Contreras Delgado**, con cédula de identidad 1721088928, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, Diciembre 2024.

[illegible]

Kevin Alexis Contreras Delgado

CI:1721088928

Dedication

I dedicate this work to my family who has always supported me and encouraged me to improve myself and always keep moving forward.

Kevin Alexis Contreras Delgado

Acknowledgment

To my beloved family, who have provided a constant source of encouragement and unwavering love. They have ingrained in me freedom, versatility, and the capacity to find strength in difficulty. I would like to express my gratitude to my advisor and mentor, Dr. Cristhian Iza, whose expert guidance and insights proved instrumental throughout this academic journey. I would also like to express my heartfelt gratitude to my girlfriend, who have been my support, sharing both the challenges and the joyful moments during this journey. Your affection and encouragement have been fundamental pillars at every step of the way.

Kevin Alexis Contreras Delgado

Resumen

En la sociedad actual, impulsada por la protección de datos y en la que la seguridad es primordial, la investigación de algoritmos de cifrado es crucial. Los rápidos avances tecnológicos han impulsado la necesidad de optimizar los algoritmos que demandan altos recursos computacionales y de fortalecer a aquellos que son vulnerables a ataques sofisticados. Esta investigación se centra en el análisis de rendimiento de algoritmos de cifrado simétricos (AES) y asimétricos (RSA, ECC), implementados y probados a través de interfaces graficas desarrolladas en Java, lenguaje de alto nivel. Las interfaces permiten a los usuarios configurar los parámetros de cada algoritmo y usar métricas de rendimiento como el tiempo de ejecución, tasa de procesamiento de datos, consumo de memoria, y consumo de CPU. Este trabajo examina el desempeño de cada algoritmo de acuerdo a los ajustes en los tamaños de texto y las configuraciones de parámetros que se definan. Los hallazgos confirman que el algoritmo RSA requiere valores más altos en cada una de las métricas ya establecidas, tomando el tipo de llave de 3072 bits. En contraste, el algoritmo ECC emerge como el más eficiente sobresaliendo tanto en tiempo de ejecución, consumo de memoria y tiempo de CPU. Esta investigación destaca la importancia elegir el algoritmo de cifrado adecuado según los recursos computacionales específicas y los requisitos de seguridad.

Palabras Clave:
Algoritmos de cifrado, AES, RSA, ECC, capacidad computacional, seguridad informática, rendimiento.

Abstract

In today's society data protection and security is critical, research on encryption algorithms is crucial. Rapid technological advances have driven the need to optimize computationally intensive algorithms and to strengthen those that are vulnerable to sophisticated attacks. This research focuses on the performance analysis of symmetric (AES) and asymmetric (RSA, ECC) encryption algorithms, implemented and tested through graphical interfaces developed in high-level Java language. The interfaces allow users to configure the parameters of each algorithm and measure the performance using metrics; such as, execution time, capacity, memory and CPU consumption. This work examines how each of these algorithms perform under various conditions, by adjusting the text size and other configuration parameters. The findings confirm that the RSA algorithm consumes significantly more computational resources, taking the 3072-bit key type. In contrast, the ECC algorithm emerges as the most efficient, excelling in both execution time, memory requirements, and CPU consumption. This research highlights the importance of choosing the appropriate encryption algorithm based on specific computational resources and security requirements.

Keywords:
Encryption algorithms, AES, RSA, ECC, computational capacity, computer security, performance.

Contents

List of Tables

List of Figures

Introduction

In the current digital era, where the world is increasingly interconnected, information security has become a critical aspect for individuals, companies, and governments. The cause is the increasing dependence on wireless communications and the exchange of sensitive data through unsecured channels. This fact has increased the need to protect information against external threats. Taking these ideas into account, cryptography emerges as a fundamental tool to safeguard the authenticity, integrity, and confidentiality of data.

Since cryptography is defined as the art of writing secret code, it has been able to evolve enormously from its classical origins to sophisticated modern techniques [13]. On the other hand, while classical methods focused mainly on preventing message interception [14], modern cryptographic techniques offer a much higher level of security [15], adapting to the needs of communications, high speed and others complex information systems.

Currently, the most used algorithms for encryption tasks due to their efficiency are Advanced Encryption Standard (AES), Rivest–Shamir– Adleman (RSA) and Elliptic Curve Cryptography (ECC), and for this reason they will be used in the comparison and implementation using a high-level and object-oriented language. Furthermore, they will be designed to operate independently, and a corresponding study and analysis will be proposed for each one. Each of these algorithms has unique characteristics that make them suitable for different applications in the area of data security. These encompass the use of AES for data transmission [16], RSA for securing electronic signatures [17], and ECC in fields like cryptocurrencies [18].

As mentioned in [19], the AES algorithm is a symmetric encryption method that has become the standard algorithm for encrypting data at rest and in communication. Its computational efficiency and high level of security make it ideal for encrypting large volumes of data. Similarly, this algorithm uses a single key to encrypt and decrypt, making it extremely fast compared to public key algorithms [20]. Additionally, its robustness against known attacks and its ability to be efficiently implemented in both hardware and software have contributed to its wide adoption in several computing sectors. The protection of mobile communications [21] and the secure storage of data in the cloud [22] are two of the sectors where the AES algorithm is best applied.

On the other hand, the RSA algorithm is an asymmetric encryption method that proposed the public key cryptography system. This encryption algorithm proposes the solution to the problem of key distribution, which is a challenge for symmetric encryption systems. As mentioned in [23, 24], the RSA algorithm is an important part of the modern public key infrastructure (PKI). This is because it includes the concept of authentication, being a method that allows establishing secure communication and remaining encrypted between both parties. Moreover, in [25], the PKI becomes important since both the sender and the receiver have not previously shared a common secret.

Another of the algorithms that will be studied in this research is the ECC [26], being widely used within the area of data security [27, 28, 29], slowly replacing RSA or AES. This asymmetric encryption method offers a level of security comparable to the encryption methods mentioned above, using much shorter keys. In [30], it is mentioned that the use of shorter keys makes ECC particularly attractive for devices with limited resources, such as mobile phones or devices belonging to the Internet of Things (IoT). Undoubtedly, these devices have features in common where storage, processing power and bandwidth are very limited. For this reason, the efficiency of the ECC algorithm both in terms of key size and processing speed positions it as a cutting-edge cryptographic technology [31, 32].

In this context, the present work focuses on the performance analysis of the AES, RSA, and ECC encryption algorithms. The Java language is used to implement each one and creating a graphic interface for each one. For each algorithm, a respective exhaustive evaluation is carried out and then a comparison among them as they are subjected to various scenarios and metrics. This analysis considers some aspects such as processing time, consumption of computational resources, and the scalability of each for different data sizes. This way, we will be able to have a deeper and more up-to-date knowledge of the performance of these algorithms. In addition, with the results obtained it will be possible to visualize how these algorithms behave depending on the configurations chosen when encrypting or decrypting data.

This research is structured as follows: Chapter 1 presents the fundamental concepts of each encryption algorithm and discusses the evolution of cryptography up to recent years. Chapter 2 outlines the latest research that employs the encryption methods analyzed in this thesis, highlighting the significance of these algorithms in the realm of data security. Chapter 3 describes the methodology to be followed for implementing each algorithm and its corresponding graphical interface, as well as detailing how to address the problem statement defined in the introduction. Finally, Chapters 4 and Chapter 5 present the results obtained, along with their respective discussions and final conclusions.

Problem statement

Although several studies have been conducted comparing encryption algorithms [33, 34, 35, 36], continuous advancements in both computational capacity and cryptanalysis techniques have triggered deep concerns in the area of data protection. Since technology will never stop evolving with new possibilities and innovations, this means that encryption algorithms that were previously considered secure can become vulnerable, forcing the adoption of more robust and complex encryption methods. This way, while encryption algorithms such as AES, RSA, and ECC offer numerous benefits in terms of security, they have also introduced a number of significant challenges.

One of the main challenges is the lack of a comparative and up-to-date analysis of the performance of these algorithms in real-world scenarios. This lack of detailed information can make it difficult for developers or security professionals to make correct decisions when selecting and implementing encryption algorithms. In addition, considering the effects that using a certain encryption algorithm would have on mobile devices and IoT systems easily complicates the implementation of these algorithms. Another challenge of this research is to be able to implement encryption algorithms that are sufficiently optimal and capable of capturing metrics of execution time, performance and memory consumption. This last process is the most important to be able to carry out a correct analysis and thus achieve a precise and fast understanding of the behavior of these algorithms. As a result, after the respective analysis of each algorithm and defining the system that has the best performance, we aspire to provide an effective and efficient solution to the growing concerns associated with the implementation of cryptographic algorithms in both conventional and emerging contexts.

Objectives

General Objective

Evaluate the performance of encryption algorithms for secure data communication.

Specific Objectives

1. Review all available research on the AES, RSA, and ECC encryption algorithms.
2. Develop graphical user interfaces for implementing and testing the encryption algorithms.
3. Implement the AES, RSA, and ECC encryption algorithms in Java language.
4. Compare the performance of AES, RSA, and ECC algorithms in terms of encryption time and computational resources required.
5. Analyze the strengths and weaknesses of each algorithm for different data sizes.

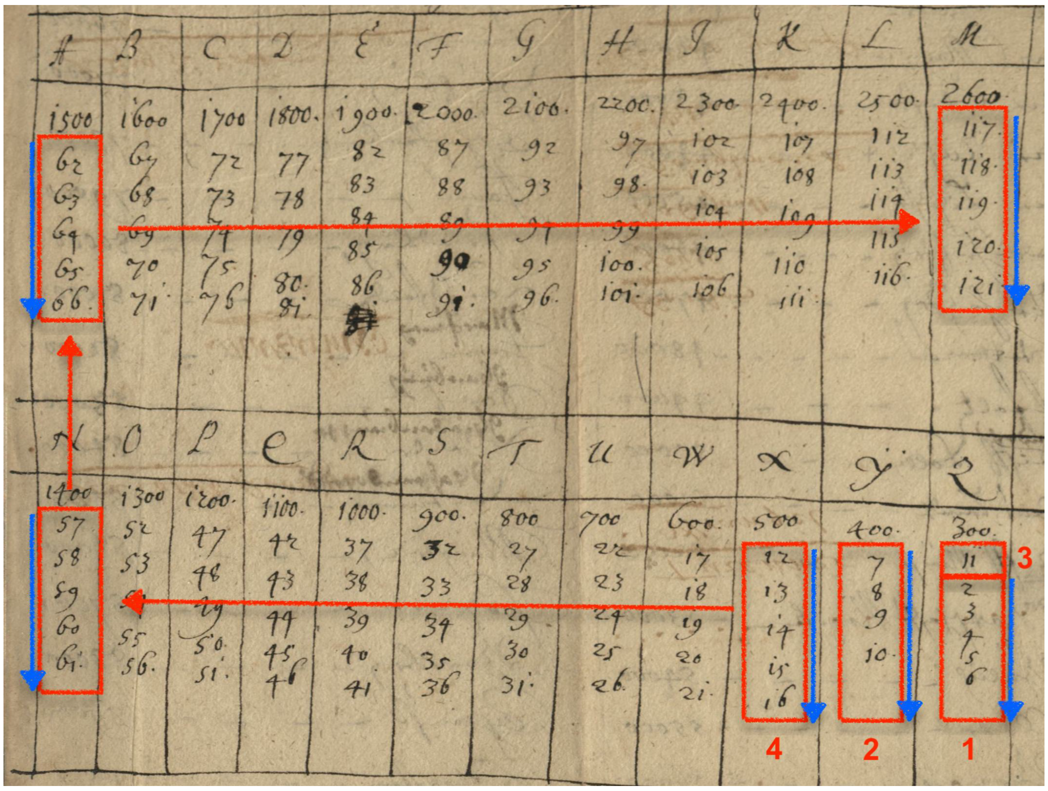
Chapter 1

Theoretical Framework

This chapter presents the theoretical framework of the different encryption algorithms AES, RSA and ECC. The theoretical foundations of each of these algorithms will be explored, including key concepts, structure, operation and cryptographic properties. The in-depth analysis of each algorithm will serve as a basis for the development of the algorithms and their respective graphical interface.

1.1 Fundamentals of Encryption

Among the simplest thoughts or ideas that may arise when hearing the word cryptography are providing secure access to the web, the German enigma machine, message encryption or simply writing a message in a way that an unauthorized person can't understand it. These terms would imply that cryptography is linked to the current technological era. However, according to [37] evidence has been found that the bases of cryptography have existed for thousands of years in the past with the discovery of "*secret writing*". In addition, to the fact that in specific years from 1400 and for the next 450 years the simplest encryption method was used, being the substitution method through a fixed table where the original letter was replaced by another. It should be noted that this process was carried out with the help of a nomenclator [38], that consisted of several encryption systems used together during the message encryption process. This tool consisted mainly of homophonic or monoalphabetic substitutions. Here, these substitutions were combined with a trigram or bigram substitution. An example of this can be seen with the homophonic substitution in the Figure 1.1.



A	B	C	D	E	F	G	H	I	K	L	M
1500	1600	1700	1800	1900	2000	2100	2200	2300	2400	2500	2600
62	67	72	77	82	87	92	97	102	107	112	117
63	68	73	78	83	88	93	98	103	108	113	118
64	69	74	79	84	89	94	99	104	109	114	119
65	70	75	80	85	90	95	100	105	110	115	120
66	71	76	81	86	91	96	101	106	111	116	121

N	O	P	Q	R	S	T	U	W	X	Y	Z
1400	1300	1200	1100	1000	900	800	700	600	500	400	300
57	52	47	42	37	32	27	22	17	12	7	11
58	53	48	43	38	33	28	23	18	13	8	2
59	[illegible]	49	44	39	34	29	24	19	14	9	3
60	55	50	45	40	35	30	25	20	15	10	4
61	56	51	46	41	36	31	26	21	16		5
											6

Figure 1.1: Part of the nomenclator showing the substitution process [1].

Over the years, encryption methods evolved together with the nomenclature, causing the number of pages to increase [1]. This encryption method was very famous, extending its use from the fourteenth to the nineteenth century [39]. According to [40], another method that was also widely used in its time was the transposition encryption method, being a method that consisted of changing the position order of each letter in a word. For example, the matrix below shows the phrase "ricos caramelos" organized in the form of a matrix and reading the columns from top to bottom and from left to right gives the encrypted message "rceialcrooassm".

$$\begin{array}{ccccc} r & i & c & o & s \\ c & a & r & a & m \\ e & l & o & s & \end{array}$$

On the other hand, around 1900 BC in ancient Egypt, scribes used hieroglyphics in a particular way, fact being reflected in several monuments [41, 37]. Similarly, in [42], the encryption method that the Greeks had called transposition with scytale is mentioned, where the message was written on a strip of cloth rolled on a stick. Indeed, when stretched again reflected a combination of letters very different from the original. Likewise, other ancient encryption methods can be mentioned, one of them is the Caesar cipher, created by the Romans. This method consisted of shifting the letters a predetermined number of positions [43].

Encryption methods have been perfected and reinvented until reaching the current era. One of the most important inventions in the field of cryptography was considered very important during World War II [44]. This invention was called the Enigma machine, see Figure 1.2, being created by Arthur Scherbius between the years 1920 and 1930 to be used by the German army during military communications. This machine did not allow messages to be easily deciphered due to its innovative structure, making it one of the first encryption machines. So, its evolution continued until the end of 1940 because of the Germans continued to add more complexity to the machine [45]. Taking this historical fact into account, according to [46], the Enigma machine was an important invention that paved the way for modern cryptography, since the theoretical foundations used for its creation. Every theoretical concept evolved in its creation have also been used for the development of new encryption and decryption techniques. Among them, we have:modern symmetric encryption (using more complex substitutions and permutations), block ciphers, and stream cryptography [47].

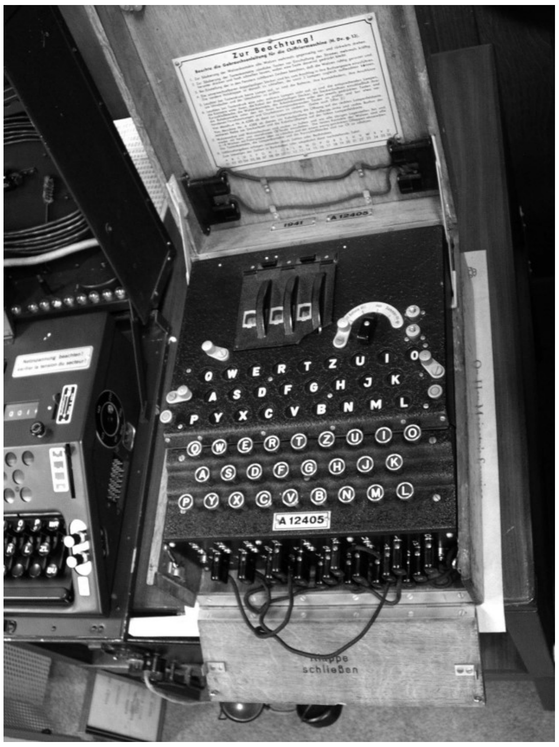


Figure 1.2: Enigma machine used during World War II [2].

1.2 Encyption types

1.2.1 Symmetric encryption

This form of encryption has been highly refined with the arrival of computing, being a method that uses a unique key to both encrypt and decrypt the message. This key must be possessed by the sender and the receiver, otherwise communication would not be possible. However, in [48] it is mentioned that in order for the message not to be intercepted while it is being sent, the communication and the sending of the key must occur within a secure channel. In Figure 1.3, we can see a diagram that represents how the encryption and decryption process with a shared key works, working as a measure to counteract the leakage of information. A real-life example could be that a company is planning to launch a new mobile application and the details of the application are being sent to the developers. In this case, it would be ideal if the information is not discovered by other companies that represent the competition.

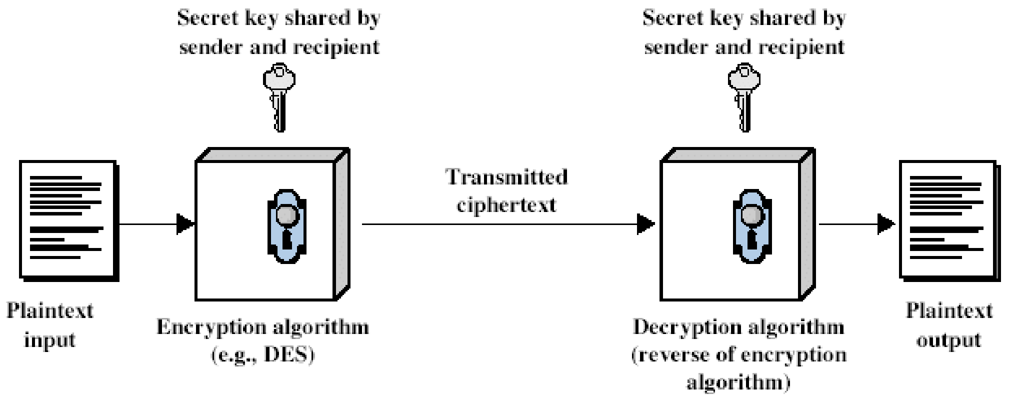

Secret key shared by sender and recipient
Secret key shared by sender and recipient
Transmitted ciphertext
Plaintext input
Encryption algorithm (e.g., DES)
Decryption algorithm (reverse of encryption algorithm)
Plaintext output

Figure 1.3: Symmetric encryption scheme [3].

This form of encryption was very important and had Data Encryption Standard (DES) algorithms as its representative. This happened after being selected by the National Institute of Standards and Technology (NIST), agency focused on research and innovation [49] that have enabled both specialists and amateurs to present their innovative ideas related to the field of technological innovation, science and industry. In the field of data security, encryption methods have been presented between the 1970s and 1990s. These algorithms have been subjected to intense testing and many attacks, such as obtaining metrics that evaluate the encryption and decryption speed, the generation time of each key and the resistance to various attacks to stress the algorithm. These types of evaluations were carried out to verify their robustness and give the green light for their use in both industry and commerce [50]. One of the most well-known encryption algorithms was selected from this type of competition, being the Data Encryption Standard (DES) algorithm, presented by IBM in 1975 [51]. Figure 1.4 shows the structure of the DES system, being part of the symmetric algorithms. Regarding its operation, it starts with an initial permutation of the plaintext P, splitting it into two 32-bit blocks ($L0$ and $R0$). It then performs 16 rounds of operations including a function F, XOR operations, and key shifts. In each round, the right half is processed with the round key and combined with the left half. In parallel, the key K is processed to generate subkeys. Finally, a 32-bit swap and reverse permutation are applied to obtain the ciphertext E.

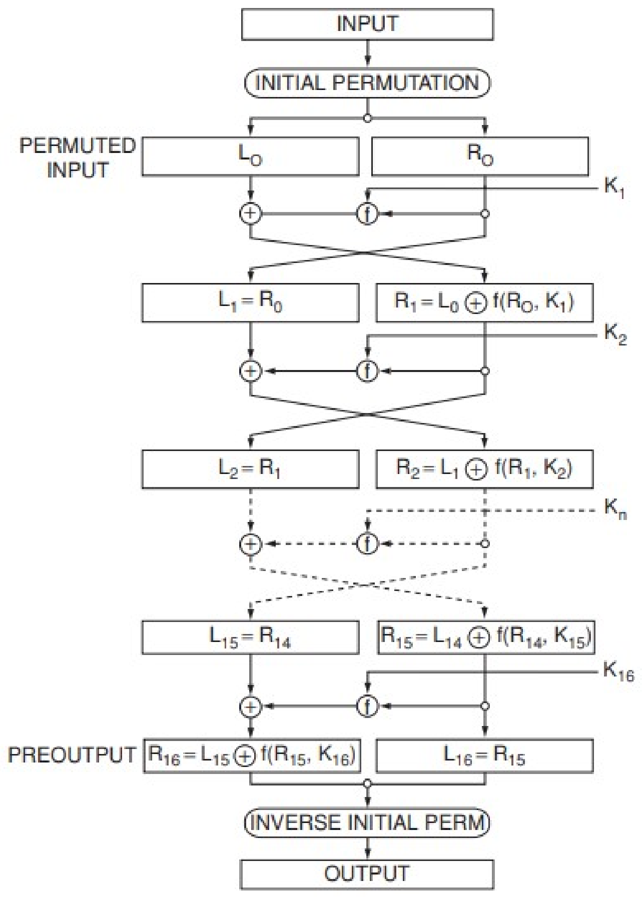


Figure 1.4: DES encryption process [4].

This type of encryption was widely used, but nowadays it is very easy to crack due to its simple architecture and its limited key size (56 bits during the process but officially 64 bits were taken, but 8 bits were used for parity and to avoid transmission errors at the bit level [52]). At the end of the 1990s another competition was held to choose the next algorithm capable of protecting commercial data in the United States [53, 54]. This is because the algorithm used at the time was already considered old and insecure to continue using it for any longer. This way, according to [55], among the algorithms that passed the first phase of the competition were: RC6 from RSA Labs, MARS from IBM, Serpent, Rijnadel, and Twofish. In the following phases, emphasis was placed on the compatibility of the algorithms for different processors, such as 32-bit processors (Pentium Pro/II, the Pentium) and 64-bit CPUs. Likewise, it was verified whether such codes could be migrated from one programming language to another, for example if the code was originally written in C, it was checked whether it could be implemented in Java. Another stage evaluated the performance of the algorithms taking into account the use of certain Smart cards [56]. These cards contained microprocessors to store any information generated during and after executing the codes. Among the types of Smart cards used in this phase were: Motorola 6805 [57], Intel 8051 [58], and ARM [59]. After all these tests and others that verified their security, the two best algorithms were chosen, being the Twofish cipher and the Rijndael cipher, but at the end the Rijndael algorithm emerged as the winner. In this way, a transition occurs in the way of protecting data since during the competition in 1999 the DES algorithm was deciphered in less than 24 hours, after having been used for more than 20 years. However, according to [60], DES encryption was not completely eliminated but was relegated to use only in legacy systems [60], that is, old technologies or computer programs that have not been updated.

Advanced Encryption Standard

One of the algorithms that will be analyzed in this research is the advanced encryption standard (AES) algorithm. The original name of this algorithm was Rijndel, a comnbination of the names of its creators Joan Daeman and Vincent Rijmen. Unfortunately, it was not popular for commercialization, and it was renamed as AES [61]. By October 2, 2000 the Rijndael algorithm was officially named AES by the NIST and on November 26, 2001 it was formally approved as a standard security system in the U.S [62].

This algorithm belongs to the group of symmetric encryption because it uses the same secret key for both encryption and decryption [63]. AES encryption is fundamentally based on applying substitution and permutation network (SPN) techniques [64]. This include a series of mathematical operations in the block cipher algorithm, dividing the input text into different blocks of 128 bits (16 bytes) to be processed, and represented as a 4x4 bytes matrix [65].

On the other hand, the key size used for this algorithm has three variants, being 128, 192, and 256 bits [66]. However, only the 128-bit size is considered the standard for AES. As seen in Figure 1.5, an input data x of 128 bits is entered and the different key sizes k that can be used, but always providing an output y of 128 bits size. In addition, in [67] it is mentioned that an important aspect of AES encryption is the number of rounds, since it depends on the length of the key used, varying from a number of rounds of 10, 12, 14 for a key size of 128, 192, 256 bits, respectively.

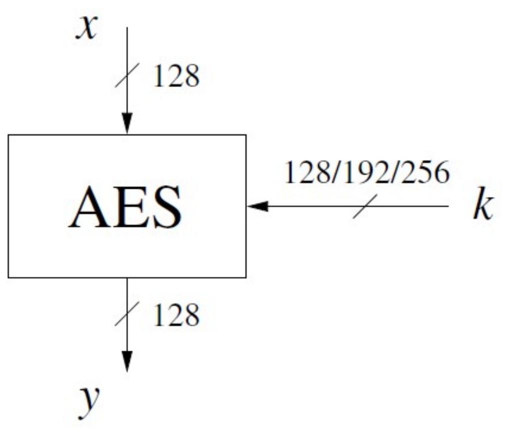


Figure 1.5: Different key sizes that AES algorithm can accept [5].

Mathematical foundations of the AES encryption algorithm

Before giving an explanation of each process and arithmetic operation involved during the execution of AES, it is necessary to know some important mathematical concepts. These are important for most of the encryption methods discussed in this research.

Galois Field: Also called finite field [68], is the most important concept in most of the operations performed in the AES algorithm and is represented as $GF()$. It consists of a certain number of elements where various arithmetic operations can be performed. However, there are two key concepts to fully understand Galois fields. These are:

- **Group:** A group is a basic algebraic structure consisting of a set of elements together with a binary operation, which combines two elements of that set to produce another element of the same set. In addition, there must also be a special element that does not change anything, such as 0 in addition, and each element must have its inverse (such as $+3$ and -3) [69].

- **Field:** A field is a more advanced mathematical structure than a group. It consists of a set of elements and two binary operations, commonly known as addition and multiplication. Also, it includes all the rules that must be followed within a group. Furthermore, it allows for extra operation, such as multiplication. This operation, together with those offered by the group, becomes the basis for applying all the transformations involved in AES encryption. [70].

Prime Fields: According to [71], prime fields are called $GF(2^8)$ in cryptography, being an extension of the prime field $GF(2)$. This extension allows much more complex operations to be performed in $GF(2^8)$ and with larger values than in $GF(2)$. In other words, the GF is an extension of the finite field concept with p^n elements, where $(n > 1)$, and p prime. This structure is the one used by AES; with $n = 8$; and $p = 2$ where $n = 8$ indicates 8-bit words.

Modular operations

- **Addition and Subtraction:** By getting together the above concepts, addition and subtraction operations can be performed according to Eq. 1.1 and Eq. 1.2. Where $A(x), B(x),$ and $C(x)$ represent polynomials such that if we have an 8-bit input $A(x) = 11001010$, represented as a polynomial we would have $A(x) = x^7 + x^6 + x^3 + x$ and the same applies to $B(x)$ and $C(x)$. In addition, we have the summation symbol that represents the sum of polynomials that is iterated $m = 8$ times. Here, m represents the number of bits. Finally, we have C_i that represents the process to calculate the coefficients of the result $C(x)$ where the coefficients of $A(x)$ and $B(x)$ are taken and the sum of both is applied modulo 2. This means that an XOR operation is performed at the bit level.

$$C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_i x^i, \quad c_i \equiv a_i + b_i \mod 2 \tag{1.1}$$

$$C(x) = A(x) - B(x) = \sum_{i=0}^{m-1} c_i x^i, \quad c_i \equiv a_i - b_i \equiv a_i + b_i \mod 2 \tag{1.2}$$

- **Multiplication:** It is an operation that must be applied when working with polynomials. First, we multiply both polynomials normally. Second, to adjust or reduce the size of the result to the rules of the field $GF(2^m)$, we must apply an “irreducible polynomial”. This type of polynomial states that all the results obtained within the field must be less than a certain limit [72], where this limit is defined by the prime field. The way to express this operation is seen in Eq. 1.3, where we first multiply $A(x)$ and $B(x)$ in the normal way, and the result could be of higher degree and larger than normal. To solve that, the irreducible polynomial $P(x)$ is applied to reduce the previous result and obtain $C(x)$. In this way $C(x)$ would comply with the rules of the field $GF(2^m)$, ensuring that each calculation performed afterwards remains within the established limit.

$$C(x) \equiv A(x) \cdot B(x) \mod P(x) \tag{1.3}$$

- **Inverse:** This concept can be explained with an informal example. We have a machine that does calculations and we need to find a piece that when combined with another piece, gives us the value 1. It works in a similar way in cryptography, since we must look for a polynomial $B(x)$ that when multiplied by another polynomial $A(x)$ according to the Eq. 1.4. In turn, said result must be reduced with the irreducible polynomial $P(x)$ and the result should give us the value 1. The way to find $B(x)$ is by applying the extended Euclidean algorithm [73], where we divide the polynomial until the remainder is 1. Thus, the polynomial associated with this result is called the inverse of $A(x)$ [74].

$$A(x) \cdot B(x) \equiv 1 \mod P(x) \tag{1.4}$$

Alternatively, this process can be done using set inverse tables or inverse S-boxes, as shown in Table 1.1.

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

Table 1.1: Multiplicative inverse table in $GF(2^8)$.

Functionality of AES

In this section, the relationship between the mathematical concepts explained above and the structure of the AES algorithm will be explained. In addition, we will explain the different layers that belong to AES and the respective order in which they must be applied will. This applies to both the encryption and decryption of information. Figure 1.6 shows how these layers are distributed and how they are applied as the encryption rounds progress.

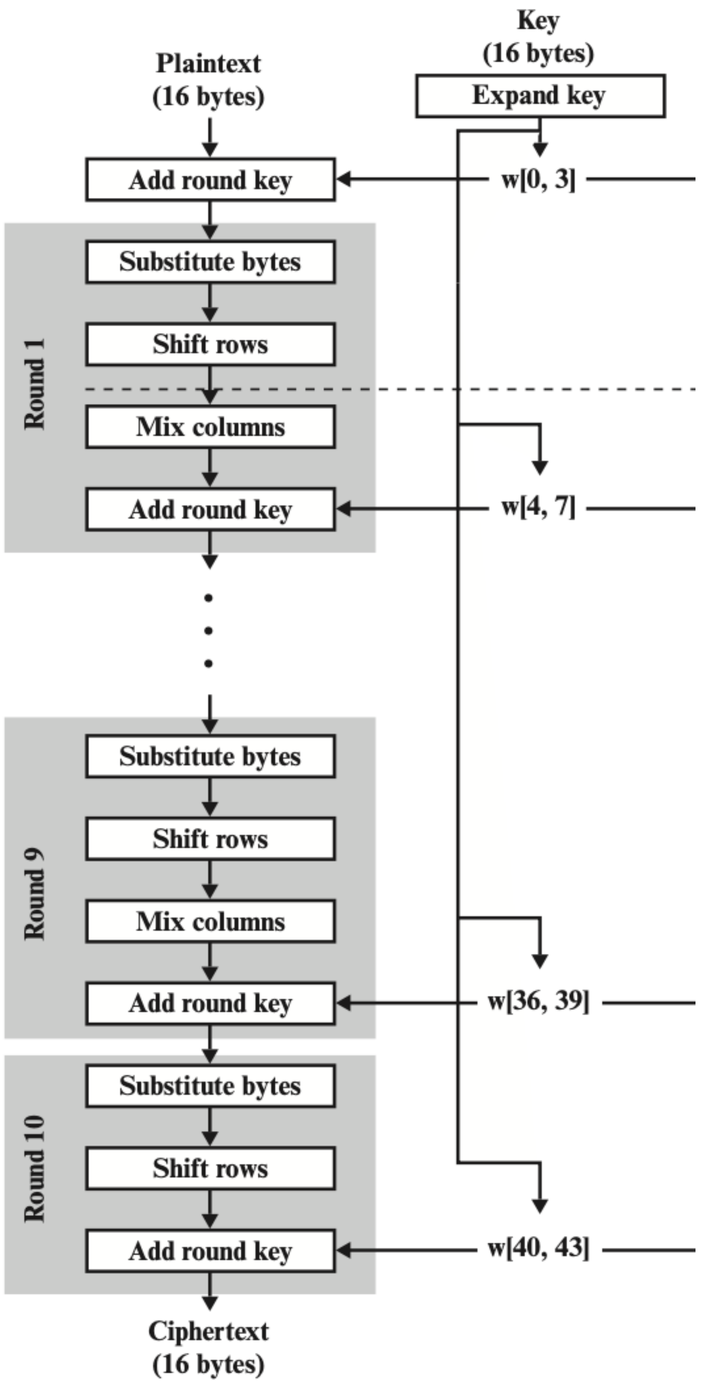


Figure 1.6: AES algorithm layers [6].

Each one the layers seen in Figure 1.6 and the respective mathematical operations performed on them are detailed as follows:

- **Key addition layer:** For this layer, a key derived from the original key through the key Schedule process is used, in addition to having a size of 128 bits and is called a round key or subkey.
  - **Key Schedule:** For the key whitening process, the original key of the size of bits accepted by AES is needed to generate the subsequent subkeys. The process is carried out by XOR operations [75] with each subkey generated, both at the beginning and at the end of the encryption process. This operation causes the need for $n + 1$ subkeys in addition to generating an extra layer of security. This process is

known as key whitening [76], referring to the combination of the data with portions of the key through logical operations. For example, if a 128-bit main key is used, the number of rounds will be equal to 10 and the number of subkeys needed will be 11 (one for the initial key whitening and the rest of the subkeys for each of the encryption rounds).

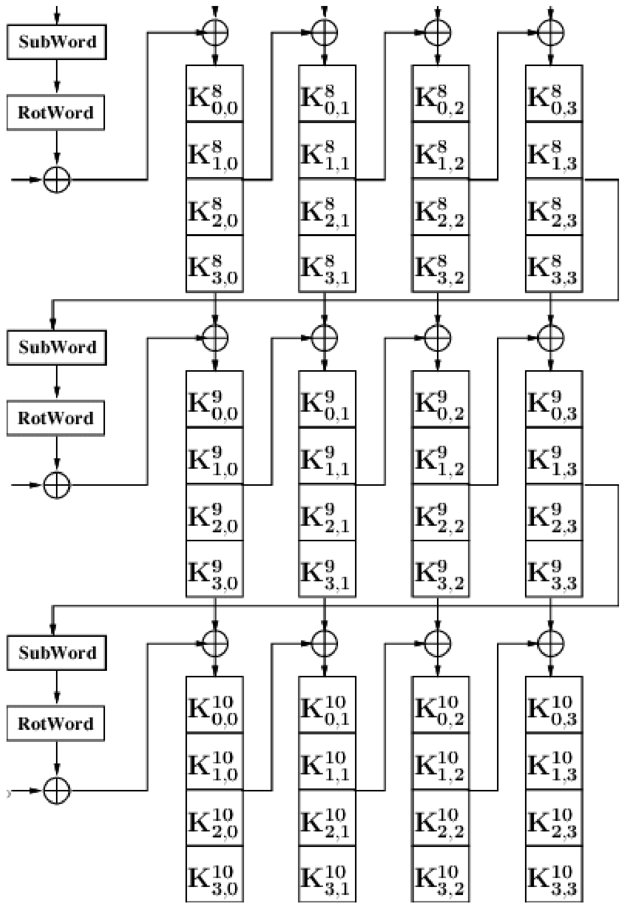

SubWord
RotWord
$K^8_{0,0}$ $K^8_{0,1}$ $K^8_{0,2}$ $K^8_{0,3}$
$K^8_{1,0}$ $K^8_{1,1}$ $K^8_{1,2}$ $K^8_{1,3}$
$K^8_{2,0}$ $K^8_{2,1}$ $K^8_{2,2}$ $K^8_{2,3}$
$K^8_{3,0}$ $K^8_{3,1}$ $K^8_{3,2}$ $K^8_{3,3}$
SubWord
RotWord
$K^9_{0,0}$ $K^9_{0,1}$ $K^9_{0,2}$ $K^9_{0,3}$
$K^9_{1,0}$ $K^9_{1,1}$ $K^9_{1,2}$ $K^9_{1,3}$
$K^9_{2,0}$ $K^9_{2,1}$ $K^9_{2,2}$ $K^9_{2,3}$
$K^9_{3,0}$ $K^9_{3,1}$ $K^9_{3,2}$ $K^9_{3,3}$
SubWord
RotWord
$K^{10}_{0,0}$ $K^{10}_{0,1}$ $K^{10}_{0,2}$ $K^{10}_{0,3}$
$K^{10}_{1,0}$ $K^{10}_{1,1}$ $K^{10}_{1,2}$ $K^{10}_{1,3}$
$K^{10}_{2,0}$ $K^{10}_{2,1}$ $K^{10}_{2,2}$ $K^{10}_{2,3}$
$K^{10}_{3,0}$ $K^{10}_{3,1}$ $K^{10}_{3,2}$ $K^{10}_{3,3}$

Figure 1.7: Key schedule for 128 bits in AES [7].

For this research, we will only focus on 128-bit key whitening. This is reflected in Figure 1.7, where we can visualize the process of creating each subkey from the last three rounds of AES encryption. Entering all the subkeys and dividing them into different parts, called "words". Here, each word represents 32 bits and is represented by the letter W. This is represented as $W[0] \mid\mid W[1] \mid\mid W[2] \mid\mid W[3]$ where $\mid\mid$ is the concatenation that joins the four words [77]. The process applies byte substitution, rotation, and XOR operations with round constants called $Rcon$. During this process, the $Rcon$ is a table of constants that is applied to each "word" of the expanded key by using the XOR operation. This result is combined again with the previous "word", being necessary for the following stages of key expansion. To sum up, the first "word" of each round is created by combining these elements with the corresponding word from the previous round, while the next three words are obtained by successive XOR with previous words. From there the rest of the subkeys will be created according to Eq. 1.5, where $W[4_i]$ is the element on the left and $i = 1, \ldots, 10$ is the number of rounds to be executed.

$$W[4_i] = W[4(i-1)] + g(W[4i-1]) \tag{1.5}$$

Here, $W[4_i]$ is the result of the XOR operation of the previous word $W[4(i-1)]$ and a nonlinear function $g(W[4i-1])$ that takes as a parameter the word $W[4i-1]$. This function $g()$ is seen on the right of Figure 1.7 and is essential to avoid symmetry between braces and to provide nonlinearity during this scheduling. The function $g()$ performs both S-Box substitution operations, as well as adding to the round coefficient that in turn belongs to the Galois field. It is worth noting that this round coefficient is only added to the left byte and is different depending on the round of execution:

$$RC[1] = x_0 = (00000001)_2, \ldots, RC[10] = x_9 = (00110110)_2$$

The rest of the keys are calculated recursively according to Eq. 1.6, where $j = 0, 1, 2, 3$ represents the number of words into which the key was divided.

$$W[4i-j] = W[4i+j-1] + W[4(i-1)+j] \tag{1.6}$$

– **Byte substitution layer:** According to Figure 1.7, the first step to follow after key scheduling is the respective application of the byte substitution layer. In simple words, it is a process that takes as reference a fixed substitution table or S-box (see Table 1.2), and depending on the row and column coordinates the original byte will be substituted according to the S-box [78]. This process can be denoted as S-box(A$[i][j]$) $=$ B$[i][j]$.

		Y															
		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
X	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

Table 1.2: S-box for byte substitution layer.

To obtain the value corresponding to the S-box table we have to obtain the multiplicative inverse. This process is repeated for each byte of the state matrix A_i. Then, the value of each byte is transformed to its bit representation and the respective arithmetic operation is applied on the fixed inverse table to obtain its value. For example, using the inverse table, we have

$$A_i = (11000010)_2 = (C2)_{\text{hex}} \rightarrow A_i^{-1} = B_i' = (2F)_{\text{hex}} = (00101111)_2$$

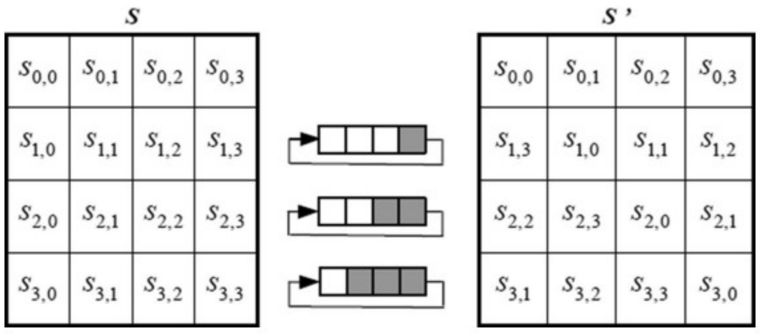
In this way, the result is taken as a vector and then the affine mapping is applied, that is, the operation seen in Eq. 1.7 is applied either for $b_0, \dots, b_7$ and $b'_0, \dots, b'_7$ where M and m represent a fixed matrix and a fixed vector, respectively.

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}}_{M} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \underbrace{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}}_{m} \text{mod} 2. \tag{1.7}$$

This process may take longer to perform, for this reason, to optimize the process only the S-box table is used by taking the hexadecimal representation of the input byte.

– **Diffusion layer:** This layer is designed to prevent any influence that one bit may have on another during the encryption process. To do this, two operations are performed at the matrix level.

ShiftRows: This operation takes each row of the state matrix and swaps the byte positions according to Figure 1.8.



S
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
S'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,3}$ $s_{1,0}$ $s_{1,1}$ $s_{1,2}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,1}$ $s_{3,2}$ $s_{3,3}$ $s_{3,0}$

Figure 1.8: Shift row scheme [8].

MixColumns: After finishing the ShiftRow, this step is applied where the columns of the state matrix are taken and mixed. To do this, a 16-byte column is taken as input and a 16-byte output column is obtained according to MixColumn$(B) = C$. Next, 4-byte vectors are taken, and algebraic operations are applied to them, that is, the vector is multiplied by a constant matrix where each multiplication and addition that is performed must be within the Galois fields. This process is repeated every four bytes of the byte column B. Once these processes are carried out in each layer and for each round for a certain number of times, the encrypted text is obtained.

Decryption process

In simple words, the decryption process is the application of the inverse of the same steps and operations used to encrypt [79]. In Figure 1.9, the steps for this process are explained below:

1. The AddRoundKey is applied taking the encrypted text and the key used for the end of the encryp-

tion round and an XOR operation is applied.

2. The inverse SubBytes is applied using the inverse version of the S-Box table seen in Table 1.3. In this way, said substitution made to encrypt the text is reversed.
3. we apply the inverse of ShiftRows, moving the bytes to the right to undo the change made in the encryption.
4. We apply the inverse of MixColumns by using the $g()$ function from $GF(2^8)$, on each column of the state matrix. Having both parameters $g()$ and the column, we apply the multiplication seen in Figure 1.10.

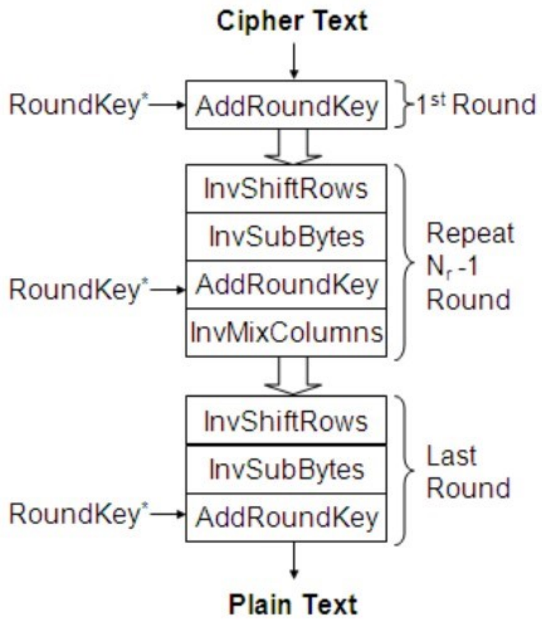


Figure 1.9: AES decryption scheme [9].

$$\tilde{S}' = \tilde{A} \cdot \tilde{S}$$

$$\begin{bmatrix} \tilde{s}'_{0,0} & \tilde{s}'_{0,1} & \tilde{s}'_{0,2} & \tilde{s}'_{0,3} \\ \tilde{s}'_{1,0} & \tilde{s}'_{1,1} & \tilde{s}'_{1,2} & \tilde{s}'_{1,3} \\ \tilde{s}'_{2,0} & \tilde{s}'_{2,1} & \tilde{s}'_{2,2} & \tilde{s}'_{2,3} \\ \tilde{s}'_{3,0} & \tilde{s}'_{3,1} & \tilde{s}'_{3,2} & \tilde{s}'_{3,3} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} \tilde{s}_{0,0} & \tilde{s}_{0,1} & \tilde{s}_{0,2} & \tilde{s}_{0,3} \\ \tilde{s}_{1,0} & \tilde{s}_{1,1} & \tilde{s}_{1,2} & \tilde{s}_{1,3} \\ \tilde{s}_{2,0} & \tilde{s}_{2,1} & \tilde{s}_{2,2} & \tilde{s}_{2,3} \\ \tilde{s}_{3,0} & \tilde{s}_{3,1} & \tilde{s}_{3,2} & \tilde{s}_{3,3} \end{bmatrix}$$

Figure 1.10: Scheme for the inverse MixColumn process.

1.2.2 Asymmetric encryption

Once we know the algorithm used by the symmetric encryption type, it is time to talk about another type of encryption that uses two types of keys to encrypt information. In this case, asymmetric encryption creates two keys called public key and private key. The first one is used to encrypt the information and the second key is used to decrypt the information. In this way, more security is provided during communication between users. As can be seen in Figure 1.11, a public key is used to convert the original message into an unreadable format. Nonetheless, the private key that is kept secret, is used to return it to its original form.

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

Table 1.3: Inverse S-Box for AES decryption.

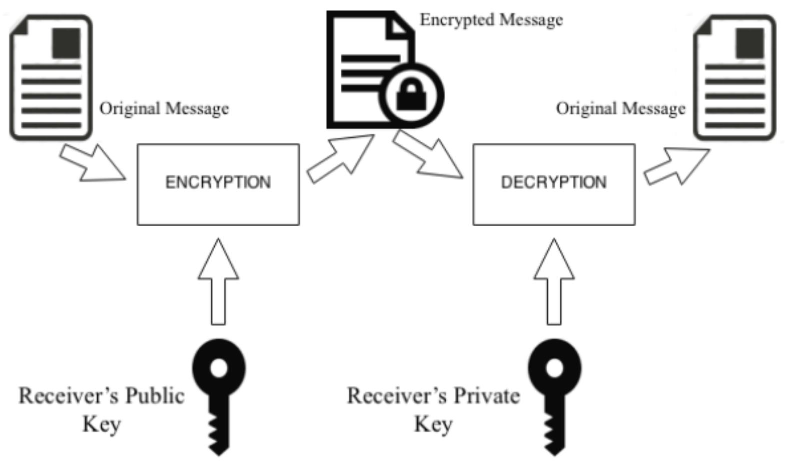


Figure 1.11: Asymmetric encryption process [10].

It should be noted that this type of encryption is mainly used to encrypt small amounts of data, such as session keys or to digitally sign messages [80]. If large volumes of data are to be encrypted, asymmetric encryption must be combined with other types of ciphers, being symmetric ciphers. The reason for this is explained in [48], where it is clarified that the speed of this asymmetric algorithm is very low compared to the speed of symmetric ciphers, in addition to requiring higher consumption of computational resources. In this case, the process is summarized in using the asymmetric cipher to encrypt the session key or shared key and using it by applying a symmetric cipher to encrypt the data [81]. Below, two of the main algorithms that apply this concept and that will be used in this research are detailed.

Rivest-Shamir-Adleman algorithm

The Rivest-Shamir-Adleman algorithm (RSA) is a widely used type of asymmetric encryption worldwide, developed in 1977 at the Massachusetts Institute of Technology (MIT), whose name comes from the first initials of its creators: Ron Rivest, Adi Shamir and Leonard Adleman[82]. This algorithm is characterized by using exponentiation to perform encryption and decryption [83]. Furthermore, this algorithm emerged as a strategy to establish secure communications without the need for the parties to first exchange private keys. [84] mentions that, the cryptographic systems available at that time required the parties to meet privately to exchange private keys. A fact that limited their practicality for secure communications on a large scale or over long distances. For this reason, Shamir, Adleman, and Rivest addressed the mathematical problem of factoring large numbers [84].

In [85], the mathematical premise was to use prime numbers to solve the problem easily. These prime numbers must be very large and then multiplied. Unfortunately, it is very difficult to solve it in the other direction, that is, by factoring the multiplication product into its original prime factors. According to [86], this fact causes the impact of the RSA algorithm on cryptography and computer security to be immediate. On the other hand, the RSA algorithm not only allows for secure data encryption, but also the creation of digital signatures that guarantee the authenticity and integrity of messages, giving it exclusivity over the rest of [87] algorithms. Thus, according to [88], the door is opened to a wide range of applications, such as allowing secure communications via the Internet, carrying out financial transactions and allowing secure data storage.

Unfortunately, although the advantages it offers are notable, it has a disadvantage that is linked to its own structure and operations it uses. The use of large prime numbers, added to the fact that it uses exponentiation as the main operation, make it a very slow algorithm compared to ciphers such as the AES [89]. This has greatly limited its use, using it in cases where it is necessary to secure the exchange of keys to perform a symmetrical [90] encryption. Despite this fact, [91] mentions that the RSA algorithm has managed to maintain its relevance, highlighting its robustness and the security it offers.

The process followed by the RSA algorithm is given by the following steps:

- Generation of prime numbers p and q.
- Calculate $n = p \cdot q$ and $\phi(n) = (p-1)(q-1)$
- Select a value e that is coprime to $\phi(n)$
- Calculation of d such that, $d \cdot e = 1 \mod \phi(n)$
- Define both keys:
  - Public key (e, n)
  - Private key (d, n)
- Apply the encryption and encryption equations

Keys generation: Depending on the desired security for the algorithm and considering the computational resources, the desired size for the prime numbers p and q that must be chosen or found is defined [92]. To do this, each number chosen or generated in the code must pass a primality test. However, performing this type of test for each number chosen may be suboptimal if the size of the number is too long [93]. Considering that the complexity of each prime number increases sequentially as the number increases in size [94]. To avoid this type of setback, certain methods have been considered that allow verifying whether a number is prime without compromising too much the effectiveness of the algorithm [95].

Algorithm 1: Pseudocode Fermat Theorem

```
Input: $n$ is a positive odd number.
Output: $p$ and $q$ are two prime numbers such that $n = pq$.
1 $x = \lfloor\sqrt{n}\rfloor + 1$;
2 $y = x^2 - n$;
3 while ($y$ is not a perfect square) do
4     $x = x + 1$;
5     $y = x^2 - n$;
6 $p = x + \sqrt{y}$;
7 $q = x - \sqrt{y}$;
```

For this reason, in the Algorithm 1, we can see the pseudocode that represents the Fermat primality test. This consists of verifying whether a number is prime if the equality of 1.8 is fulfilled.

$$a^{\widetilde{p}-1} \not\equiv 1 \tag{1.8}$$

Here, a is a random value that is coprime such that $a^{n-1} \not\equiv 1$ taken from the interval $(1 < a < n-1)$ where n is the value that is to be checked if it is prime [96]. Next, the operation of Eq. 1.8 is checked and if the result is 1 the number n could be considered prime. It should be noted that this process must be carried out with several values for a, since if any of them after applying 1.8 is not 1 the number is considered composite [97].

According to [98], Carmichael numbers can fool this test, considered composite numbers that can behave as prime numbers even if several values are taken for a.

Another method used to check that a number is prime is the Miller-Rabin primality test [17]. To apply this method, the number chosen for the test is treated differently if it is even or odd. If $\widetilde{p}$ is an odd prime value, it is decomposed according to Eq. 1.9, where r is odd.

$$\widetilde{p} - 1 = 2^u r \tag{1.9}$$

Then, a value a must be chosen within the interval $(1 < a < n-1)$ and that satisfies the following equalities:

- $a^r \not\equiv 1 \mod p$
- $a^{r2^j} \not\equiv \widetilde{p} - 1 \mod \widetilde{p}$

If the value satisfies the equalities such that $j = 0, 1, \ldots, u-1$ the value $\widetilde{p}$ is considered a composite number, otherwise it is a prime number [99]. This process is reflected in Algorithm 2, where it shows the Miller-Rabin test and how it applies the Square-and-Multiply technique [100] to calculate modular powers. Here, the value r is taken in its binary form and is evaluated at the bit level from the most significant to the least significant. At the beginning, the result is set to 1. For each bit of r, the current value is squared and, if the resulting bit is 1, an additional "multiplication" by the base is performed. In this way, according to [101], the process of alternating between squares and multiplications allows to compute $a^r \mod \widetilde{p}$ efficiently, and repeating the iterations according the value s selected.

Algorithm 2: Pseudocode Miller-Rabin test

```
Input: Integer $s$ and $\widetilde{p}$ with $\widetilde{p} - 1 = 2^u r$
Output: Confirmation if $\widetilde{p}$ is prime or not
1  for $i = 1, \dots, s$ do
2      select $a \in \{2, 3, 4, \dots, \widetilde{p} - 2\}$;
3      $z = a^r \mod \widetilde{p}$;
4      if $z \not\equiv 1$ and $z \not\equiv \widetilde{p} - 1$ then
5          for $j = 1, 2, \dots, u - 1$ do
6              $z = z^2 \mod \widetilde{p}$;
7              if $z = 1$ then
8                  "$\widetilde{p}$ is composite";
9          if $z \neq \widetilde{p} - 1$ then
10             "$\widetilde{p}$ is composite";
11 "$\widetilde{p}$ is prime";
```

After finding the two corresponding prime values p and q, we have to follow the steps from the beginning of this section. Here, the expression that defines the public key $k_{pub} = (n, e)$ and the private key $k_{priv} = (d, n)$ can be seen. In addition, the rest of the parameters $n = p.q$ are calculated, the value of the Euler function $\phi(n) = (p-1)(q-1)$ [102], also a value e is chosen such that $(1 < e < \phi(n))$ and that is coprime to $\phi(n)$ [103], using the formula $gcd(e, \phi(n)) = 1$ [104]. As a final part, the private key d is calculated using the multiplicative inverse $d \cdot e = 1 \mod \phi(n)$ [105].

Encryption and decryption process

In the Equations 1.10 and 1.11, we can see the formulas that are applied to the input data to perform the encryption and decryption, respectively [106]. Where e is the public key, d is the private key, C represents the original message and M is the encrypted message. The last two parameters are transformed into an integer and passed to the respective operation exponentiation and modulo [107]. However, according to [108], handling such large numbers is very complicated to do directly. For this reason, alternative techniques are used, so we can reduce the difficulty when performing the exponentiation of numbers.

$$C = M^e \mod n \tag{1.10}$$

$$M = C^d \mod n \tag{1.11}$$

Fast exponentiation method

This method can be seen in Algorithm 3, where it is represented the pseudocode for the Square-and-Multiply for Modular Exponentiation [109] method. This process has already been used for the creation of public and private keys. In a simple way, this code the bits that conform H and start reading it from left to right. Applying square or multiplication according the current bit in each iteration. The advantage of applying this method is to avoid direct calculations with very large numbers and to improve efficiency compared to direct exponentiation. Besides, it is effective even if the key size is large, be it 1024 bits, 2048 bits, and 3072 bits. These are the sizes used in this research.

Algorithm 3: Pseudocode Square-and-Multiply

```
Input: Selected value $x$;
exponent $H = \sum_{i=0}^{t} h_i 2^i$ where $h_t = 1$ and $h_i \in 0$;
Output: $x^H \mod n$
1 $r = x$;
2 for $i = t - 1, \dots, 0$ do
3     $r = r^2 \mod n$;
4     if $h_i = 1$ then
5         $r = r \cdot x \mod n$;
6 $r$;
```

Another method used that can improve the efficiency of RSA is the Chinese Remainder Theorem (CRT) method [110]. Its objective in a nutshell is to avoid performing arithmetic operations with a long value of n and only take its components p and q to perform exponentiation separately to put those results together at the end [111].

Elliptic Curve algorithm

The Elliptic Curve cryptography (ECC) is considered the youngest among the encryption algorithms discussed in this research. This is because it was created in the 1980s and is part of the group of asymmetric ciphers [112]. The reason why it is taken into account is that it provides a level of security similar to or even better than the RSA algorithm [113]. This method takes key sizes such as 160 bits or 256 bits [34], being much smaller than those handled by RSA, ranging from 1024 bits to 3072 bits.

This algorithm was developed from research in public key cryptography that sought to create more efficient and secure systems than those existing in the 1980s [114]. For this reason, the idea of implementing the concept of elliptic curves to cryptography was introduced by Victor Miller and Neal Koblitz independently in 1985 [115]. This idea took as a reference the mathematical properties of elliptic curves over finite fields, a concept already mentioned for the AES algorithm. In addition, to being based on the generalized discrete logarithm problem that allows certain protocols such as the one proposed by Diffie-Hellman for key exchange to use the concept of elliptic curves [116].

In [117], it is mentioned how in 1976 Whitfield Diffie and Martin Hellman spoke for the first time of this new method to secure key exchange through the use of basic modular arithmetic. Allowing two people or users to exchange keys without having to meet physically. This method was published in a paper entitled “New directions in cryptography” [118], where it is established that both parties must define a secret value and apply a one-way function to obtain the respective key. Furthermore, this function has the characteristic of being easy to implement, however implementing its inverse is very complicated and takes a lot of time [119]. On the other hand, in [120], it is mentioned that as technology has advanced, this algorithm has gained popularity and acceptance in various applications, both in security, in the area of communication on mobile devices, to carry out financial transactions, and in authentication systems on the Internet, to mention the most important ones. Taking all this into account, it is possible to say that ECC is considered one of the safest and most efficient methods for public key cryptography [121], especially in a context where the need to protect large volumes of data and communications continues to increase. In addition, according to [122], it is the ideal algorithm for devices with processing and storage limitations because it does not consume many computational resources.

It is important to be able to define a good environment to create the cryptosystem [123]. In other words, find a cyclic group defined by a polynomial [124], as shown in Figure 1.12, where the respective graph is shown for the polynomials $y^2 = x^3 - 4x + 1$ on the left and $y^2 = x^3 - 5x + 5$ on the right. The coefficients a, b, and c are coefficients that cause a deformation in the system.

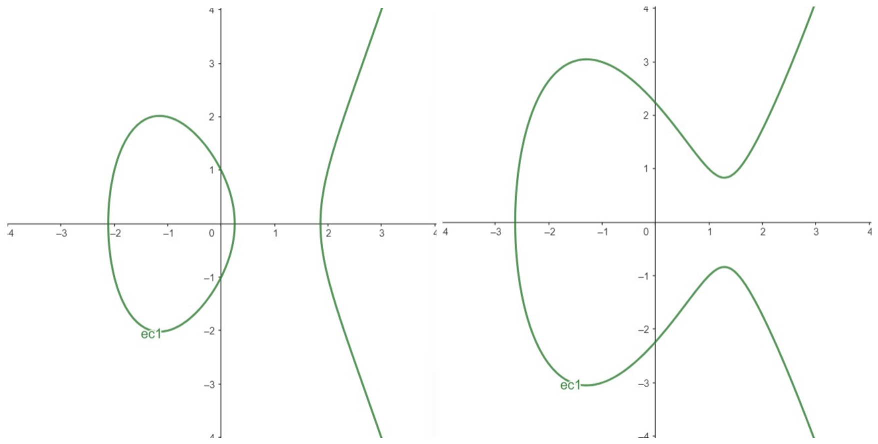


Figure 1.12: Elliptic curves representation.

As can be seen in Figure 1.12, both create a kind of curve that is composed of a set of coordinates or points (x, y) that belong to the set of real numbers R. However, to apply it to cryptography, this field of real numbers must be changed by the finite field GF(), where the operations are performed by applying modulo to a prime number p [125]. In [26], it is mentioned that these elliptic curves must be non-singular or that their graphical representation must not have vertices or self-intersections. The way to check this condition is by applying the discriminant $-16(4a^3 + 27b^2)$ and obtaining a result different from zero. An example of this non-singularity is seen in Figure 1.13, where we see a real elliptic curve equation graph over real numbers, having the general structure $y^2 = x^3 - 3x + 3$. In this equation, their respective coefficients determine the shape of the elliptic curve when graphing it.

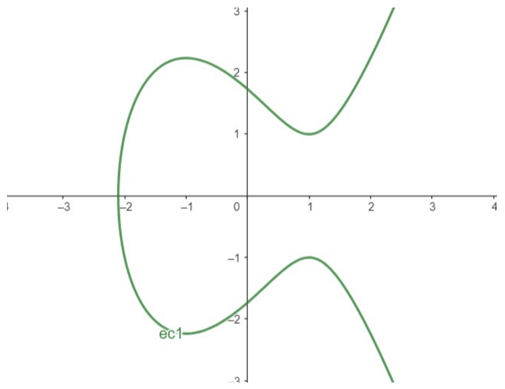


Figure 1.13: Elliptic curve with different coefficients.

Regarding the operations that can be performed we have:

- **Point addition** $P + Q$**:** According to [126], this means that an addition is performed between two different points, such that $R = P + Q$. Then it is made to cross those points on the curve, as seen in 1.12, this causes the line to extend so that there is another intersection with another point on the curve. If this point is located on the x-axis, the reflected point $R = (x_3, y_3)$ will be obtained. The operations that are performed to calculate those points are:

$$s = \frac{y_2 - y_1}{x_2 - x_1} \mod p \tag{1.12}$$

$$x_3 = s^2 - x_1 - x_2 \tag{1.13}$$

$$y_3 = s(x_1 - x_3) - y_1 \tag{1.14}$$

where s represents the slope of the line, and the equations 1.13 and 1.14 represent the values for the point R.

- **Point doubling $P + P$:** According to [127], this translates to performing an addition between the same point, such that $P+Q$ where $P = Q$ and resulting in the expression $R = P+P = 2P$. In this case, as seen in 1.15, a tangent line is drawn that passes through the point P until a new intersection point with the elliptic curve is obtained. When this is achieved, said point is reflected in the x-axis that represent the point $R = (x_3, y_3)$. In this case, the operations necessary to calculate the points are:

$$s = \frac{3x_1^2 + a}{2y_1} \mod p \tag{1.15}$$

$$x_3 = s^2 - 2x_1 \tag{1.16}$$

$$y_3 = s(x_1 - x_3) - y_1 \tag{1.17}$$

where s represents the slope of the tangent line, and the equations 1.16 and 1.17 represent the values for the point R.

Discrete Logarithm Problem (DLP)

This problem is based on a theorem stating that points on an elliptic curve have cyclic subgroups and that under certain conditions all points on an elliptic curve form a cyclic group [128]. Using this theorem, the fact of finding the correct value for the exponent x in the equation $g^x \equiv h \mod p$ is established, where g and h are elements of a finite group, and p is a prime number [126]. Although it may not seem so at first glance, in [129], it is mentioned that this equation is difficult to solve at a computational level because there is no algorithm that is efficient enough to achieve it.

On the other hand, this concept is taken to perform the encryption of elliptic curves, changing its name to Elliptic Curved Discrete Logarithm Problem (ECDLP) when adapted to be used with elliptic curves [130]. In simple words, it is about finding an integer k knowing the existence of a point P of the curve and in this way finding another point T, following the expression $T = kP$. This means that a scalar multiplication of P by k must be performed, in addition to having a difficulty similar to that of the original discrete logarithm problem. To fulfill these multiplication operations, the definitions seen on modular arithmetic [131] must be applied. As seen in Algorithm 4, the pseudocode is represented with the Double-and-Add Algorithm for Point Multiplication, being a variant of the square-and-multiply algorithm, changing the doubling instead of squaring and the addition of P instead of multiplication. In this case, d is the binary representation of the exponent. In addition, [132] mentions that the fact that ECC has a similar difficulty to DLP is what allows ECC to offer a high level of security when exchanging smaller keys, compared to other encryption algorithms.

Algorithm 4: Pseudocode Double-and-add

```
  Input: An elliptic curve $E$ and its point $P$;
  A value $d = \sum_{i=0}^{t} d_i 2^i, \quad d_i \in \{0, 1\}, \quad d_t = 1$
  Output: $T = dP$
1 $T = P$;
2 for $i = t - 1, \dots, 0$ do
3     $T = T + T \mod n$;
4     if $d_i$ then
5         $T = T + P \mod n$;
6 T;
```

Key exchange process

This process is a variation of the original Diffie-Hellman protocol where each party or user generates a key pair, the private key being the private key from a known point on the [133] elliptic curve. This process is summarized as follows:

1. Alice chooses a random private key $k_{prA} = a$ within the range $2, 3, \dots, \#e - 1$, where $\#e$ is the number of points within the curve.
2. The public key $k_{pubA} = a \times P = A = (x_A, y_A)$ is calculated, where P is a known point on the elliptic curve.
3. Bob chooses his private key $k_{prB} = b$ within the same range.
4. The public key $k_{pubB} = b \times P = B = (x_B, y_B)$ is calculated.
5. Alice and Bob exchange the public keys A and B that they calculated.
6. Alice having Bob's public key B and taking her own private key a must calculate the joint secret such that $aB = T_{AB}$
7. Bob performs the same process taking Alice's public key A and his own private key b to calculate the joint secret $bA = T_{AB}$

Chapter 2

State of the Art

This chapter will discuss data protection using encryption algorithms such as AES, RSA and ECC. Besides, articles will be reviewed that compare and analyze these algorithms, detailing their differences and their importance for data security.

2.1 Advanced Encryption Standard (AES)

Data protection has evolved to become a fundamental part of information security and data transmission [134]. For this reason, it has been considered to use an encryption algorithm known as AES, being proven to be faster, stronger, and more secure when transferring data over nonsecure channels [135]. However, due to its architecture and the fact that it uses a symmetric encryption method, it has been the victim of constant cyber-attacks and hacks [136, 137, 138, 139]. This has greatly reduced its reliability. Therefore, alternative ways have been conceived to make it more secure by combining it with other encryption algorithms [140, 141]. Among the most common encryption algorithms that are considered to be combined with AES are RSA [142] and ECC [143]. This is because its architecture and operating method are based on asymmetric encryption [144]. In this way, as mentioned in [145], a hybrid encryption algorithm combining AES with RSA is implemented. This new algorithm becomes beneficial since it creates a non-linearity during encryption with AES, in addition to greatly reducing measure algebraic and linear attacks [146]. Another example of a hybrid encryption algorithm is seen in [147]. AES is combined with ECC and thus generates a shared key that is used to both encrypt and decrypt the data. This method is considered efficient and fast since it considers the use of one of several elliptic curve options [148], where complexity and security increase depending on the chosen curve.

2.1.1 AES Applications

Continuing with the articles, we are now going to take a close look at those articles that mention the various applications that AES algorithm has in the field of data security. In [149], a variant of the AES algorithm is proposed that includes a structure called modified positive polarity reed muller (MPPRM) [150], resulting from the modification of the positive polarity reed muller architecture (PPRM). This new way of representing Boolean functions replaces the Look-Up Tables (LUTs) [151] and thus reduces hardware requirements. Thus, the new model is implemented in a field programmable gate array (FPGA) [152] as a security measure to keep data safe during wireless transmission. Likewise, [153] describes the design of a substitution byte (S-box) for the AES algorithm, optimizing its implementation in Spartan – 3 FPGA to be more efficient in area through the integration of the S-box and its reverse logic in a single signal-controlled module. This reduces the number of logic gates needed by approximately 5%, using only 78 EX-OR gates and 36 AND gates. The application has proven to be very useful in ensuring protection against side channel attacks by including data with a random bit sequence. On the other hand, in [154] a new hardware masking technique [155] is presented for the AES algorithm that can maintain a constant latency, unlike previous methods that do increase the latency of the algorithm. The idea is based on Masked Dual-Rail with Pre-charge Logic (LMDPL) that helps to avoid interference and failures. This way, it is possible to implement AES encryption of type AES-128 in 10 cycles and AES-256 in 14 cycles and test its resistance to attacks. This way, they try to extract information from the energy consumption of the device. In the same field of security, the AES algorithm has been used to protect data on IoT devices. For example, in [156], the authors propose a variation of the AES encryption method to maintain information security in the Internet of Things (IoT) environment. To do this, the architecture of the encryption process is modified either by increasing operations to improve the diffusion of the data and thus, improve the avalanche effect or by modifying the operations of the sub-bytes. Achieving that, there is more dependence on the round keys in each round of encryption. Taking these articles into account, the importance of the AES algorithm in the field of data security has been demonstrated.

2.1.2 AES Challenges

In this section, we will analyze the weaknesses and challenges that we must face when choosing the AES algorithm. An example can be seen in [157], where the problem that exists with data that is stored in the cloud is raised, since this method of backing up information is mostly used by organizational applications and big data due to its extensive advantages [158]. However, this storage method uses the AES-256 algorithm as a protection measure, being its most secure version but whose execution time is high. Besides, the authors propose a new form of 3D-AES encryption that provides a higher level of security by encrypting data stored in the cloud, unlike the standard security methods offered by the cloud service. Another of the challenges presented by AES is mentioned in [159], where the difficulty that exists when efficiently implementing SubByte transformation operations using LUTs or MixColumns operations is mentioned. Unfortunately, these operations can take up a lot of time, memory space and slow down the process. To solve this, the authors propose the use of shifting and XOR operations, to improve the performance of the algorithm both in speed and resource consumption. Within the same area of data security, we have [160], that mentions the Extended Sparse Linearization (XSL) attack as a cryptanalysis method that generated controversy after being proposed in 2002, stating that it could break AES encryption faster than brute force attacks. However, this type of attack requires a high computational cost, contradicting this statement when compared to other attacks. This fact leaves the AES algorithm as a secure algorithm, but still in the sight of future attacks. Therefore, each of the previous articles demonstrate how the AES algorithm, despite having a relatively simple architecture, can be complicated to implement for certain devices. Or even being the target of attacks to break its security.

2.2 Rivest Shamir Adleman (RSA)

If a developer wants to exchange the shared key for symmetric encryption over an unsafe network like the well-known internet network, the RSA algorithm is a better option [25]. Being an algorithm that applies asymmetric encryption [144], it has become an important part of issues related to electronic signatures, as mentioned in [161], where a new scheme for the creation of this type of algorithm is proposed. The authors use signatures where a smaller number of prime numbers is required when generating the public and private keys. Similarly, according to [162], a new communication protocol is proposed based on intelligent proxies that allow users to maintain anonymity. As this is a system that works in real time, it is very useful when performing correct authentication and eliminating known security risks, in addition to providing a better level of security for smart devices. Another area where the use of RSA stands out is network security, as mentioned in [163]. Here, the combination of the DES and RSA encryption algorithms is proposed in addition to the SHA-1 function. It results in a process that reduces the size of the key and improves the existing complexity when generating the keys, all while using fewer computational resources. Apart from this, for the field of banking transactions, [164] mentions the use of Firebase Password Authentication together with RSA Encryption as authentication methods for this type of application that offers a virtual financial management service. Likewise, the RSA algorithm is useful for the transmission of messages between individuals by applying the Public Key Infrastructure (PKI) to create both keys (the public key for the one who encrypts the message and the private key for the one who decrypts the message) [165, 166].

2.2.1 RSA Applications

Taking a more focused look at the applications of the RSA algorithm, we can mention the research made in [167]. Where the authors propose a new image encryption algorithm that combines RSA with chaotic maps. To do this, images are used as input data, permuting each one through specific maps. Subsequently, these permuted images are encrypted with the RSA algorithm and then with a 3D logistic map. At the end of the process, the encrypted images are swapped one last time. This approach ensures that the encrypted images have a high level of security, a uniform histogram, and a low correlation, being the main characteristics evaluated by the authors. In addition, according to [168], the use of chaotic maps provides complexity and randomness to the encryption process. This kind of method is beneficial if combined with the robustness of the algorithm when working with large prime numbers that are difficult to factor. Following the same path of security, we have the work seen in [169], where the authors propose a modification in the RSA architecture by increasing the number of private keys to improve its security. Considering that this type of encryption uses a public key and a private key, it must be taken into account that the private key may be vulnerable to inspection. The authors use this modified encryption method to encrypt image arrays, obtaining results that demonstrate that this new system is much more difficult to break, in addition to significantly increasing the security of the RSA cryptosystem. On the other hand, the RSA algorithm can be used to maintain security during real-time message exchange. For example, in [170], the use of the RSA algorithm is mentioned to improve the security of instant messaging applications. The authors explain how the RSA decryption process can be accelerated using a technique called the Chinese Remainder Theorem (CRT). This technique, known as RSA-CRT [171], allows messages to be decrypted up to three times faster than normal RSA, especially when very long keys are used. To do this, a comparison is made between the original RSA method and the modified RSA-CRT method, using three key sizes: 1024 bits, 2048 bits, and 4096 bits. The results obtained demonstrate that there is a reduction in the execution time for both data encryption and decryption. Making it a practical tool for maintaining security in messaging applications, helping to keep conversations private and secure. Another example is seen in [172], where a modification in the architecture of the RSA algorithm is mentioned to maintain security during data transfer on the Internet. To do this, the researchers propose a version of RSA with 3keys (3k) to make it more secure, in addition to implementing and testing it in the MATLAB application. Their results show that this version of RSA with 3k is efficient at keeping information secure, just having few errors when recovering encrypted messages. After analyzing these articles, the importance of this algorithm in the field of data security is evident, since many companies and people need to send private information securely over the Internet.

2.2.2 RSA Challenges

Although the RSA algorithm is very useful when transferring data over the Internet, it also presents certain disadvantages or challenges that must be taken into account when using it. An example of this is seen in [173]. Here, the authors highlight an important weakness of RSA, being that some RSA private keys are easier for attackers to guess. To test this, a program was created that uses graphics processing unit (GPU) to find these weak keys faster than with normal processors or central processing unit (CPU). This new program was tested with different key sizes and GPU configurations, demonstrating that it can be useful for internet providers. It would make possible to detect and eliminate weak keys from their users, thus improving security. This program can be run with either a GPU or CPU, in addition to being freely accessible to users. Another example can be seen in [25], where the author refers to the advantages offered by the RSA algorithm compared to symmetric algorithms. However, in exchange for a better security level, the execution time increases considerably. Likewise, in [160], an analysis of different symmetric and asymmetric algorithms is carried out, where the preference of symmetric algorithms over asymmetric ones is established due to their lower efficiency. Knowing that symmetric algorithms are faster and more efficient, makes them more attractive to software developers. However, asymmetric algorithms allow developers to adjust the security of their software by modifying the key size, something that symmetric algorithms do not allow. In this way, flexibility in determining the key size is an advantage of asymmetric algorithms, but their main disadvantage remains the lower efficiency compared to symmetric algorithms. Another investigation focused on highlighting the weaknesses of the RSA algorithm is [174], focusing on slowness of this encryption system compared to other encryption algorithms. This slowness becomes a significant problem when data needs to be encrypted and decrypted in real time. For example, in the case of digital video, where speed is crucial to maintaining the quality and fluidity of the content. Additionally, RSA may require more computational resources, limiting its use on devices with limited capabilities. Therefore, it is necessary to combine the virtues of certain encryption algorithms into one, using the security of RSA together with the speed of lighter algorithms. This way, it would be possible to create a hybrid encryption system that is both secure and fast. Furthermore, it can result in an suitable application for working with data in real time. In this way, we can see how several articles have mentioned the main disadvantages and challenges that must be taken into account during the implementation of the RSA algorithm for data protection.

2.3 Elliptic curve cryptography (ECC)

Elliptic curves offer a variant of asymmetric encryption, due to it takes advantage of more advanced mathematical properties allowing to have a higher level of security. Among the characteristics of ECC we have that it requires smaller key sizes [175], lower memory usage [176], lower bandwidth [177], and better performance when doing the same work as his previous encryption colleagues [113]. On the one hand, in [178], the optimization of the ECC algorithm for the P-256 elliptic curve is proposed through the use of 256-bit prime fields. This process ensures that the algorithm is executed in less time and consumes less energy resources, being beneficial for devices belonging to the Internet of Things (IoT) [30]. Similarly, the use of IoT, despite it allows both virtual and physical communication between various devices regardless of the type of hardware or software, also represents a great risk regarding data privacy. As seen in [179], this weakness in data security is related to hardware limitations, making the task of implementing security measures almost impossible.

2.3.1 ECC Applications

The ECC algorithm has been used for various purposes, one of this is mentioned in [180], where the authors address the issue of protecting confidential grayscale and color digital images during their transmission over networks. This proposal results in a method that combines modified elliptic curve cryptography (MECC) with the Hill cipher (HC) [181], called MECCHC. Considering the layer of protection that Hill Cipher offers by being a symmetric method that simplifies and speeds up the key generation process. Here, the combination of the two methods make it more difficult for intruders to decrypt the images. Thus, HC uses 4x4 keys along with elliptic curve parameters to facilitate decryption without the need to calculate the inverse matrix. In the same field of data security, we have [182], inhere an improvement to security in real-time IoT applications is proposed. For this, the authors develop a hardware accelerator that is able of improving the performance of the ECC, specifically for the NIST P-256/-521 elliptic curves. The results have shown that their proposal is effective for algorithms to perform faster and safer calculations, in addition to testing the design with different technologies to ensure its effectiveness and efficiency. Another way of applying ECC is by combining it with other encryption algorithms. For example, in [183], the creation of an improved method to protect information in embedded systems [184] is mentioned, being common electronic devices in everyday life. These devices work by combining two encryption techniques: AES for its speed and ECC for its security. Here, the ECC algorithm is used to protect the encryption key that will be used when the AES algorithm is applied when encrypting medical images. The authors have optimized the hardware for both ECC and AES, allowing faster, more efficient calculations and reducing processing time without losing security. Additionally, the ECC algorithm is used in devices with few computational and energy resources, as mentioned in [185], where it is proposed to improve the security and efficiency of light electronic devices using ECC. The authors implement the ECC algorithm focused on devices with space and memory limitations, through the use of the specific Galois field (GF (2^3)). In this way, the size and speed of the cryptographic processor can be reduced. Testing was performed on an 8-bit ECC cryptoprocessor, providing results demonstrating that the compact design is suitable for lightweight devices. Finally, we have one of the properties of elliptic curves that can allow the creation of new encryption methods. This can be seen in [186], where the authors propose the creation of a new algorithm called supersingular isogeny based key exchange (SIKE), specialized in exchanging keys through isogenies between supersingular elliptic curves. This algorithm can be seen as an alternative to traditional systems such as RSA and ECC, as it offers resistance against quantum computer attacks. The key advantage of SIKE is that it maintains small key sizes, similar to those of ECC, while providing security against quantum threats. This makes it especially useful in situations where there could be quantum attacks. In this way, we have analyzed some articles that detail the use of the ECC algorithm in the field of information security, and its importance for devices with limited resources. In addition to being the door to the development of new encryption techniques.

2.3.2 ECC Challenges

Although the ECC algorithm has several advantages compared to other encryption methods, there are also challenges it must face as the technology evolve. One of them is seen in [187], where the authors mention how the great advances in quantum computing threaten the security of current cryptographic systems. The authors mention that quantum algorithms can easily break many of the encryption methods we use today, such as AES, 3DES, RSA, Diffie-Hellman, and ECC. An example of this is the Grover and Shor quantum algorithms, two algorithms that can solve problems much faster than traditional methods. To demonstrate this, such vulnerability is analyzed, and different types of post-quantum cryptosystems that could be secure against quantum attacks are explored. Another challenge that must be faced when using ECC is seen in [188], where it is proposed to improve IoT security using ECC cryptography in embedded devices. However, the ECC algorithm faces security threats, being the most important the side channel attacks (SCA). The article reviews the latest proposals to prevent these attacks in ECC, specifically in primary fields, evaluating the balance between security and performance. Furthermore, suitable solutions for these devices are examined, as well as the problems associated with these countermeasures. On the other hand, there is a disadvantage that the ECC algorithm has when using it in IoT. This can be seen in [189], where the authors carry out an analysis of some encryption methods, including ECC. The authors mention that although ECC offers protection, it presents significant challenges in its implementation for authentication. This is because if there are no additional security measures in place, it can be vulnerable and create security holes. Additional layers of defense are thus required to ensure maximum protection against attacks, but increasing computational complexity and potential hardware overhead. Through the review of these articles, it has been proven that both the ECC algorithm and the rest of the encryption algorithms have disadvantages or challenges that must be resolved either for future technologies or simply to achieve a balance between security and efficiency.

2.4 Comparative evaluation of studies on encryption algorithms

This section presents articles that analyze the performance of several encryption algorithms considering metrics. For this reason, in [190], a study is carried out where AES, DES, Blowfish, and RSA methods are compared, taking into account data such as text or images and time and performance metrics. Furthermore, in [33], the analysis between the AES and DES symmetric encryption algorithms is mentioned, taking into account time, throughput, and CPU utilization through the use of different text sizes. In the same way, in [191], a comparison is made between several symmetric algorithms and asymmetric algorithms, taking into account the data in the cloud computing system. Apart from this, research has also been done focused on analyzing the performance of these same algorithms on smaller devices. As mentioned in [192], it is proposed to compare the performance of the AES, DES, TEA, RSA, and REA algorithms in both execution time and battery consumption, considering the use of files of various sizes. Although most analysts take into account the use of text as their main tool for these analyses because it can be easily manipulated, in addition to allowing real data to be represented with variable length and complexity. In addition, the security offered by AES, RSA, and ECC algorithms may be affected over the years with the arrival of quantum computing. This is seen in [193], where the vulnerability of these traditional algorithms against the new algorithms that work on quantum computers is highlighted. Surpassing them both in efficiency and time. For this reason, the article establishes what measures can be taken to resolve privacy risks, in addition to minimizing the effect on users' privacy requirements in the IoT.

With all these articles, it has been possible to describe the strengths and weaknesses of encryption methods in data protection, both individually and as a set. Taking this into account, this research offers a more practical approach, through the development of graphical interfaces. This feature will allow the evaluation of performance, and also allow the visualization of the process followed by each algorithm. In addition, because the high-level Java language is used, it is possible to develop and implement optimizations to these codes for several use cases. This fact is valuable for future developers who wish to delve deeper into the subject of data security through practice.

Chapter 3

Methodology

This chapter describes in detail the methodological approach adopted to achieve the stated objectives. First, we described in detail the process to solve the problem. Then, we analyzed the steps to develop the graphical interfaces for each selected symmetric and asymmetric encryption algorithm. Finally, we propose the metrics needed to evaluate the performance of each algorithm.

3.1 Problem analysis and solution development

Throughout this section, we take into account what was discussed in chapter 2 and 3, where some encryption algorithms have a series of challenges that have been addressed according to time execution or computational resources consumption. Unfortunately, articles such as [192, 33, 191] have proposed evaluation metrics for different encryption algorithms in different scenarios. These scenarios can be varying the data size, modifying operations in the system architecture or using different hardware. Taking into account, such results after several years of technological advancement can result in an encryption algorithm that was good at that time, but is no longer as good today. On the other hand, although a certain number of articles talk about a cryptographic system and present a code implementation, it lacks a graphical interface or is simply not available. In addition, it must be taken into account that the repositories of the vast majority of these articles work with predefined input data, and if they are modified to include your own data, the result may not be optimal or simply give an error. Taking this into account, performing the analysis of different encryption techniques remains as a subject of study, mainly due to the computing resources required to process large volumes of data to validate and replicate the results. This is because most experiments involve have to process millions of operations per second, as well as handling a dataset that can easily reach several gigabytes. To address this problem, reduced datasets have been used to minimize computational costs, implement the latest versions of high-level language and modern hardware. Therefore, this research aims to create a graphical interface for the encryption systems AES, RSA, and ECC. As well as to be able to visualize the process that each one generate, and to allow capturing the metrics necessary for performance analysis. The interfaces designed have the following characteristics:

- **Clarity of Navigation:** Each interface features an arrangement of elements that makes it easy to use, as well as allowing interaction with data and algorithms through buttons and sliders.
- **Process transparency:** Provide a visualization of the encryption and decryption process, allowing to see how the data changes with each stage.

- **Configurability:** Provide the ability to choose encryption modes either using a specific key size, a desired elliptic curve, or encryption and decryption mode.

3.2 Implementation details

Performance criteria for build the encryption algorithms

- **Key and input data size:** The size of the keys and the plaintext is essential to verify the performance of the algorithm during encryption and decryption.
- **Operation modes efficiency:** It provides a selection of different encryption modes to visualize how the data is processed. In this case, each algorithm has two or three execution modes. The user can configure the encryption and decryption options before executing the algorithm.
- **Scalability:** Each algorithm allows to process different data sizes.
- **Metrics capture:** Each algorithm allows to capture the respective evaluation metrics with each execution.

3.3 Symmetric encryption model

For the development of a symmetric algorithm, the AES encryption algorithm is taken as a representative of this type of encryption. This model has two processes: generation of the respective subkeys according to the size of the key and the encryption and decryption process using certain operation modes. Our proposal takes a string of characters to generate the required alphanumeric subkeys, which are then applied to the corresponding transformation round during the encryption and decryption of the text. This key can only contain characters from the traditional alphabet, specifically lowercase letters, as illustrated in Figure 3.1.

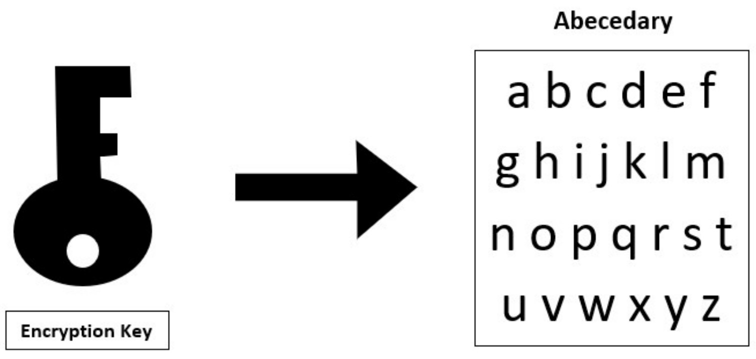


Figure 3.1: Characters that conform the symmetric encryption key.

In contrast, the AES encryption model presents two modes of operation: Electronic Codebook (ECB) and the Cipher Block Chaining (CBC). Among them, the CBC model requires two input parameters to generate the respective subkeys: the original encryption key and the initialization vector (IV). The IV must match the format and size of the main key. Both modes of operation are described as follow:

- **ECB mode:** In this mode, each block of plaintext is encrypted independently using the same key, as can be seen in Figure 3.2. Consequently, identical blocks of plaintext will produce identical blocks of ciphertext.

This fact could potentially reveal patterns in the encrypted data. For further details on this mode of operation, please refer to the article [194].

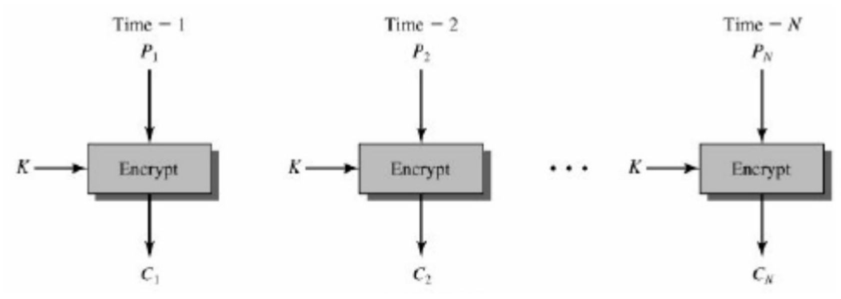


Figure 3.2: Electronic Codebook mode structure [11].

- **CBC mode:** In this mode, each block of the plaintext is XORed with the previous encrypted block before encryption, as illustrated in Figure 3.3. This chained process ensures that blocks of identical text result in different encrypted blocks, improving the security of the information by modifying the patterns of the encrypted data. For further details on this mode of operation, please refer to the article [195].

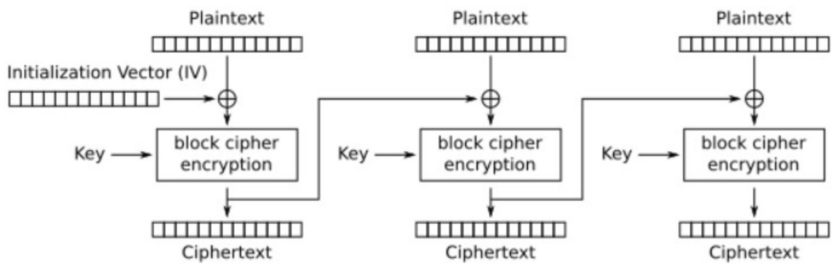


Figure 3.3: Cipher Block Chaining mode structure [12].

During the implementation of AES algorithm, different layers are considered. Among these layers, the most significant processes include:

- **SubKey generation:** create each subkey before encryption and decryption.
- **RoundKey addition:** Applying the round key through the addition operation.
- **Byte substitution:** Using fixed substitution tables or S-boxes for replacing bytes.
- **ShiftRow:** Performing a byte level shift for each row in the block.
- **MixColumns:** Mixing byte values at the columns level within the block

After finish the required number of rounds the result is given as a hexadecimal format and is referred to as the ciphertext. During decryption, the process is reversed: the ciphertext is taken as the main input, with the same subkeys used for encryption. AES algorithm decrypts by performing the reverse process proceeding from round 10 back to round 1, producing a hexadecimal string that the algorithm converts into a readable text format. It is important to know that each data block processed by AES must be of uniform size, so the size of the plaintext must be adjusted. This adjustment is achieved through padding, as discussed in [11]. Padding involves the use of strings, such as series of $0's$ or $1's$ to achieve the block size required by AES. For this implementation, the 128-bit AES version was used and the taken padding was a string of $0's$ to include it in the plaintext when necessary.

3.4 Asymmetric encryption models

This research includes the implementation of two algorithms from the asymmetric encryption category. Asymmetric encryption uses a pair of keys to complete the process, one to encrypt the information or message and the other to decrypt the information.

3.4.1 RSA model

To generate the public and private keys of the RSA system, two random numbers p and q must be selected. These numbers should have the same bit size and satisfy certain mathematical requirements. The RSA algorithm implemented in this research supports three key sizes, being 1024 bits, 2048 bits, and 3072 bits. Consequently, p and q must be prime numbers and should each be half the bit length of the desired keys. Fro example, if the key size is 1024 bits, the bit-length of p and q should be 512 bits.

The key generation process starts by computing $n = p \cdot q$. The result n will have a size that will be twice that of p and q, and this process is applied for all key sizes of the algorithm. The values for p and q are not manually set by the used, but the algorithm will automatically generate a random value for each variable and applies the Fermat's Little theorem to ensure these values are prime. The next step involves calculating the Euler function $\phi(n)$, which is used to determine the number of positive integers less than n and that are coprime to n. The $\phi()$ value is important for calculating the private key d and establishing the relationship between the public and private keys. It ensures the correct functioning of encryption and decryption process.

The algorithm then generates a new random value e, which represents the public key. This value must meet specific criteria: it should be less than n, and pass the Fermat test to verify if it is a prime number. Moreover, e must be coprime to n. This condition is verified using the following equation 3.1.

$$gcd(e, \phi(n)) = 1 \tag{3.1}$$

If e does not net this requirement the process will select a new value e and repeat the verification until a suitable e is found. Once e is determined, the private key d can be calculated using the formula 3.2.

$$d \cdot e = 1 \mod \phi(n) \rightarrow d = e^{-1}(\mod \phi(n)) \tag{3.2}$$

This equation indicates that "d is the modular multiplicative inverse of e modulo $\phi(n)$". In other words, when the value d is multiplied by e, the result divided by $\phi(n)$ must leave 1 as a remainder. With both the public and private keys generated, the next step involves using this keys for data encryption and decryption.

To perform encryption using the RSA algorithm, the system starts by processing the plaintext defined by the user, ensuring that the input of the plaintext does not exceed the size of the key use. For example, if the user selects a key size of 2048 bits, from the options presented in the graphical interface, the input text must be the same size or smaller. The system will detect if the text exceeds the extension limit and will provide a warning box for the user to correct the text. In this way, errors during the encryption and decryption process are avoided. This constraint is due to the limitations inhered in RSA algorithm. This algorithm cannot handle excessively large text lengths due to its computational process.

Once the plain text is defined, the algorithm converts the string of characters into hexadecimal format and then transform it into decimal format, as shown in Figure 3.4. This conversion is crucial because the encryption formula defined in 1.10 requires all parameters be positive.

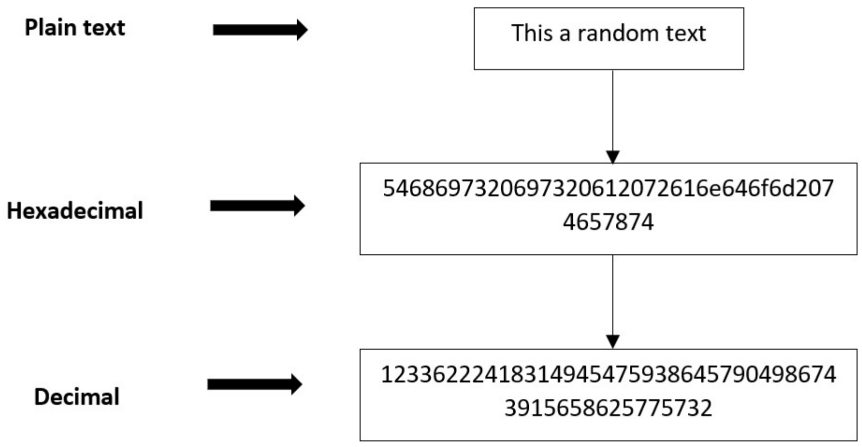


Figure 3.4: Transformation process for the plaintext in the RSA algorithm.

While it is possible to apply encryption directly, doing so would result in a significant computational load. To optimize resource usage, the Square-and-Multiply for Modular Exponentiation is employed, as illustrated in Algorithm 3. This method performs operations at the bit level by using the public key, iterating through each bit value from left to right. During this process square of the value or the multiplication is applied according to the bit value, until going through the entire chain. The final result will be the ciphertext represented in hexadecimal format.

The decryption process follows a similar approach, but with a modified equation 1.11, using the private key instead of the public key and taking the ciphertext instead of the plaintext. After this process, the program will show the decrypted text in the original format that was entered in plaintext.

3.4.2 ECC model

In the ECC model, the principle of generating two keys - a public key and a private key - is similar to RSA but uses key of smaller sizes. Despite the smaller key sizes, ECC provides a higher level of security for key exchange between users. Additionally, ECC encryption is usually combined with other encryption methods, specifically symmetric algorithms. In this implementation, ECC encryption method has been used to create the public key, the private key, and the shared key.

To begin with the key creation process, the user must first select an elliptic curve that will used for the symmetric encryption, among them there are 3.3, 3.4, and 3.5:

$$\begin{aligned} y^2 &= x^3 + 1174x + 1 \pmod{2^{251} - 9} \\ x &= 1582619097725911541954547006453739763\ldots \\ y &= 1408064552111686812003849472883342418\ldots \end{aligned} \tag{3.3}$$

$$\begin{aligned} y^2 &= x^3 - 3x + 4105836372515214\ldots(\mod 115792089210356248\ldots) \\ x &= 48439561293906451759052585252797914202\ldots \\ y &= 36134794756577183366534992453949150051\ldots \end{aligned} \tag{3.4}$$

$$\begin{aligned} y^2 &= x^3 + 6277101735\ldots x + 27026278796\ldots(\mod 627710173538668\ldots) \\ x &= 2722199071451431051069537841689051308491261328636682477198 \\ y &= 4418243372714423442765109689069973422089114465591206284845 \end{aligned} \tag{3.5}$$

These three equations follow the format of an elliptic curve equation, being defined as seen in Eq. 3.6 where a and b represent the coefficients of the elliptic curve and p is the prime number that defines the finite field.

$$y^2 = x^3 + ax + b\ (mod \quad p) \tag{3.6}$$

In addition, each of the above equations has the values x and y, representing the coordinates of the base point or generator $g()$ on the curve.

Once the curve to be used is defined, two random numbers must be chosen by the users who will carry out the communication. This generated value represents the private key of each individual. Once the numbers are defined, reminding they are randomly generated by the algorithm, the public key is calculated using the following equation 3.7, where the value a represents the private key generated by the user and the operation performed is the multiplication of a point on the curve by a scalar.

$$k_{pubA} = a \times P = A = (x_A, y_A) \tag{3.7}$$

After this process the algorithm will create the joint secret or the encryption key given by the following equation 3.8, where a represents the secret key of the person who is going to calculate the shared key and B is the public key that has been received from another person.

$$T_{AB} = aB \tag{3.8}$$

This process must result in the same value of T_{AB} and must be carried out by both users to verify that it is indeed only the two of them involved in the conversation. Otherwise, the communication has been intercepted. After finishing the key exchange process, the algorithm will take a coordinate from either the x-axis or the y-axis of the joint secret and apply a SHA-1 to it to form a new 128-bit key. It will then apply the same process that is carried out for the AES encryption algorithm. In this way, the result of the data encryption is a text string represented in hexadecimal.

In the case of decryption, the algorithm takes the encrypted text and, knowing the values of both the elliptic curve and the joint secret, proceeds to apply the respective SHA-1 to the key and perform the reverse process of the decryption defined by the AES algorithm. Thus, the final result of the program is the original text string that the user entered at the beginning of the process. In this way, it has been possible to simulate the exchange of keys as well as achieve the integration of a symmetric encryption algorithm with an asymmetric encryption algorithm.

3.5 Experimental setup

The implementation of the encryption algorithms was carried out using the Java programming language, together with the Javac compiler. The development environment (IDE) used fr coding and debugging was Visual Studio Code (VS Code). This setup provides a versatile platform for testing algorithms. The encryption algorithms were executed on a hardware system equipped with the following specifications:

- **Operating System:** Windows 11 64-bit operating system
- **Processor:** Intel(R) Core(TM) i7-12700H CPU 2.30 GHz
- **Memory:** 16GB RAM

These hardware and software configurations played an important role in the execution and performance of the selected encryption algorithms. This allowed to ensure a robust platform for the development, testing and evaluation of the AES, RSA, and ECC algorithms.

3.5.1 Performance evaluation metrics

The performance of the encryption algorithms is evaluated through several key metrics. Among them we have:

Execution time

This metric measures the time required to convert plaintext into ciphertext. This metric is taken in milliseconds. The variation in the measurement of this metric is given by the size of the text that is introduced to the program. In addition, when recording the decryption time, it is preferable that both times are similar. It guarantees the efficiency of the cryptographic system.

Throughput

Throughput is defined as the capacity of the algorithm to process a certain amount of data in bytes in one second (Bps). This measurement is recorded in bytes and its variation depends on the size of the input data, the mode of operation, the selected key size, or the type of curve chosen before executing the code.

RAM consumption

RAM consumption is given by the amount of memory required to store the keys, intermediate data, and any additional structure that the algorithm needs for its operation. This measurement is taken in bytes and can vary depending on the hardware power and the size of the input text.

CPU consumption

CPU consumption quantifies the processing load over the processor for the encryption algorithm, including the time it takes the algorithm to perform an operation during encryption or decryption, as well as the intensity of the calculations performed. This measurement is given as a percentage and can vary depending on the size of the input text, the power of the hardware, and the initial settings that each algorithm has before executing it.

Chapter 4

Results and Discussion

In this chapter, we are going to analyze the results obtained through several testing in each of the encryption algorithms: AES, RSA, and ECC. Each test was performed considering the previously mentioned metrics and different text size were applied. The tests were run individually for each model encryption and then compared with each other. After that, the best algorithm will be selected and a comparative analysis will be carried out.

4.1 Evaluation of AES algorithm

When analyzing the results obtained by the AES algorithm, it is important to notice that each execution of the program was evaluated with different sizes of text as input. This analysis focuses on execution time. To do so, the following byte sizes were used for the tests: 446, 1100, 2142, 2972, 3686.

An important aspect when selecting the most efficient encryption algorithm is to choose one that does not take too long to execute, even with large data. For this reason, we pay special attention to encryption time when evaluating the relevance of encryption algorithms. This involves testing the overall performance of the system in real time applications. Furthermore, encrypting sensitive data quickly is essential to preserve security without sacrificing the speed and usability of programs that use encryption models to protect critical information.

Throughout this analysis, comparative graphs of each mode of the AES algorithm will be displayed. Each encryption mode defines the efficiency of the algorithm is in terms of time and consumption of computational resources. In Figure 4.1, each lines represents the time taken by the algorithm to encrypt the input data, using different text sizes in each run. However, when working on a Visual Studio Code environment with Java language, several factors related to the Java Virtual Machine (JVM) [196] and the development environment can have some influence the results. The JVM uses a Just-In-Time compiler that optimizes code at run time and requires warm-up time to achieve optimal performancem as well as garbage collection [197]. This can cause pauses during code execution. Also, heap size and JVM settings affect performance and cause inconsistencies results [198]. Similarly, Visual Studio Code and its extensions may add a slight overhead, especially in debug mode.

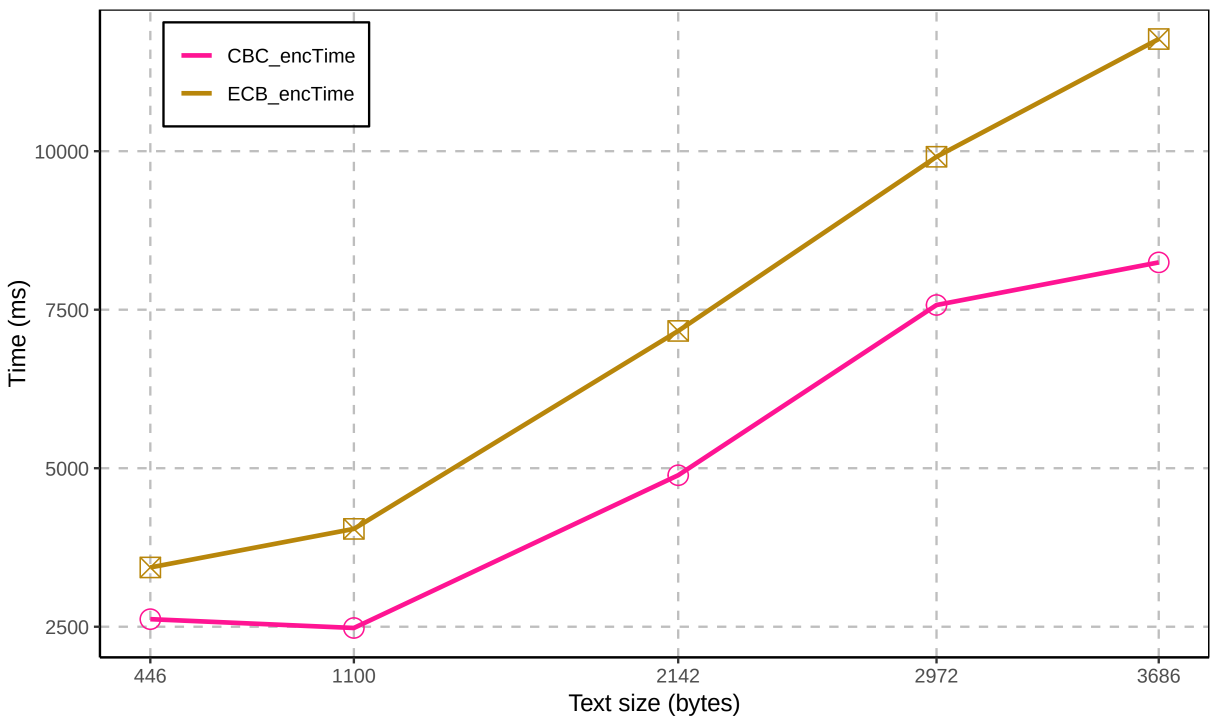


Figure 4.1: Execution time in milliseconds for cipher modes ECB and CBC.

Figure 4.1 shows a progressive growth for the execution time for both encryption mode. The separation between both lines in the graph occurs with an almost similar interval for each text size. Although it is expected that the CBC encryption mode is faster than the ECB, the figure shows the opposite. This may be due to the block management of the algorithm where each block is treated individually. Another factor affecting the time execution is the use on the CPU memory and cache. This occurs when working with many blocks of data at the same time. This difference seen in the Figure 4.1 shows how the ECB mode time execution is getting worse as the data size increases.

Another way to evaluate the performance of an encryption algorithm is through its throughput. corresponding to 446 bytes of text where the difference between ECB and CBC mode is almost zero. This is due to the size of the data used during that test, which makes it difficult to notice a significant difference. In addition, the throughput of the CBC encryption mode increases notably during the passage of 446 and 1100 bytes, and stabilizes with a slight increase for larger sizes. In contrast, the ECB mode also shows an increase, but in a more gradual and subtly manner. Growing remains below the CBC mode throughput the entire range of sizes evaluated, maintaining an almost uniform separation interval for each text size. For the ECB mode of the AES algorithm, independent process of each data block allows parallel processing, resulting in a higher throughput. However, the simplicity also makes it less secure, since identical text blocks produce identical cipher blocks. In contrast, the CBC encryption mode introduces a dependency between blocks, where each block is combined with the previous cipher block before encryption. This dependency limits parallel processing and generally results in lower throughput than the ECB mode. Despite this, the CBC mode offers higher security by generating unique cipher blocks even for identical text. In this analysis it is observed that the throughput of the CBC mode exceeds the ECB mode, which can be attributed to similar factors seen for Figure4.1.

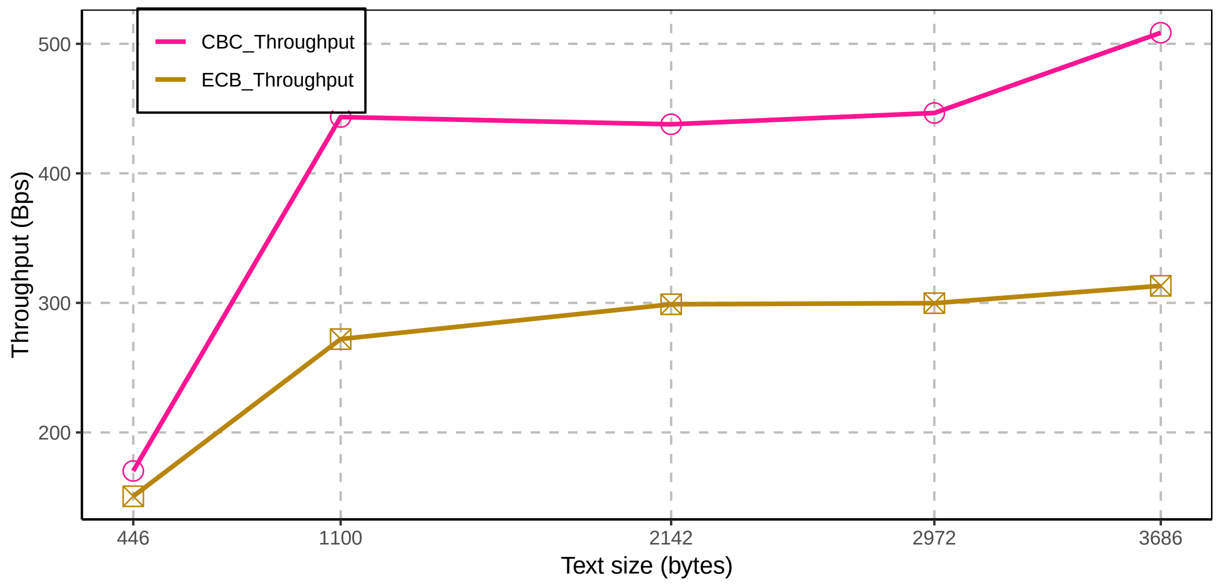


Figure 4.2: Throughput in bytes for cipher modes ECB and CBC.

Another critical metric to evaluate the performance of the encryption is the memory consumption that occurs during its execution. Figure 4.3 shows the memory usage in bytes of both ECB and CBC encryption modes. Notably there is not significant difference in temporary storage requirements between two modes, as the AES algorithm utilizes a block size of 128 bits for encryption regardless for the chosen mode. In ECB mode, each block of data is encrypted using the same key, resulting in a straightforward implementation. In contrast, CBC cipher mode encrypts each block after combining it with the previous encrypted block using an XOR operation. This creates a dependency between blocks and requires the use of an initialization vector to process the first block of text.

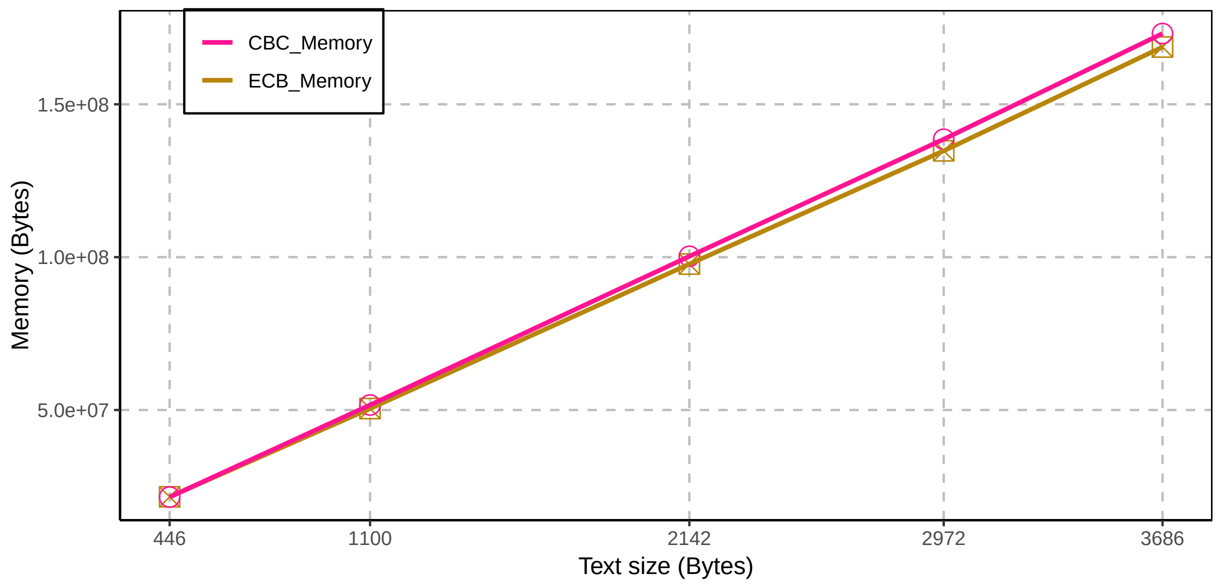


Figure 4.3: Memory consumption in bytes for cipher modes ECB and CBC.

It is important to highlight that memory usage is relatively high, ranging between 21 million bytes and 173 million bytes even for small texts inputs. This suggests that the current implementations may not be optimal for resource-constrained devices, particularly in applications where memory is a critical resource.

Lastly, the fourth metric assesses the percentage of Central Processing Unit (CPU) consumption, providing insights into the algorithms' processor utilization. Figure 4.4 demonstrate that the version of AES that exerts the least load on the CPU is the ECB mode. As noted in earlier sections, ECB mode features a simpler structure compared to the CBC mode. This design minimize the intermediate operations that could delay the encryption process. Additionally, CPU shows a proportional growth as the text size increases, reaching a maximum value of approximate CPU consumption of 65% for the ECB mode and an approximate value of 72% for the CBC mode.

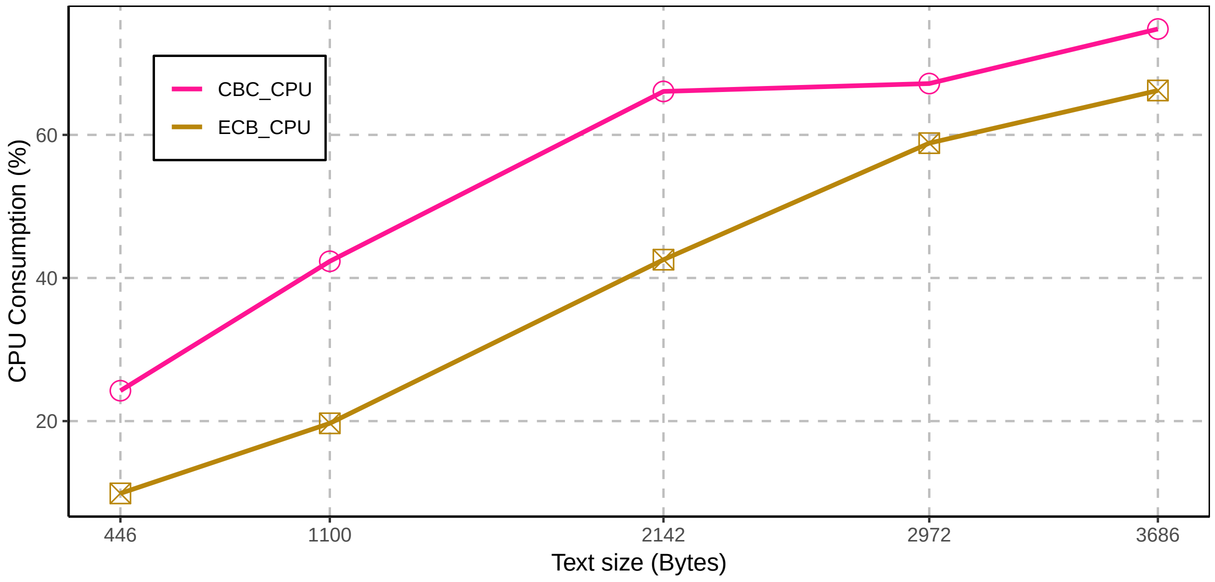


Figure 4.4: CPU consumption percentage for cipher modes ECB and CBC.

4.2 Evaluation of RSA algorithm for different key size

In each test performed for the RSA algorithm the length of the text was adjusted to match the size of the key. Three key size were evaluated: 1024 bits, 2048 bits, and 3072 bits. Figure 4.5 shows the result of five encryption tests for each key size. As in the case of AES algorithm, the encryption time similar to the decryption time for these three types of keys, making it unnecessary to display on a chart.

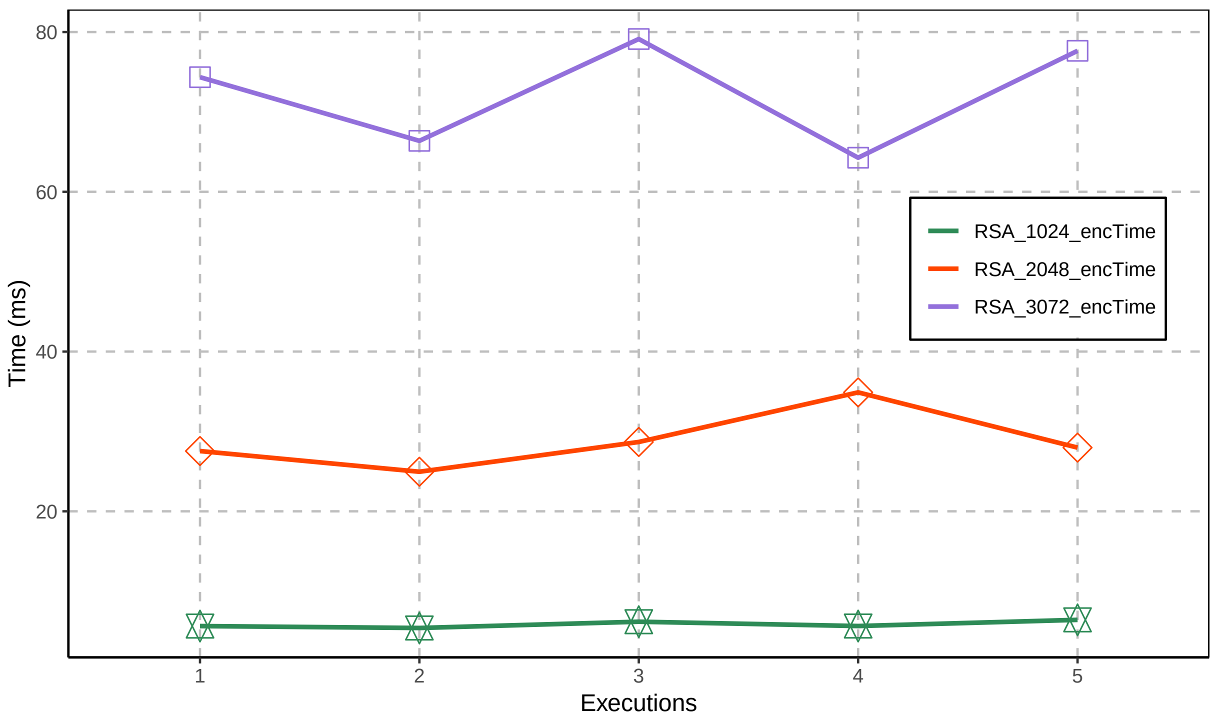


Figure 4.5: Execution time in milliseconds for different key size in RSA algorithm.

The green line at the bottom of the graph represents the 1024-bit key, which consistently shows the fastest execution time, averaging around 5 milliseconds. This key size shows the most uniform and stable performance compared to the others. The orange line, corresponding to the 2048-bit key, shows execution times ranging from 25 and 35 milliseconds, being a moderate increment compared with the 1024-bit key. The purple line at the top represents the 3072-bit key, which has the longest and the most fluctuating execution times, ranging between 65 and 80 milliseconds, significantly higher compared to the rest of the smaller key sizes. These results clearly demonstrate the correlation between the key size and the execution time in RSA encryption. As the key size increases, so does the execution time, due to the increased computational complexity of larger keys. Specifically, doubling the key size from 1024 to 2048 bits results in a fivefold increase in execution time, while increasing the key size from 2048 to 3072 bits leads to a 60% increase in execution time.

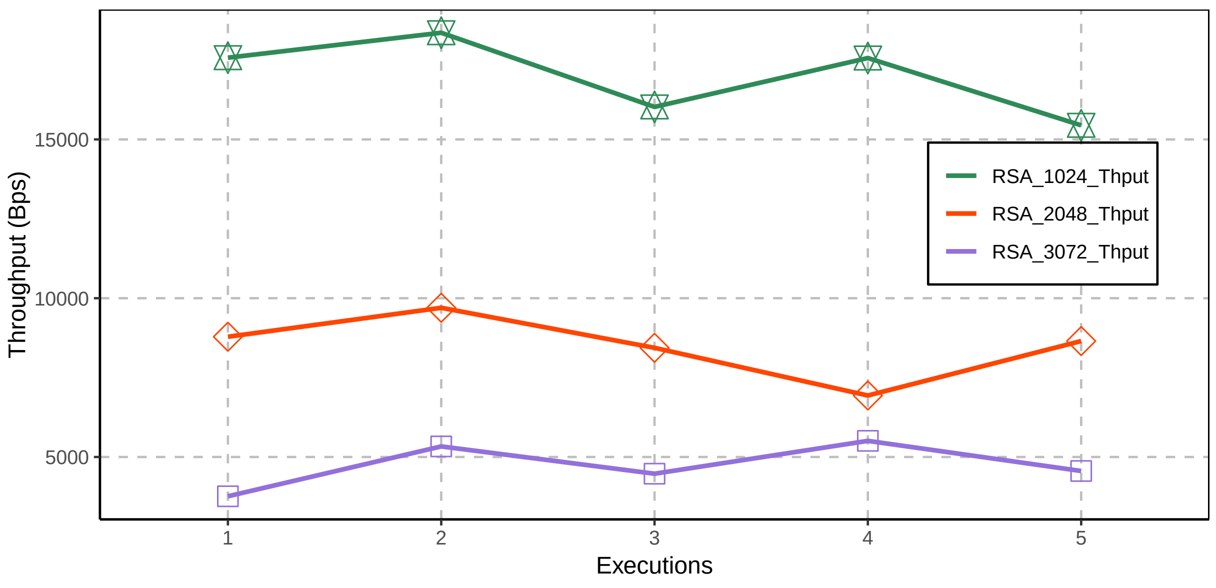


Figure 4.6: Throughput in bytes for different key size in RSA algorithm.

Figure 4.6 presents the throughput values in bytes for different key sizes in the RSA algorithm. As a result, the same lines seen previously are obtained but organized in a different way. First, the green line for the 1024-bit key shows the highest throughput, whit values ranging from approximately 15000 and 18000 bytes. In the center of the graph, the orange line representing the 2048-bit key shows average throughput with a slightly wider range, varying between 8000 and 10000 bytes. Finally, the purple line represents the 3072-bit key, showing the lowest throughput, with a range between 4000 and 6000 bytes. As in Figure 4.5, an inverse relationship between key size and throughput. This is because the amount of data that can be encrypted in a given period of time is directly affected by the key size. In other words, longer keys such as 2048 and 3072 bit keys take longer to process each operation compared to 1024-bit key. Therefore, performance decreases dramatically as the key size increases.

Figure 4.7 shows the memory consumption for the RSA algorithm. The purple line, representing the 3072-bit key, shows the highest memory consumption, remaining consistently around 80 MB. The orange line, associated with the 2048-bit key, presents a significantly lower memory usage, maintaining a steady value around 10 MB. Lastly, the green line for the 1024-bit key presents the lowest memory usage, barely visible at the bottom of the graph with values very close to the 2048-bit key. suggesting a highly efficient use of the memory for smaller keys.

In contrast to throughput and execution time, memory usage remains remarkably stable across all executions for each key size. However, the disparity in memory usage between the different key sizes is substantial. The 3072-bit key requires approximately eight times more memory than the 2048-bit key, which in turn uses slightly more memory than the 1024-bit key. These results are expected, given that RSA keys rely on modular exponentiation operations, which demand additional storage to manage large numbers. As key sizes increases, so too the memory required to store these large numbers and perform the corresponding operations. When moving to 2048 and 3072 bit keys, memory consumption increases significantly due to the need for more space, not only for storing the keys but also for handling intermediate operations, data buffers and mathematical structures during encryption and decryption.

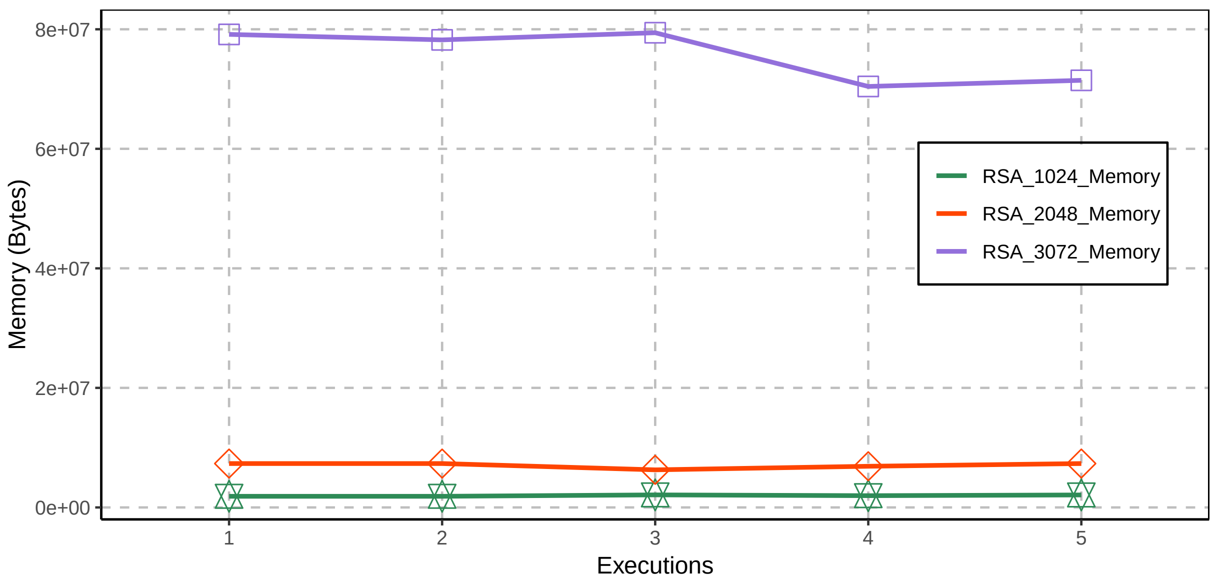


Figure 4.7: Use of memory in bytes for different key size in RSA algorithm.

Figure 4.8 shows results of the CPU usage for different RSA key sizes. Herein, the difference in CPU usage between the 3072-bit and 2048-bit keys is less pronounced compared to the other recorded metrics. This suggest that the computational demand on encryption and description begins to significantly impact the processor starting at the 2048-bit key size. In scenarios involving resources constrained devices, such as an FPGA or an IoT device, performance could decrease, and the CPU usage might exceed the values observed here. Ultimately, the text size used is limited by the key length, underscoring that even relatively small text encryption with the RSA requires substantial computational resources.

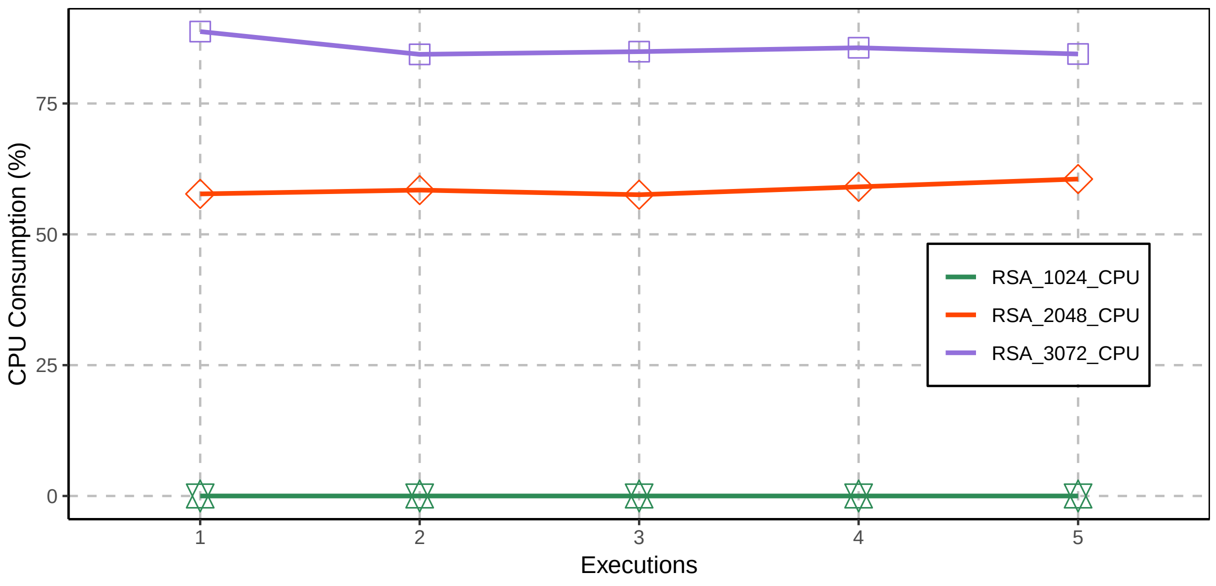


Figure 4.8: CPU consumption percentage for different key size in RSA algorithm.

4.3 Evaluation of ECC algorithm for different elliptic curves

In the analysis of encryption using elliptic curves, different text sizes measured in bytes were again considered: 446, 1100, 2142, 2972 and 3686. This process is essential to visualize the performance of the algorithm when using elliptic curves with different characteristics. In addition, the encryption and decryption process was carried out by combining a symmetric algorithm with the ECC algorithm. For this case, the AES algorithm was used, which was mentioned and analyzed previously.

When analyzing Figure 4.9, an overlap in the execution times of each curve is observed. Curve E-1174, represented by the green line, shows a slightly higher execution time for small text sizes compared to the other curves. However, for texts larger than 3968 bytes, the recorded time places it in an intermediate position. In contrast, the M-221 curve, represented by the blue line, starts with the lowest execution time for small texts and shows a constant growth, allowing it to obtain the shortest time when processing larger text sizes. On the other hand, the P-256 curve, represented by the orange line, starts with a time similar to that of the M-221, but shows a steeper growth, especially for large texts, resulting in the longest execution time.

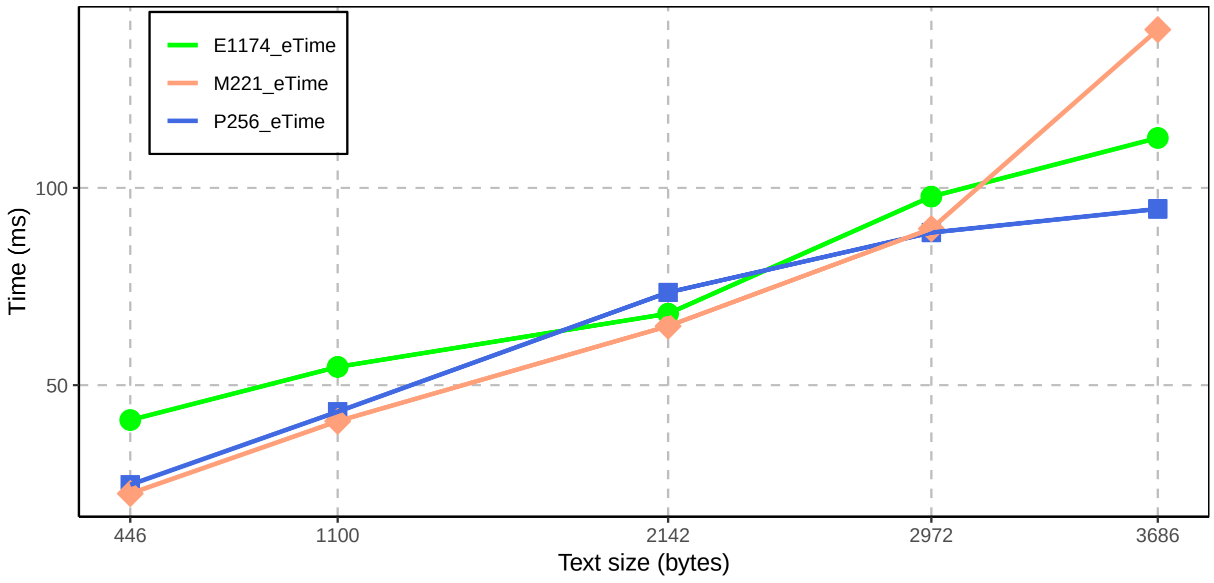


Figure 4.9: Execution time in milliseconds for different elliptic curves in ECC algorithm.

Referring to Figure 4.10, the line representing the E-1174 curve shows the highest performance, growing consistently for all text sizes. Its performance is comparable to the maximum value recorded for the P-256 curve, which starts with the lowest performance. As the input data size increases, its performance grows steadily, outperforming the M-221 curve for larger text sizes. The P-256 curve exhibits an interesting behavior, as its performance increases progressively up to about 3500 bytes, but then experiences a significant drop for a text size of 3686 bytes. These observations suggest that the choice of the elliptic curve may depend on the size of the data to be encrypted. At first glance, the E-1174 curve seems to be the optimal choice for applications that require high and constant performance, while the P-256 curve might be more suitable for applications that handle variable text sizes and require shorter execution times.

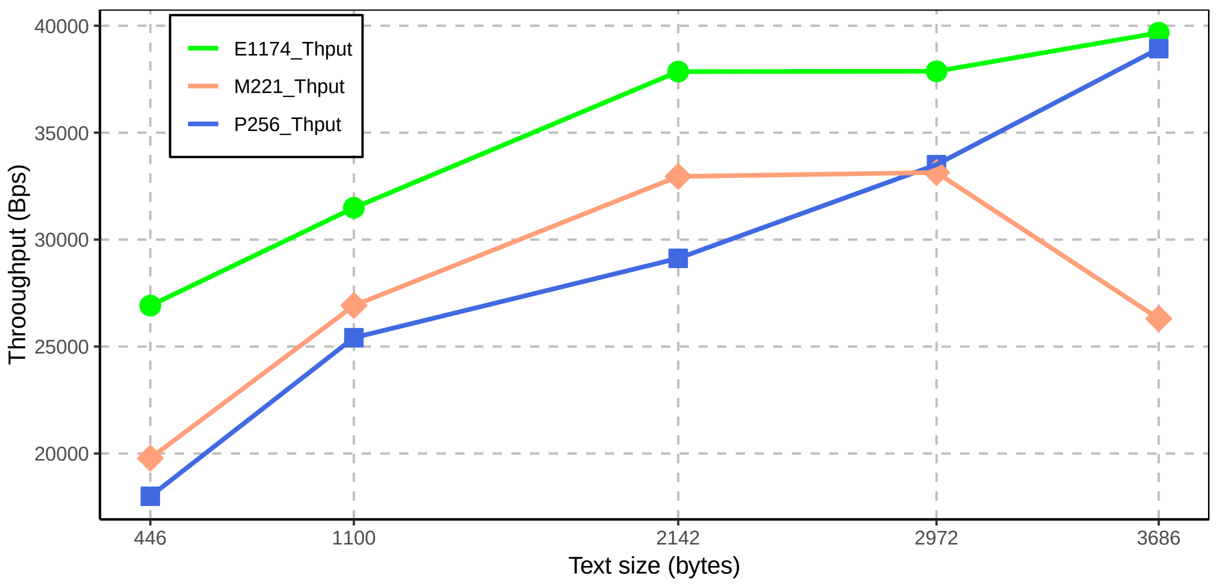


Figure 4.10: Throughput in bytes for different elliptic curves in ECC algorithm.

The next metric to evaluate elliptic curves is memory consumption. In Figure 4.11, the green line representing the E-1174 curve shows unstable behavior, reaching its minimum value with data sizes of 446 bytes and increasing up to 1100 bytes. However, a noticeable drop in memory consumption is observed when processing data of 2142 bytes, which may be related to the factors discussed in Figures: 4.1, 4.2, and 4.3. On the other hand, the M-221 and P-256 curves present a consistent pattern, with a progressive growth in memory consumption as the data size increases. It is important to consider that consumption values may be influenced by external factors related to the work environment.

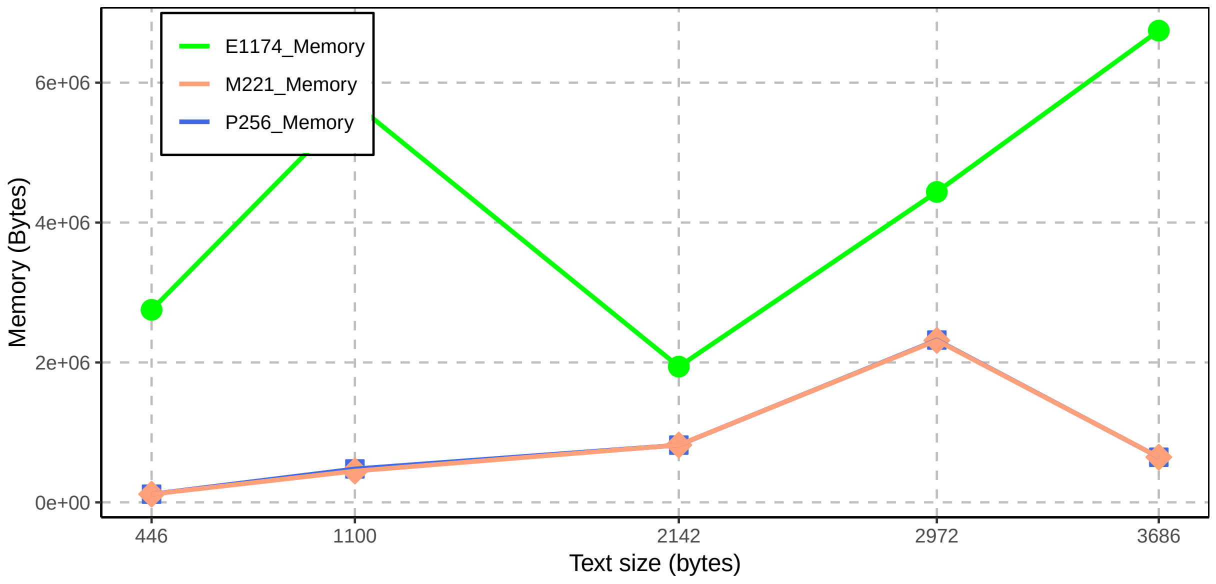


Figure 4.11: Memory consumption in bytes for different elliptic curves in ECC algorithm.

The P-256 curve stands out for its efficient use of memory, which explains its optimal performance in speed and throughput, as seen in Figures 4.5 and 4.6. Thus, the choice of the elliptic curve for a specific application must consider not only performance, but also memory consumption. Both the P-256 and M-221 curves are the most efficient in terms of memory, making them ideal choices for embedded devices or applications with limited resources. The results of the ECC algorithm can vary significantly between the E-1174, M-221 and P-256 curves due to their design and optimization differences. The E-1174 curve offers the highest throughput and the lowest memory consumption, being suitable for devices with more resources. In contrast, the M-221 curve, although exhibiting lower throughput, strikes a good balance between performance and memory consumption, making it suitable for applications requiring a combination of security and efficiency [199]. Meanwhile, the P-256 curve, despite having the lowest throughput, is widely used in critical applications due to its high level of security [200].

Finally, when analyzing the percentage of CPU consumption as a function of the input text length, Figure 4.12 shows that the implementation using the M-221 curve exerts less load on the processor, peaking at approximately 45%. In contrast, the version employing the P-256 curve exhibits the highest values, around 58%. However, all the values in the graph seem to converge, suggesting that if they continued to grow, the three lines could merge into one.

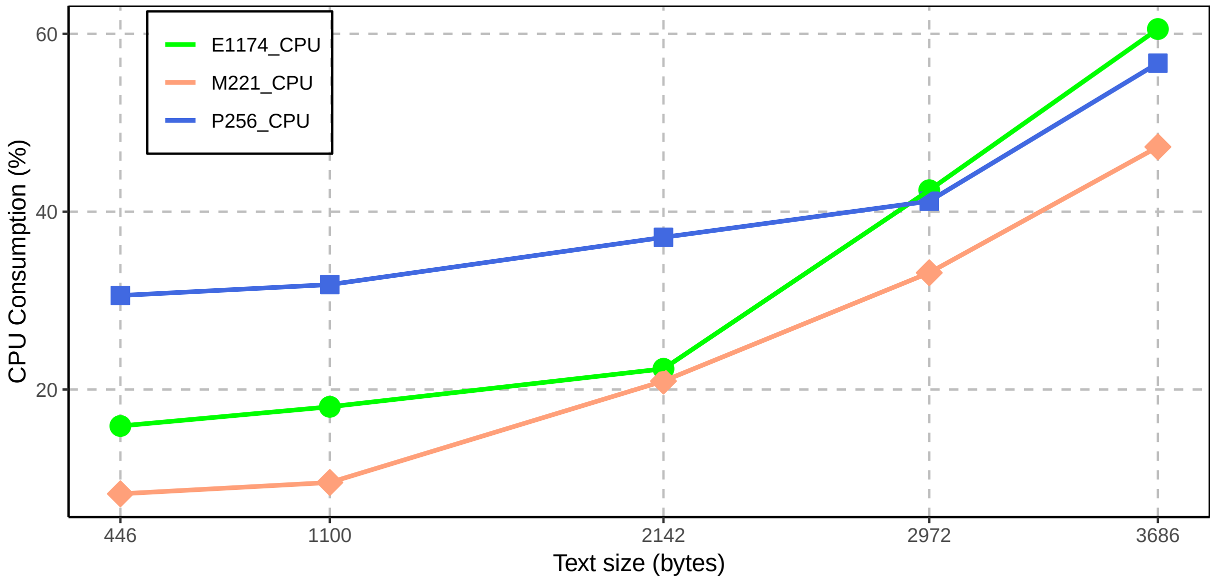


Figure 4.12: Percentage of CPU consumption for different elliptic curves in ECC algorithm.

4.4 Discussion of the results

To compare the algorithms, two fundamental requirements were established: to select the most optimal version of each encryption algorithm, either the one with the lowest or highest values according to the metric, and to define a specific input data size for testing. This last requirement is justified because the input size of the RSA algorithm is determined by the selected key size, which implies that all algorithms must be evaluated with the same amount of data. This approach facilitates a clear visualization of the performance of the RSA algorithm relative to each other and allows identifying the encryption algorithm with the best overall performance. For this analysis, the ECB and CBC encryption modes of the AES algorithm, the 3072-bit key of RSA, and the P-256 and M-221 curves of ECC were chosen. Comparing RSA and ECC with the best version of AES provides a comprehensive view on how these algorithms can work together and which one is best suited for different security and performance needs in various applications.

In Figures 4.13 and 4.14, the CBC mode of AES has been considered as the most efficient version, comparing it with the P-256 curve of ECC and the 3072-bit version of RSA. The results show that all algorithms maintain almost constant values during the tests. In the upper part, the values of the CBC mode are the highest compared to RSA and ECC. Although the CBC mode is the fastest version of AES, in the lower part, the P-256 curve of ECC presents the lowest values in the graph. According to the articles mentioned in previous sections and what is observed in Figure 4.13, the ECC algorithm is expected to have the lowest execution time. The values obtained from the ECC algorithm are derived from the combination of key creation with the encryption and decryption processes of the symmetric AES algorithm. Therefore, it is concluded that ECC is more efficient in terms of execution time and that its combination with AES improves the performance of the latter.

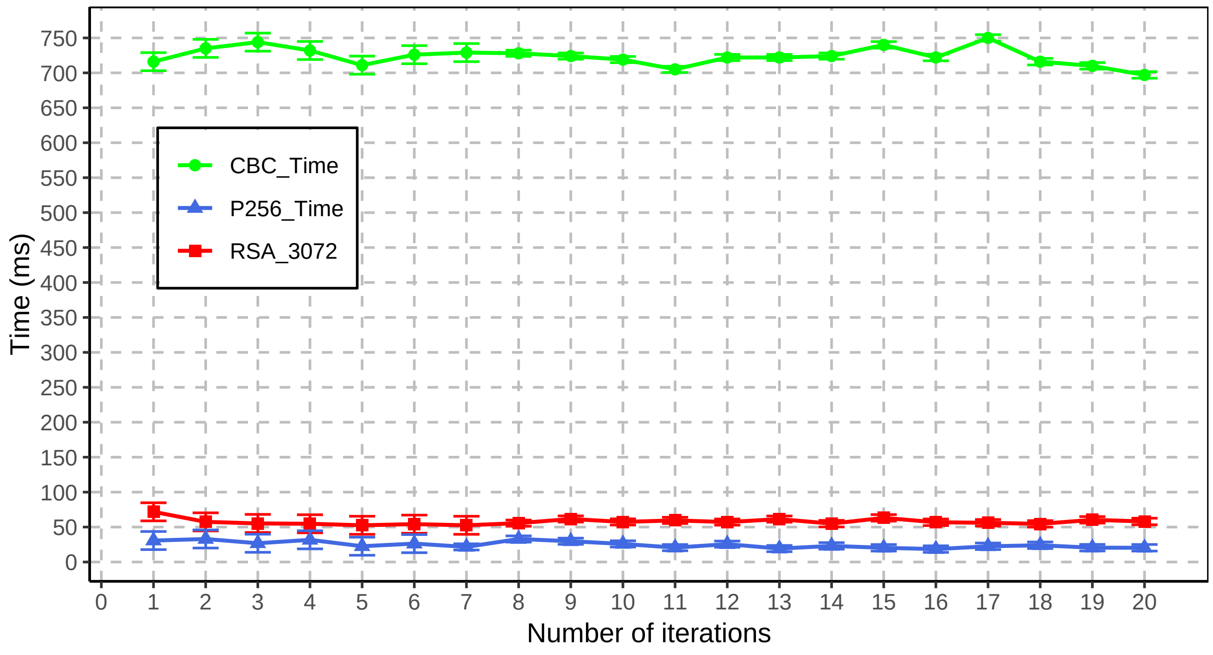


Figure 4.13: Execution time comparison between CBC cipher mode from AES, 3072-bit key from RSA, and the elliptic curve P-256 from ECC.

A similar behavior is observed in the throughput metric. In this case, the CBC mode of operation of AES, the 3072-bit version of RSA, and the ECC system with the P-256 elliptic curve were selected. Unlike the execution time values, where lower values are sought to define the optimal version of the algorithm, the throughput values must be higher. This indicates the capacity of the algorithm to process large or small volumes of data per second. In Figure 4.14, a more irregular trend is evident in all the lines, with the most oscillating and highest values corresponding to the ECC algorithm. In the lower part, the values of the CBC mode of the AES algorithm are presented. These results demonstrate the influence of the algorithm's architecture on its performance. For ECC, its high throughput values are due to the fact that it requires smaller keys, which reduces the number of operations required for encryption or decryption. As a result, the algorithm can process a larger amount of data in less time, as evidenced in papers such as [121]. Regarding the runtime analysis, the ECC was combined with the AES encryption process, allowing for an optimization in data processing per unit of time.

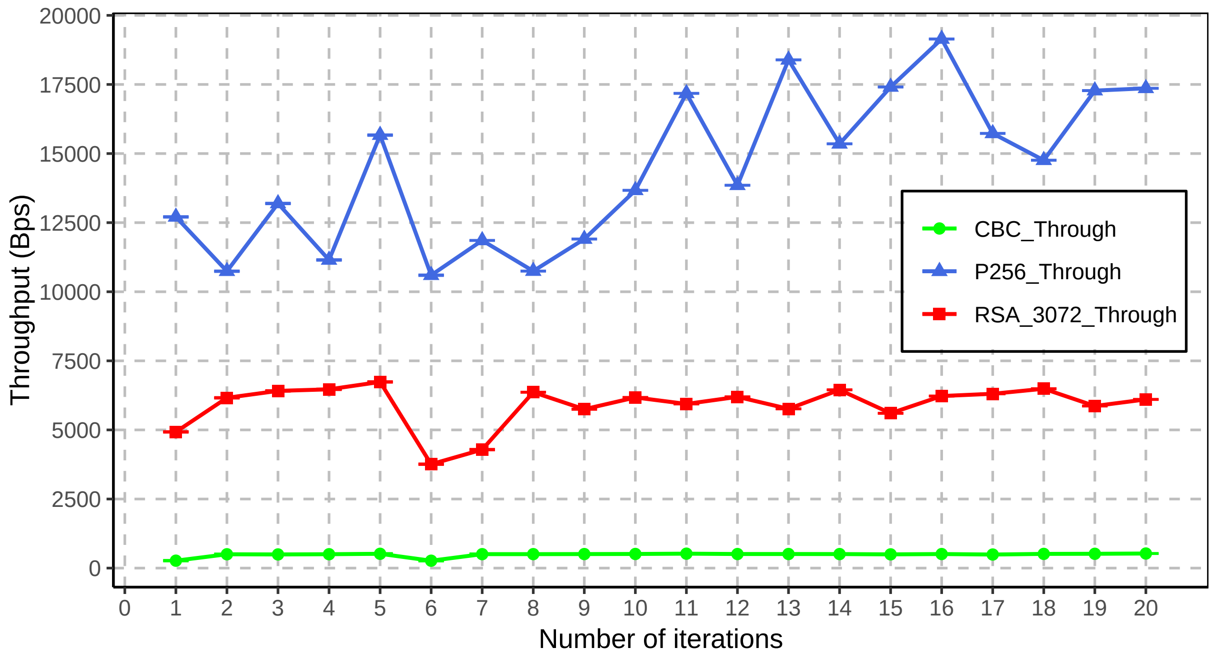


Figure 4.14: Comparison of throughput among CBC cipher mode from AES, 3072-bit key from RSA, and the elliptic curve P-256 from ECC.

In the analysis of the energy or memory consumption in bytes, the same versions of the algorithms have been used. However, for the AES algorithm, the ECB mode of operation was selected, since the processes and operations required by this mode do not demand as much space as the CBC mode. Thus, in Figure 4.15, an almost uniform trend is observed for all algorithms, with the exception of the RSA algorithm. In this case, an increase in memory consumption can be noticed between iterations 8 and 10, which could be attributed to the execution environment, since the memory was not freed in time when capturing this value. Furthermore, as evidenced in previously analyzed articles and in the structure of the RSA algorithm analysis, the highest energy consumption values correspond to RSA, while the lowest values are associated with ECC. This is due to the number of operations and the excessive length of the keys used in RSA. All of this is presented without considering the additional memory that would be necessary to generate the parameters that define each key generation process.

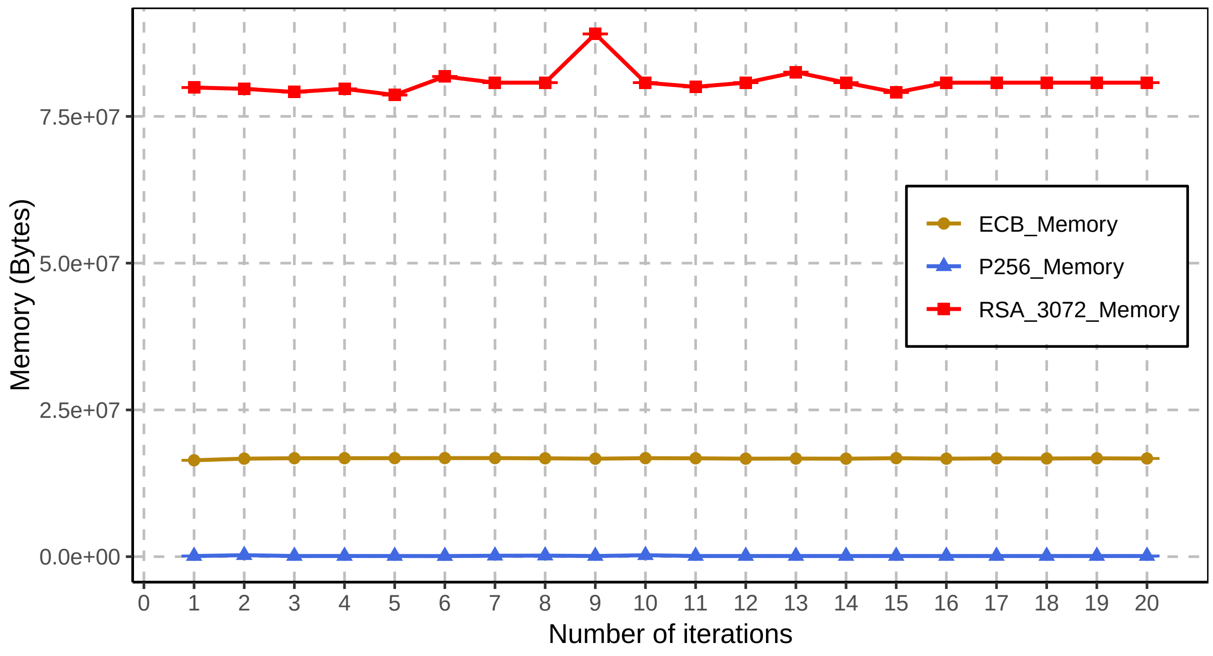


Figure 4.15: Comparison of use of memory between ECB cipher mode from AES algorithm and the curve P-256 from ECC algorithm.

The most intriguing case is observed in the CPU metric, where the error associated with each obtained value is most evident. For this measurement, the ECB mode of operation of AES, the 3072-bit key length of RSA and the ECC with the M-221 elliptic curve have been considered. Furthermore, it is observed that the error associated with each value progressively decreases as the tests progress. This can be attributed to the constant execution time and the background processes running on the PC, which can interfere with an accurate analysis of the CPU. Nevertheless, an almost uniform relationship is observed between all the values. Thus, in Figure 4.16, a less uniform trend is evident compared to the memory consumption metric. The highest CPU consumption values correspond to the RSA algorithm, reaching almost full processor utilization. On the contrary, the lowest values belong to the ECC algorithm, which approach zero. This significant difference arises from the inherent architecture of both algorithms. The power required to manipulate large numbers and perform complex operations is considerable, which could be suboptimal if implemented on resource-constrained devices, such as in the case of IoT.

Therefore, when considering all the results and after comparing the performance of each algorithm in the same scenario, the superiority of the ECC algorithm over the others is evident. Also, although the RSA algorithm has proven to be secure due to the complexity of its operations, it requires a considerable amount of resources to carry out these tasks.

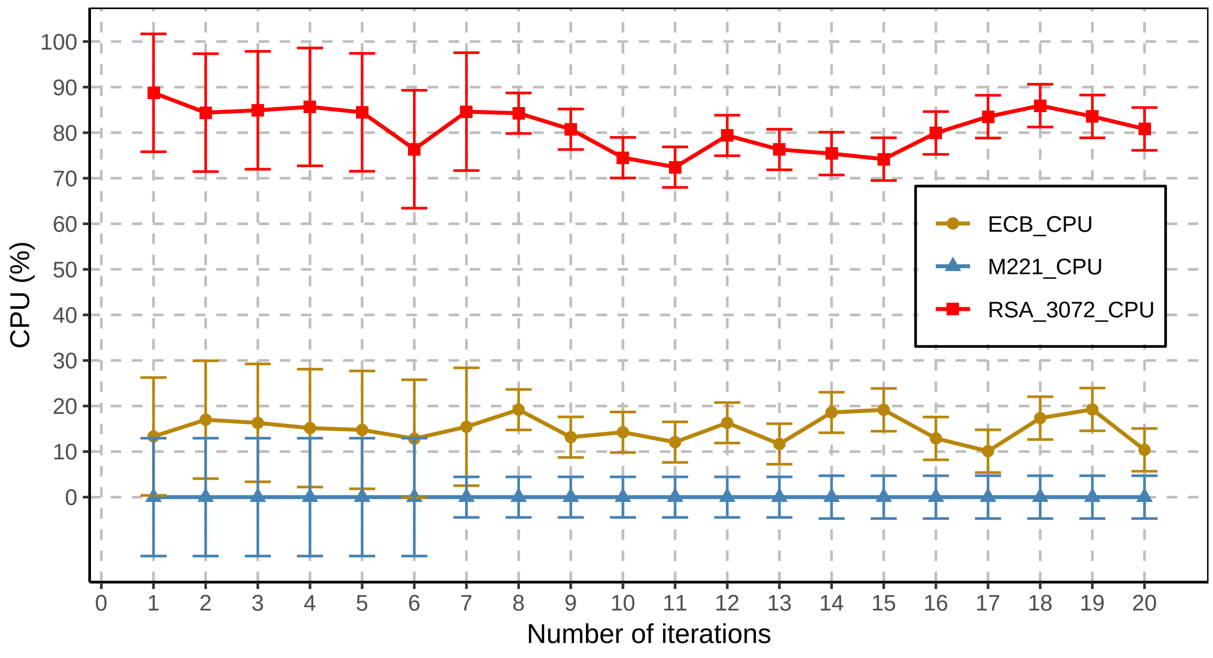


Figure 4.16: Comparison of CPU consumption between ECB cipher mode from AES algorithm and the curve M-221 from ECC algorithm.

Chapter 5

Conclusions

This chapter presents the conclusions derived from the evaluation of the cryptographic encryption algorithms analyzed in this thesis. Throughout the research, various algorithms were compared to identify which of them offers a better performance in different usage scenarios. Exhaustive tests were carried out to obtain a comprehensive view of their performance. In addition, graphical interfaces were implemented that facilitated the understanding of the internal processes of each algorithm, allowing a clearer evaluation of the differences and similarities between them. The conclusions presented below summarize the most relevant findings and offer a critical perspective on the applicability of the algorithms in real environments.

1. Both individual and overall results indicate that the asymmetric ECC encryption algorithm is more efficient in terms of time and resource consumption, and also shows a higher capacity for processing blocks of information compared to the symmetric AES encryption algorithm and its RSA equivalent. This was confirmed by tests with different text sizes, where the ECC algorithm consistently yielded lower values in all evaluated metrics. Furthermore, when comparing the overall performance of all algorithms, it was observed that the ECC model has a greater potential for handling large volumes of information simultaneously.

2. In terms of efficiency, the experimental results revealed that the RSA algorithm has significant limitations in terms of resource and time consumption. One notable point is the high CPU consumption, which reached approximately 80% when using a 3072 bit key. This high resource demand suggests that using the RSA algorithm on low-end devices might not be ideal, as it could accelerate battery drain and cause overheating, which could lead to both hardware and software failures.

3. The results obtained for each algorithm highlight that architecture is a crucial factor for both performance and security. As pointed out in recent research, certain structural aspects of these algorithms need to be fine-tuned to optimize their performance. A higher level of complexity in an algorithm usually translates into greater security, although this can lead to a decrease in its performance. In the case of the RSA algorithm, the encryption and decryption process is based on modular multiplication; however, the need to find extremely large prime values and the difficulty of performing attacks that attempt to factor these numbers contribute to RSA remaining a secure and relevant algorithm today, despite its challenges in terms of efficiency.

4. A graphical Java interface was successfully developed for each of the encryption algorithms. These interfaces not only allow selecting between different versions of the same algorithm, but also facilitate the visualization of the encryption key generation process, as well as the encryption and decryption procedures. In addition,

each interface was individually tested to ensure that no collisions occurred in the encryption processes, ensuring the integrity and reliability of the system. This approach improves the understanding of how the algorithms work and provides users with an interactive tool to experiment with different encryption methods.

In conclusion, the analysis has highlighted the advantages of the asymmetric ECC algorithm in efficiency and capacity, recognizing that AES, as a symmetric algorithm, is not directly comparable with RSA and ECC. The importance of the algorithms' architecture was emphasized and graphical interfaces in Java were developed to facilitate their understanding.

5.1 Future Work

Throughout this research, the importance of the performance of encryption algorithms and their ability to modify the format of the processed data has been examined. Based on these findings, the following lines of work are suggested for the future:

1. Implementation of algorithms in FPGAs: Investigate the feasibility of implementing more robust encryption algorithms in FPGA devices. This would allow taking advantage of the flexibility and parallelism options offered by these devices, thus addressing problems associated with RSA encryption, such as slow processing and high energy consumption.

2. Optimization for IoT devices: Implement these algorithms in hardware with limited computational resources, such as IoT devices. Along with evaluating their performance, it will be essential to design cryptographic systems that require less processing power, energy and memory, while ensuring that they maintain the same level of security as on more powerful devices.

3. Impact of quantum computing: Explore the consequences and challenges that quantum computing poses for current encryption algorithms. It is suggested to implement symmetric and asymmetric algorithms using post-quantum computing techniques and analyze their performance. This approach would facilitate the adaptation of algorithms to improve their resistance to possible quantum attacks.

These proposals will allow further exploration of the security capabilities that cryptographic systems can offer and to optimize their performance on lower-capacity devices.

Bibliography

[1] E. Antal, P. Zajac, and J. Mírka, "Solving a mystery from the thirty years' war: Karel rabenhaupt ze such´ e's encrypted letter to landgravine amalie elisabeth," in *International Conference on Historical Cryptology*, 2021, pp. 12–24.

[2] C. Christensen, "Polish mathematicians finding patterns in enigma messages," *Mathematics Magazine*, vol. 80, no. 4, pp. 247–273, 2007.

[3] M. A. Baba, A. Yusuf, A. Ahmad, and L. Maijama'a, "Performance analysis of the encryption algorithms as solution to cloud database security," *International Journal of Computer Applications*, vol. 975, p. 8887, 2014.

[4] A. Biryukov and C. De Cannière, "Data encryption standard (des)," *Encyclopedia of cryptography and security*, pp. 295–301, 2011.

[5] J. P. ChristofPaar and B. Preneel, "Understanding cryptography: A textbook for students and ractitioners," *Springer*, 2010.

[6] H. Hoomod and A. Radi, "New secure e-mail system based on bio-chaos key generation and modified aes algorithm," *Journal of Physics: Conference Series*, vol. 1003, p. 012025, 05 2018.

[7] S. S. Ali and D. Mukhopadhyay, "A differential fault analysis on aes key schedule using single fault," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2011, pp. 35–42.

[8] S. Srisakthi and A. Shanthi, "Towards the design of a stronger aes: Aes with key dependent shift rows (kdsr)," *Wireless Personal Communications*, vol. 114, pp. 3003–3015, 2020.

[9] P. Telagarapu, B. Biswal, and V. S. Guntuku, "Design and analysis of multimedia communication system," in *2011 Third International Conference on Advanced Computing*. IEEE, 2011, pp. 193–197.

[10] P. Bhatia and R. Sumbaly, "Framework for wireless network security using quantum cryptography," *arXiv preprint arXiv:1412.2495*, 2014.

[11] D. Blazhevski, A. Bozhinovski, B. Stojchevska, and V. Pachovski, "Modes of operation of the aes algorithm," 2013.

[12] A. Zohair, "What is cbc?" https://www.educative.io/answers/what-is-cbc, 2024, educative.

[13] H. Li and Y. Wang, "The history of cryptography and its applications," *International Journal of Social Science and Education Research*, vol. 5, no. 3, pp. 343–349, 2022.

[14] S. Bhattacharya, "Cryptology and information security-past, present, and future role in society," *International Journal on Cryptography and Information Security (IJCIS)*, vol. 9, no. 1/2, pp. 13–39, 2019.

[15] F. Maqsood, M. Ahmed, M. M. Ali, and M. A. Shah, “Cryptography: a comparative analysis for modern techniques,” *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, 2017.

[16] M. M. Yahaya and A. Ajibola, “Cryptosystem for secure data transmission using advance encryption standard (aes) and steganography,” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, vol. 5, no. 6, pp. 317–322, 2019.

[17] C. Fu and Z.-l. Zhu, “An efficient implementation of rsa digital signature algorithm,” in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2008, pp. 1–4.

[18] H. Rezaeighaleh and C. C. Zou, “New secure approach to backup cryptocurrency wallets,” in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[19] S. Rawal, “Advanced encryption standard (aes) and it’s working,” *International Research Journal of Engineering and Technology*, vol. 3, no. 8, pp. 1165–1169, 2016.

[20] N. Aleisa, “A comparison of the 3des and aes encryption standards,” *International Journal of Security and Its Applications*, vol. 9, no. 7, pp. 241–246, 2015.

[21] A. H. Awlla and S. M. A. Aziz, “Secure device to device communication for 5g network based on improved aes,” *The Scientific Journal of Cihan University–Sulaimaniya*, vol. 5, no. 1, pp. 57–67, 2021.

[22] S. N. Mendonca, “Data security in cloud using aes,” *Int. J. Eng. Res. Technol*, vol. 7, 2018.

[23] L. Zou, M. Ni, Y. Huang, W. Shi, and X. Li, “Hybrid encryption algorithm based on aes and rsa in file encryption,” in *Frontier Computing: Theory, Technologies and Applications (FC 2019) 8*. Springer, 2020, pp. 541–551.

[24] O. G. Abood and S. K. Guirguis, “A survey on cryptography algorithms,” *International Journal of Scientific and Research Publications*, vol. 8, no. 7, pp. 495–516, 2018.

[25] T. S. Obaid, “Study a public key in rsa algorithm,” *European Journal of Engineering and Technology Research*, vol. 5, no. 4, pp. 395–398, 2020.

[26] S. U. Nimbhorkar and L. Malik, “A survey on elliptic curve cryptography (ecc),” *International Journal of Advanced Studies in Computers, Science and Engineering*, vol. 1, no. 1, pp. 1–5, 2012.

[27] ——, “A survey on elliptic curve cryptography (ecc),” *International Journal of Advanced Studies in Computers, Science and Engineering*, vol. 1, no. 1, pp. 1–5, 2012.

[28] S. Mohammadi and S. Abedi, “Ecc-based biometric signature: A new approach in electronic banking security,” in *2008 International Symposium on Electronic Commerce and Security*. IEEE, 2008, pp. 763–766.

[29] B. Nair and C. Mala, “Analysis of ecc for application specific wsn security,” in *2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*. IEEE, 2015, pp. 1–6.

[30] L. Marin, M. Piotr Pawlowski, and A. Jara, “Optimized ecc implementation for secure communication between heterogeneous iot devices,” *Sensors*, vol. 15, no. 9, pp. 21 478–21 499, 2015.

[31] J. N. Ugwu, S. A. Mogaji, S. E. Akinsanya, J. O. Awoyemi, and J. C. Obi, "Comparative analysis of different multilevel states for four notable cryptographic schemes," *Researchers Journal of Science and Technology*, vol. 4, no. 2, pp. 60–77, 2024.

[32] S. S. Dhanda, B. Singh, and P. Jindal, "Lightweight cryptography: a solution to secure iot," *Wireless Personal Communications*, vol. 112, no. 3, pp. 1947–1980, 2020.

[33] S. D. Rihan, A. Khalid, and S. E. F. Osman, "A performance comparison of encryption algorithms aes and des," *International Journal of Engineering Research & Technology (IJERT)*, vol. 4, no. 12, pp. 151–154, 2015.

[34] D. Mahto and D. K. Yadav, "Rsa and ecc: A comparative analysis," *International journal of applied engineering research*, vol. 12, no. 19, pp. 9053–9061, 2017.

[35] B. S. Kumar, V. R. Raj, and A. Nair, "Comparative study on aes and rsa algorithm for medical images," in *2017 international conference on communication and signal processing (ICCSP)*. IEEE, 2017, pp. 0501–0504.

[36] K. Gupta and S. Silakari, "Ecc over rsa for asymmetric encryption: A review," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 3, p. 370, 2011.

[37] D. Davies, "A brief history of cryptography," *Information Security Technical Report*, vol. 2, no. 2, pp. 14–17, 1997.

[38] J. F. Dooley, *A brief history of cryptology and cryptographic algorithms*. Springer, 2013, vol. 21.

[39] C. Bauer, *Secret history: The story of cryptology*. Chapman and Hall/CRC, 2021.

[40] S. A. Hannan and A. M. A. M. Asif, "Analysis of polyalphabetic transposition cipher techniques used for encryption and decryption," *International Journal of Computer Science and Software Engineering*, vol. 6, no. 2, p. 41, 2017.

[41] T. M. Damico, "A brief history of cryptography," *Inquiries Journal*, vol. 1, no. 11, 2009.

[42] W. A. Kotas, "A brief history of cryptography," 2000.

[43] A. M. Qadir and N. Varol, "A review paper on cryptography," in *2019 7th international symposium on digital forensics and security (ISDFS)*. IEEE, 2019, pp. 1–6.

[44] M. Rathidevi, R. Yaminipriya, and S. Sudha, "Trends of cryptography stepping from ancient to modern," in *2017 International Conference on Innovations in Green Energy and Healthcare Technologies (IGEHT)*. IEEE, 2017, pp. 1–9.

[45] A. Mackenzie, "Undecidability: The history and time of the universal turing machine," *Configurations*, vol. 4, no. 3, pp. 359–379, 1996.

[46] R. E. Klima, R. Klima, N. P. Sigmon, and N. Sigmon, *Cryptology: classical and modern*. Chapman and Hall/CRC, 2018.

[47] E. Antal and V. Hromada, "A new stream cipher based on fialka m-125," *Tatra Mountains Mathematical Publications*, vol. 57, no. 1, pp. 101–118, 2013.

[48] M. N. Alenezi, H. Alabdulrazzaq, and N. Q. Mohammad, "Symmetric encryption algorithms: Review and evaluation study," *International Journal of Communication Networks and Information Security*, vol. 12, no. 2, pp. 256–272, 2020.

[49] H. Madushan, I. Salam, and J. Alawatugoda, "A review of the nist lightweight cryptography finalists and their fault analyses," *Electronics*, vol. 11, no. 24, p. 4199, 2022.

[50] A. Menezes and D. Stebila, "The advanced encryption standard: 20 years later," *IEEE Security & Privacy*, vol. 19, no. 6, pp. 98–102, 2021.

[51] A. G. Konheim, "Horst feistel: the inventor of lucifer, the cryptographic algorithm that changed cryptology," *Journal of Cryptographic Engineering*, vol. 9, no. 1, pp. 85–100, 2019.

[52] H. Eberle, "A high-speed des implementation for network applications," in *Annual International Cryptology Conference*. Springer, 1992, pp. 521–539.

[53] K. Jones, "Who's who in aes?" 2001.

[54] M. E. Smid, "Development of the advanced encryption standard," *Journal of Research of the National Institute of Standards and Technology*, vol. 126, 2021.

[55] M. Dworkin, "Second advanced encryption standard candidate conference rome, italy," *Journal of Research of the National Institute of Standards and Technology*, vol. 104, no. 4, 1999.

[56] M. Savari and M. Montazerolzohour, "All about encryption in smart card," in *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*. IEEE, 2012, pp. 54–59.

[57] G. Keating, "Performance analysis of aes candidates on the 6805 cpu core," in *Proceedings of the second AES Candidate Conference*, 1999, pp. 109–114.

[58] M. U. Kamaluddin, S. Shahbudin, N. M. Isa, and H. Z. Abidin, "Teaching the intel 8051 microcontroller with hands-on hardware experiments," in *2015 IEEE 7th International Conference on Engineering Education (ICEED)*. IEEE, 2015, pp. 100–105.

[59] N. Pohlmann, H. Reimer, W. Schneider, H. Handschuh, and E. Trichina, "High density smart cards: New security challenges and applications," in *ISSE/SECURE 2007 Securing Electronic Business Processes: Highlights of the Information Security Solutions Europe/SECURE 2007 Conference*. Springer, 2007, pp. 251–259.

[60] A. Hamza and B. Kumar, "A review paper on des, aes, rsa encryption standards," in *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*. IEEE, 2020, pp. 333–338.

[61] T. Jamil, "The rijndael algorithm," *IEEE potentials*, vol. 23, no. 2, pp. 36–38, 2004.

[62] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr, "Advanced encryption standard (aes)," 2001.

[63] P.-A. Fouque, J. Jean, and T. Peyrin, "Structural evaluation of aes and chosen-key distinguisher of 9-round aes-128," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I.* Springer, 2013, pp. 183–203.

[64] Y. Dodis, J. Katz, J. Steinberger, A. Thiruvengadam, and Z. Zhang, "Provable security of substitution-permutation networks," *Cryptology ePrint Archive*, 2017.

[65] P. Mellu and S. Mali, "Aes: Asymmetric key cryptographic systemǁ," *International Journal of Information Technology and Knowledge Management*, vol. 4, no. 1, pp. 113–117, 2011.

[66] N. Floissac and Y. L'Hyver, "From aes-128 to aes-192 and aes-256, how to adapt differential fault analysis attacks on key expansion," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2011, pp. 43–53.

[67] P. Derbez, P.-A. Fouque, and J. Jean, "Improved key recovery attacks on reduced-round aes in the single-key setting," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 371–387.

[68] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of finite fields*. Springer Science & Business Media, 2013, vol. 199.

[69] P. Deligne and G. Lusztig, "Representations of reductive groups over finite fields," *Annals of Mathematics*, vol. 103, no. 1, pp. 103–161, 1976.

[70] A. Weil, "Numbers of solutions of equations in finite fields," 1949.

[71] J. Großschädl and E. Savaş, "Instruction set extensions for fast arithmetic in finite fields gf (p) and gf (2 m)," in *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer, 2004, pp. 133–147.

[72] M. Albrecht, G. Bard, and W. Hart, "Efficient multiplication of dense matrices over gf (2)," *arXiv preprint arXiv:0811.1714*, 2008.

[73] A. Iliev and N. Kyurkchiev, "The faster extended euclidean algorithm," in *Collection of scientific works from conference*, 2018, pp. 21–26.

[74] M. Jing, Y. Chen, Y. Chang, and C. Hsu, "The design of a fast inverse module in aes," in *2001 International conferences on info-tech and info-net. Proceedings (Cat. No. 01EX479)*, vol. 3. IEEE, 2001, pp. 298–303.

[75] F. Huo and G. Gong, "Xor encryption versus phase encryption, an in-depth analysis," *IEEE Transactions on Electromagnetic Compatibility*, vol. 57, no. 4, pp. 903–911, 2015.

[76] P. Onions, "On the strength of simply-iterated feistel ciphers with whitening keys," in *Cryptographers' Track at the RSA Conference*. Springer, 2001, pp. 63–69.

[77] J. Takahashi, T. Fukunaga, and K. Yamakoshi, "Dfa mechanism on the aes key schedule," in *Workshop on fault diagnosis and tolerance in cryptography (FDTC 2007)*. IEEE, 2007, pp. 62–74.

[78] D. Canright, “A very compact s-box for aes,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 441–455.

[79] M. K. Misra, R. Mathur, and R. Tripathi, “A new encryption/decryption approach using aes,” in *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*. IEEE, 2021, pp. 1–5.

[80] M. Tyagi, M. Manoria, and B. Mishra, “Analysis and implementation of aes and rsa for cloud,” *International Journal of Applied Engineering Research*, vol. 14, no. 20, pp. 3918–3923, 2019.

[81] E. S. I. Harba, “Secure data encryption through a combination of aes, rsa and hmac,” *Engineering, Technology & Applied Science Research*, vol. 7, no. 4, pp. 1781–1785, 2017.

[82] D. Cardoso, D. Docherty, and Y. Paksoy, “Security & encryption rsa project,” 2022.

[83] C. McIvor, M. McLoone, and J. V. McCanny, “Modified montgomery modular multiplication and rsa exponentiation techniques,” *IEE Proceedings-Computers and Digital Techniques*, vol. 151, no. 6, pp. 402–408, 2004.

[84] H. Orman, *Encrypted Email: The History and Technology of Message Privacy*. Springer, 2015.

[85] W. H. Wong, “Timing attacks on rsa: revealing your secrets through the fourth dimension,” *XRDS: Crossroads, The ACM Magazine for Students*, vol. 11, no. 3, pp. 5–5, 2005.

[86] R. Minni, K. Sultania, S. Mishra, and D. R. Vincent, “An algorithm to enhance security in rsa,” in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 2013, pp. 1–4.

[87] E. Elgeldawi, M. Mahrous, and A. Sayed, “A comparative analysis of symmetric algorithms in cloud computing: a survey,” *International Journal of Computer Applications*, vol. 182, no. 48, pp. 7–16, 2019.

[88] V. Shukla, A. Mishra, and S. Agarwal, “A new one time password generation method for financial transactions with randomness analysis,” in *Innovations in Electrical and Electronic Engineering: Proceedings of ICEEE 2020*. Springer, 2021, pp. 713–723.

[89] D. S. Gunasekaran and M. Lavanya, “A review on enhancing data security in cloud computing using rsa and aes algorithms,” *Int J Adv Eng Res*, vol. 9, no. 4, pp. 1–7, 2015.

[90] E. Jintcharadze and M. Iavich, “Hybrid implementation of twofish, aes, elgamal and rsa cryptosystems,” in *2020 IEEE East-West Design & Test Symposium (EWDTS)*. IEEE, 2020, pp. 1–5.

[91] F. Mallouli, A. Hellal, N. S. Saeed, and F. A. Alzahrani, “A survey on cryptography: comparative study between rsa vs ecc algorithms, and rsa vs el-gamal algorithms,” in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2019, pp. 173–176.

[92] J. Hastad, “N using rsa with low exponent in a public key network,” in *Advances in Cryptology—CRYPTO’85 Proceedings 5*. Springer, 1986, pp. 403–408.

[93] L. S. Reddy, “Rm-rsa algorithm,” *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 25, no. 1, pp. 1–13, 2022.

[94] S. Sepahvandi, M. Hosseinzadeh, K. Navi, and A. Jalali, “An improved exponentiation algorithm for rsa cryptosystem,” in *2009 International Conference on Research Challenges in Computer Science*. IEEE, 2009, pp. 128–132.

[95] G. C. Marchesan, N. R. Weirich, E. C. Culau, I. I. Weber, F. G. Moraes, E. Carara, and L. L. de Oliveira, “Exploring rsa performance up to 4096-bit for fast security processing on a flexible instruction set architecture processor,” in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2018, pp. 757–760.

[96] M. A. Budiman and D. Rachmawati, “Using random search and brute force algorithm in factoring the rsa modulus,” *Data Science: Journal of Computing and Applied Informatics*, vol. 2, no. 1, pp. 45–52, 2018.

[97] C. Vuillaume, T. Endo, and P. Wooderson, “Rsa key generation: new attacks,” in *Constructive Side-Channel Analysis and Secure Design: Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings 3*. Springer, 2012, pp. 105–119.

[98] B. Fagin, “Composite numbers that give valid rsa key pairs for any coprime p,” *Information*, vol. 9, no. 9, p. 216, 2018.

[99] R. Shams, F. H. Khan, and M. Umair, “Cryptosystem an implementation of rsa using verilog,” *International Journal of Computer Networks and Communications Security*, vol. 1, no. 3, pp. 102–109, 2013.

[100] B. Zhao, L. Wang, K. Jiang, X. Liang, W. Shan, and J. Liu, “An improved power attack on small rsa public exponent,” in *2016 12th International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2016, pp. 578–581.

[101] H.-M. Sun, M.-E. Wu, W.-C. Ting, and M. J. Hinek, “Dual rsa and its security analysis,” *IEEE Transactions on Information Theory*, vol. 53, no. 8, pp. 2922–2933, 2007.

[102] V. A. Roman’kov, “New probabilistic public-key encryption based on the rsa cryptosystem,” *Groups Complexity Cryptology*, vol. 7, no. 2, pp. 153–156, 2015.

[103] H. R. Dorbidi, “A note on the coprime graph of a group,” *International Journal of Group Theory*, vol. 5, no. 4, pp. 17–22, 2016.

[104] M. J. Wiener, “Cryptanalysis of short rsa secret exponents,” *IEEE Transactions on Information theory*, vol. 36, no. 3, pp. 553–558, 1990.

[105] R. S. Dhakar, A. K. Gupta, and P. Sharma, “Modified rsa encryption algorithm (mrea),” in *2012 second international conference on advanced computing & communication technologies*. IEEE, 2012, pp. 426–429.

[106] N. Y. Goshwe, “Data encryption and decryption using rsa algorithm in a network environment,” *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 13, no. 7, p. 9, 2013.

[107] R.-J. Hwang, F.-F. Su, Y.-S. Yeh, and C.-Y. Chen, “An efficient decryption method for rsa cryptosystem,” in *19th International Conference on Advanced Information Networking and Applications (AINA’05) Volume 1 (AINA papers)*, vol. 1. IEEE, 2005, pp. 585–590.

[108] M. Gobi, R. Sridevi, and R. Rahini, "A comparative study on the performance and the security of rsa and ecc algorithm," *International journal of advanced network and application*, 2015.

[109] C. Mclvor, M. McLoone, and J. V. McCanny, "Fast montgomery modular multiplication and rsa cryptographic processor architectures," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 1. IEEE, 2003, pp. 379–384.

[110] D. Pei, A. Salomaa, and C. Ding, *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.

[111] V. Kumar, R. Kumar, and S. Pandey, "An enhanced and secured rsa public key cryptosystem algorithm using chinese remainder theorem," in *Smart and Innovative Trends in Next Generation Computing Technologies: Third International Conference, NGCT 2017, Dehradun, India, October 30-31, 2017, Revised Selected Papers, Part II 3*. Springer, 2018, pp. 543–554.

[112] S. S. Kumar, S. A. Prabhu, R. Durga, and S. Jeevitha, "A survey on various asymmetric algorithms," *International Research 7. Journal of Advanced Engineering and Science*, vol. 1, no. 4, pp. 117–119, 2016.

[113] Z. Vahdati, S. Yasin, A. Ghasempour, and M. Salehi, "Comparison of ecc and rsa algorithms in iot devices," *Journal of Theoretical and Applied Information Technology*, vol. 97, no. 16, p. 4293, 2019.

[114] D. Tarasick, D. Wardle, J. Kerr, J. Bellefleur, and J. Davies, "Tropospheric ozone trends over canada: 1980–1993," *Geophysical research letters*, vol. 22, no. 4, pp. 409–412, 1995.

[115] M. Y. Malik, "Efficient implementation of elliptic curve cryptography using low-power digital signal processor," in *2010 the 12th international conference on advanced communication technology (ICACT)*, vol. 2. IEEE, 2010, pp. 1464–1468.

[116] I. Connell, "Elliptic curve handbook," *McGill University*, 1999.

[117] W. Diffie, M. E. Hellman, and J. Ellis, "Public key cryptography," in *IEEE International Symposium on Information Theory*, 1976.

[118] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, 2022, pp. 365–390.

[119] ——, "Multiuser cryptographic techniques," in *Proceedings of the June 7-10, 1976, national computer conference and exposition*, 1976, pp. 109–112.

[120] V. Rudnytskyi, O. Korchenko, N. Lada, R. Ziubina, L. Wieclaw, and L. Hamera, "Cryptographic encoding in modern symmetric and asymmetric encryption," *Procedia Computer Science*, vol. 207, pp. 54–63, 2022.

[121] M. Suárez-Albela, P. Fraga-Lamas, and T. M. Fernández-Caramés, "A practical evaluation on rsa and ecc-based cipher suites for iot high-security energy-efficient fog and mist computing devices," *Sensors*, vol. 18, no. 11, p. 3868, 2018.

[122] M. A. Khan, M. T. Quasim, N. S. Alghamdi, and M. Y. Khan, "A secure framework for authentication and encryption using improved ecc for iot-based medical sensor data," *IEEe Access*, vol. 8, pp. 52 018–52 027, 2020.

[123] M. Ariffin and N. Abu, "-cryptosystem: A chaos based public key cryptosystem," *Int. Jour. Cryptology Research*, vol. 1, no. 2, pp. 149–163, 2009.

[124] J. Xie, P. K. Meher, M. Sun, Y. Li, B. Zeng, and Z.-H. Mao, "Efficient fpga implementation of low-complexity systolic karatsuba multiplier over $gf(2\{m\})$ based on nist polynomials," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 7, pp. 1815–1825, 2017.

[125] A. Menezes, "The elliptic curve discrete logarithm problem: State of the art." *IWSEC*, p. 218, 2008.

[126] F. Liu, "A tutorial on elliptic curve cryptography (ecc)," *Brandenburg Technical University of Cottbus: Computer Networking Group*, pp. 1–58, 2002.

[127] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis, "An rns implementation of an $f\_\{p\}$ elliptic curve point multiplier," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 6, pp. 1202–1213, 2008.

[128] S. Galovich, "The class group of a cyclic p-group," *Journal of Algebra*, vol. 30, no. 1-3, pp. 368–387, 1974.

[129] H. Corrigan-Gibbs and D. Kogan, "The discrete-logarithm problem with preprocessing," in *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*. Springer, 2018, pp. 415–447.

[130] D. Mahto, D. A. Khan, and D. K. Yadav, "Security analysis of elliptic curve cryptography and rsa," in *Proceedings of the world congress on engineering*, vol. 1, 2016, pp. 419–422.

[131] M. Müller-Olm and H. Seidl, "Analysis of modular arithmetic," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 29, no. 5, pp. 29–es, 2007.

[132] H. Debiao, C. Jianhua, and H. Jin, "An id-based client authentication with key agreement protocol for mobile client–server environment on ecc with provable security," *Information Fusion*, vol. 13, no. 3, pp. 223–230, 2012.

[133] G. Yuxuan, Z. Ling, and Y. Jing, "An efficient group key negotiation scheme based on ecc and key chains," in *2023 IEEE 6th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*. IEEE, 2023, pp. 1182–1189.

[134] M. R. Ashila, N. Atikah, E. H. Rachmawanto, C. A. Sari *et al.*, "Hybrid aes-huffman coding for secure lossless transmission," in *2019 Fourth International Conference on Informatics and Computing (ICIC)*. IEEE, 2019, pp. 1–5.

[135] P. Kumar and S. B. Rana, "Development of modified aes algorithm for data security," *Optik*, vol. 127, no. 4, pp. 2341–2345, 2016.

[136] L. Zhang, A. A. Ding, Y. Fei, and Z. H. Jiang, "Statistical analysis for access-driven cache attacks against aes," *Cryptology ePrint Archive*, 2016.

[137] T. Tiessen, L. R. Knudsen, S. Kölbl, and M. M. Lauridsen, "Security of the aes with a secret s-box," in *International Workshop on Fast Software Encryption*. Springer, 2015, pp. 175–189.

[138] M. Dubois and E. Filiol, "Hacking of the aes with boolean functions," *arXiv preprint arXiv:1609.03734*, 2016.

[139] C. Nandini and K. Rekha, "A review of aes and visual cryptographic techniques for added security," *Perspectives in Communication, Embedded-systems and Signal-processing-PiCES*, vol. 1, no. 2, pp. 6–7, 2017.

[140] Y. Liu, W. Gong, and W. Fan, "Application of aes and rsa hybrid algorithm in e-mail," in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*. IEEE, 2018, pp. 701–703.

[141] S. Rehman, N. Talat Bajwa, M. A. Shah, A. O. Aseeri, and A. Anjum, "Hybrid aes-ecc model for the security of data over cloud storage," *Electronics*, vol. 10, no. 21, p. 2673, 2021.

[142] L. K. Galla, V. S. Koganti, and N. Nuthalapati, "Implementation of rsa," in *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. IEEE, 2016, pp. 81–87.

[143] H. Marzouqi, M. Al-Qutayri, and K. Salah, "Review of elliptic curve cryptography processor designs," *Microprocessors and Microsystems*, vol. 39, no. 2, pp. 97–112, 2015.

[144] T. P. Innokentievich and M. V. Vasilevich, "The evaluation of the cryptographic strength of asymmetric encryption algorithms," in *2017 Second Russia and Pacific Conference on Computer Technology and Applications (RPC)*. IEEE, 2017, pp. 180–183.

[145] S. Kuswaha, S. Waghmare, and P. Choudhary, "Data transmission using aes-rsa based hybrid security algorithms," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 3, no. 4, pp. 1964–1969, 2015.

[146] A. Bar-On, O. Dunkelman, N. Keller, E. Ronen, and A. Shamir, "Improved key recovery attacks on reduced-round aes with practical data and memory complexities," *Journal of Cryptology*, vol. 33, no. 3, pp. 1003–1043, 2020.

[147] S. Sharma and V. Chopra, "Analysis of aes encryption with ecc," in *Proceedings of the 17th International Interdisciplinary Conference on Engineering Science & Management, Dubai, UAE*, 2016, pp. 1–2.

[148] J.-P. Flori, J. Plut, J.-R. Reinhard, and M. Ekerå, "Diversity and transparency for ecc," *Cryptology ePrint Archive*, 2015.

[149] T. M. Kumar, K. S. Reddy, S. Rinaldi, B. D. Parameshachari, and K. Arunachalam, "A low area high speed fpga implementation of aes architecture for cryptography application," *Electronics*, vol. 10, no. 16, p. 2023, 2021.

[150] T. Sasao and J. T. Butler, "The eigenfunction of the reed-muller transformation," 2007.

[151] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," *Cryptology ePrint Archive*, 2020.

[152] I. Skliarova and V. Sklyarov, *FPGA-BASED hardware accelerators*. Springer, 2019.

[153] R. Sornalatha, N. Janakiraman, K. Balamurugan, A. Kumar Sivaraman, R. Vincent, and A. Muralidhar, "Fpga implementation of protected compact aes s–box using cqcg for embedded applications," in *Smart Intelligent Computing and Communication Technology*. IOS Press, 2021, pp. 396–401.

[154] P. Sasdrich, B. Bilgin, M. Hutter, and M. E. Marson, "Low-latency hardware masking with application to aes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 300–326, 2020.

[155] T. De Cnudde, M. Ender, and A. Moradi, "Hardware masking, revisited," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 123–148, 2018.

[156] M. Santhanalakshmi, M. Lakshana, and M. S. GM, "Enhanced aes-256 cipher round algorithm for iot applications," *The Scientific Temper*, vol. 14, no. 01, pp. 184–190, 2023.

[157] N. A. N. Adnan and S. Ariffin, "Big data security in the web-based cloud storage system using 3d-aes block cipher cryptography algorithm," in *Soft Computing in Data Science: 4th International Conference, SCDS 2018, Bangkok, Thailand, August 15-16, 2018, Proceedings 4*. Springer, 2019, pp. 309–321.

[158] P. A. Abdalla and A. Varol, "Advantages to disadvantages of cloud computing for small-sized business," in *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2019, pp. 1–6.

[159] M. A. Balasubramaniam, "High speed aes cipher engine," *International Journal of Engineering and Applied Sciences*, vol. 2, no. 7, p. 257876.

[160] Y. Salami, V. Khajevand, and E. Zeinali, "Cryptographic algorithms: a review of the literature, weaknesses and open challenges," *J. Comput. Robot*, vol. 16, no. 2, pp. 46–56, 2023.

[161] J. H. Seo, "Efficient digital signatures from rsa without random oracles," *Information Sciences*, vol. 512, pp. 471–480, 2020.

[162] M. Mumtaz, J. Akram, and L. Ping, "An rsa based authentication system for smart iot environment," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 758–765.

[163] B. B. Sundaram, N. Kannaiya Raja, N. Sreenivas, M. K. Mishra, B. Pattanaik, and P. Karthika, "Rsa algorithm using performance analysis of steganography techniques in network security," in *International Conference on Communication, Computing and Electronics Systems: Proceedings of ICCCES 2020*. Springer, 2021, pp. 713–719.

[164] Y.-X. Toh, S.-C. Chong, and L.-Y. Chong, "A secured self-financial management application with rsa encryption technology," in *AIP Conference Proceedings*, vol. 3153, no. 1. AIP Publishing, 2024.

[165] P. Aiswarya, A. Raj, D. John, L. Martin, and G. Sreenu, "Binary rsa encryption algorithm," in *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. IEEE, 2016, pp. 178–181.

[166] F. Li, Z. Liu, T. Li, H. Ju, H. Wang, and H. Zhou, "Privacy-aware pki model with strong forward security," *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 10 049–10 065, 2022.

[167] E. A. Abbas, T. A. Karam, and A. K. Abbas, "Image cipher system based on rsa and chaotic maps," *Eurasian Journal of Mathematical and Computer Applications*, vol. 7, no. 4, pp. 4–17, 2019.

[168] A. Pourjabbar Kari, A. Habibizad Navin, A. M. Bidgoli, and M. Mirnia, "A new image encryption scheme based on hybrid chaotic maps," *Multimedia Tools and applications*, vol. 80, pp. 2753–2772, 2021.

[169] H. R. Hashim, "A new modification of rsa cryptosystem based on the number of the private keys," *American Scientific Research Journal for Engineering, Technology, and Sciences*, no. 24, pp. 270–279, 2016.

[170] A. K. Hussain *et al.*, "A modified rsa algorithm for security enhancement and redundant messages elimination using k-nearest neighbor algorithm," *IJISET-International Journal of Innovative Science, Engineering & Technology*, vol. 2, no. 1, pp. 858–862, 2015.

[171] S. Ganapathy *et al.*, "A secured storage and privacy-preserving model using crt for providing security on cloud and iot-based applications," *Computer Networks*, vol. 151, pp. 181–190, 2019.

[172] I. Al_Barazanchi, S. A. Shawkat, M. H. Hameed, and K. S. L. Al-Badri, "Modified rsa-based algorithm: A double secure approach," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 17, no. 6, pp. 2818–2825, 2019.

[173] P. Russek, P. Karbownik, and K. Wiatr, "Weak rsa key discovery on gpgpu," *International Journal of Electronics and Telecommunications*, vol. 65, 2019.

[174] T. H. Obaida, A. S. Jamil, and N. F. Hassan, "A review: Video encryption techniques, advantages and disadvantages," *Webology (ISSN: 1735-188X)*, vol. 19, no. 1, 2022.

[175] A. Hafsa, N. Alimi, A. Sghaier, M. Zeghid, and M. Machhout, "A hardware-software co-designed aes-ecc cryptosystem," in *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. IEEE, 2017, pp. 50–54.

[176] K. Sarwar, S. Yongchareon, and J. Yu, "Lightweight ecc with fragile zero-watermarking for internet of things security," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 867–872.

[177] V. Gopinath and R. Bhuvaneswaran, "Design of ecc based secured cloud storage mechanism for transaction rich applications." *Computers, Materials & Continua*, vol. 57, no. 2, 2018.

[178] S. Gueron and V. Krasnov, "Fast prime field elliptic-curve cryptography with 256-bit primes," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 141–151, 2015.

[179] B. Hammi, A. Fayad, R. Khatoun, S. Zeadally, and Y. Begriche, "A lightweight ecc-based authentication scheme for internet of things (iot)," *IEEE Systems Journal*, vol. 14, no. 3, pp. 3440–3450, 2020.

[180] C. Rajvir, S. Satapathy, S. Rajkumar, and L. Ramanathan, "Image encryption using modified elliptic curve cryptography and hill cipher," in *Smart Intelligent Computing and Applications: Proceedings of the Third International Conference on Smart Computing and Informatics, Volume 1*. Springer, 2020, pp. 675–683.

[181] R. K. Hasoun, S. F. Khlebus, and H. K. Tayyeh, “A new approach of classical hill cipher in public key cryptography,” *International Journal of Nonlinear Analysis and Applications*, vol. 12, no. 2, pp. 1071–1082, 2021.

[182] S. Di Matteo, L. Baldanzi, L. Crocetti, P. Nannipieri, L. Fanucci, and S. Saponara, “Secure elliptic curve crypto-processor for real-time iot applications,” *Energies*, vol. 14, no. 15, p. 4676, 2021.

[183] A. Hafsa, A. Sghaier, J. Malek, and M. Machhout, “Image encryption method based on improved ecc and modified aes algorithm,” *Multimedia Tools and Applications*, vol. 80, pp. 19 769–19 801, 2021.

[184] J. A. Stankovic, “Real-time and embedded systems,” *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 205–208, 1996.

[185] A. Shantha, J. Renita, and E. N. Edna, “Analysis and implementation of ecc algorithm in lightweight device,” in *2019 International conference on communication and signal processing (ICCSP)*. IEEE, 2019, pp. 0305–0309.

[186] D. B. Roy and D. Mukhopadhyay, “Post quantum ecc on fpga platform,” *Cryptology ePrint Archive*, 2019.

[187] R. Bavdekar, E. J. Chopde, A. Bhatia, K. Tiwari, S. J. Daniel *et al.*, “Post quantum cryptography: Techniques, challenges, standardization, and directions for future research,” *arXiv preprint arXiv:2202.02826*, 2022.

[188] R. Abarzúa, C. Valencia, and J. López, “Survey for performance & security problems of passive side-channel attacks countermeasures in ecc,” *Cryptology ePrint Archive*, 2019.

[189] A. Srivastava and A. Kumar, “A review on authentication protocol and ecc in iot,” in *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. IEEE, 2021, pp. 312–319.

[190] M. Panda, “Performance analysis of encryption algorithms for security,” in *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. IEEE, 2016, pp. 278–284.

[191] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, “Comprehensive study of symmetric key and asymmetric key encryption algorithms,” in *2017 international conference on engineering and technology (ICET)*. IEEE, 2017, pp. 1–7.

[192] M. T. Rouaf and A. Yousif, “Performance evaluation of encryption algorithms in mobile devices,” in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*. IEEE, 2021, pp. 1–5.

[193] K. Sarwar, S. Yongchareon, and J. Yu, “A brief survey on iot privacy: Taxonomy, issues and future trends,” in *Service-Oriented Computing–ICSOC 2018 Workshops: ADMS, ASOCA, ISYyCC, CloTS, DDBS, and NLS4IoT, Hangzhou, China, November 12–15, 2018, Revised Selected Papers 16*. Springer, 2019, pp. 208–219.

[194] A. H. Khan, M. A. Al-Mouhamed, A. Almousa, A. Fatayar, A. Ibrahim, and A. Siddiqui, “Aes-128 ecb encryption on gpus and effects of input plaintext patterns on performance,” in *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2014, pp. 1–6.

[195] G. F. Naiem, S. Elramly, B. E. Hasan, and K. Shehata, “An efficient implementation of cbc mode rijndeal aes on an fpga,” in *2008 National Radio Science Conference*. IEEE, 2008, pp. 1–8.

[196] V. K. Myalapalli and S. Geloth, “High performance java programming,” in *2015 International Conference on Pervasive Computing (ICPC)*. IEEE, 2015, pp. 1–6.

[197] T. Suezawa, “Persistent execution state of a java virtual machine,” in *Proceedings of the ACM 2000 conference on Java Grande*, 2000, pp. 160–167.

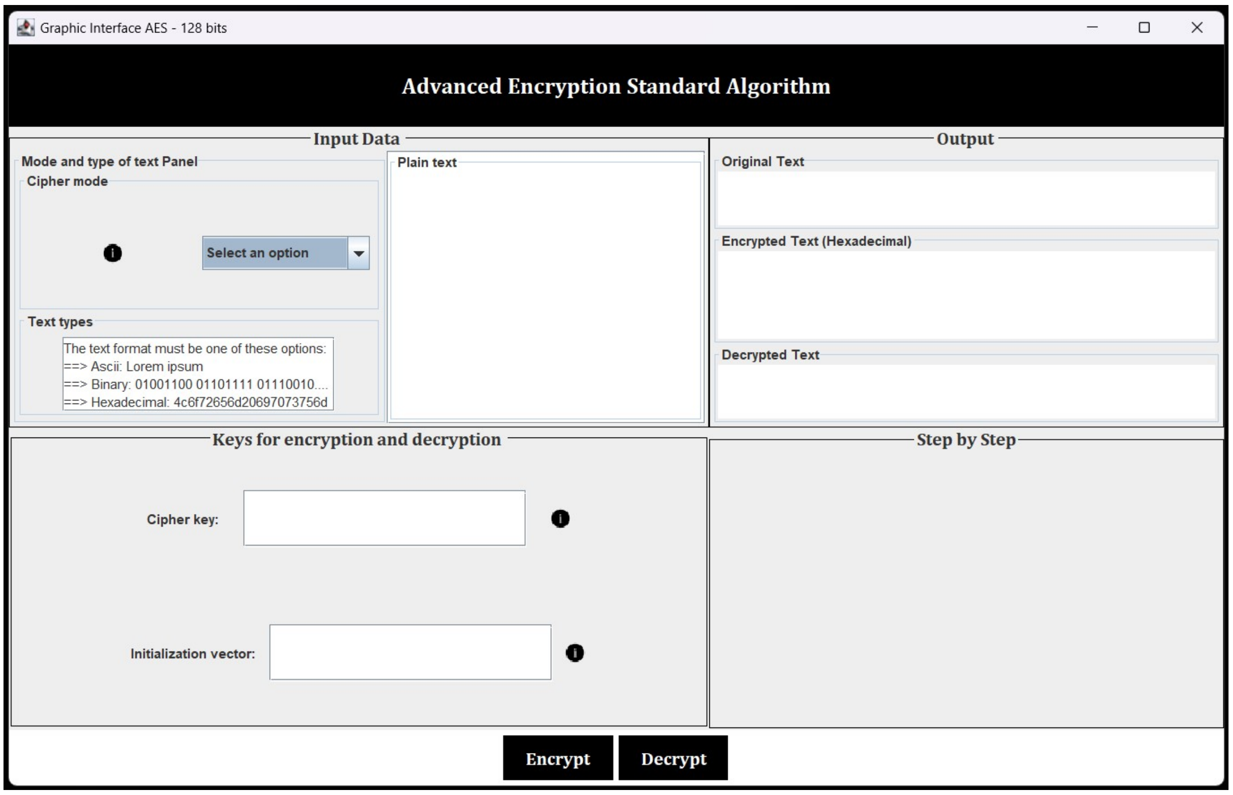
[198] R. Küsters, T. Truderung, B. Beckert, D. Bruns, M. Kirsten, and M. Mohr, “A hybrid approach for proving noninterference of java programs,” in *2015 IEEE 28th Computer Security Foundations Symposium*. IEEE, 2015, pp. 305–319.

[199] M. A. Shaaban, A. S. Alsharkawy, M. T. Abou-Kreisha, and M. A. Razek, “Efficient ecc-based authentication scheme for fog-based iot environment,” *International Journal of Computer Networks & Communications (IJCNC)*, vol. 15, no. 4, 2023.

[200] Z. Liu, H. Seo, A. Castiglione, K.-K. R. Choo, and H. Kim, “Memory-efficient implementation of elliptic curve cryptography for the internet-of-things,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 521–529, 2018.

Appendix A

AES Graphic Interface


Graphic Interface AES - 128 bits
Advanced Encryption Standard Algorithm
Input Data
Mode and type of text Panel
Cipher mode
Select an option
Text types
Plain text
Output
Original Text
Encrypted Text (Hexadecimal)
Decrypted Text
Keys for encryption and decryption
Cipher key:
Initialization vector:
Step by Step
Encrypt
Decrypt

The text format must be one of these options:
==> Ascii: Lorem ipsum
==> Binary: 01001100 01101111 01110010....
==> Hexadecimal: 4c6f72656d20697073756d

Figure A.1: Graphic Interface for AES algorithm

Appendix B

RSA Graphic Interface

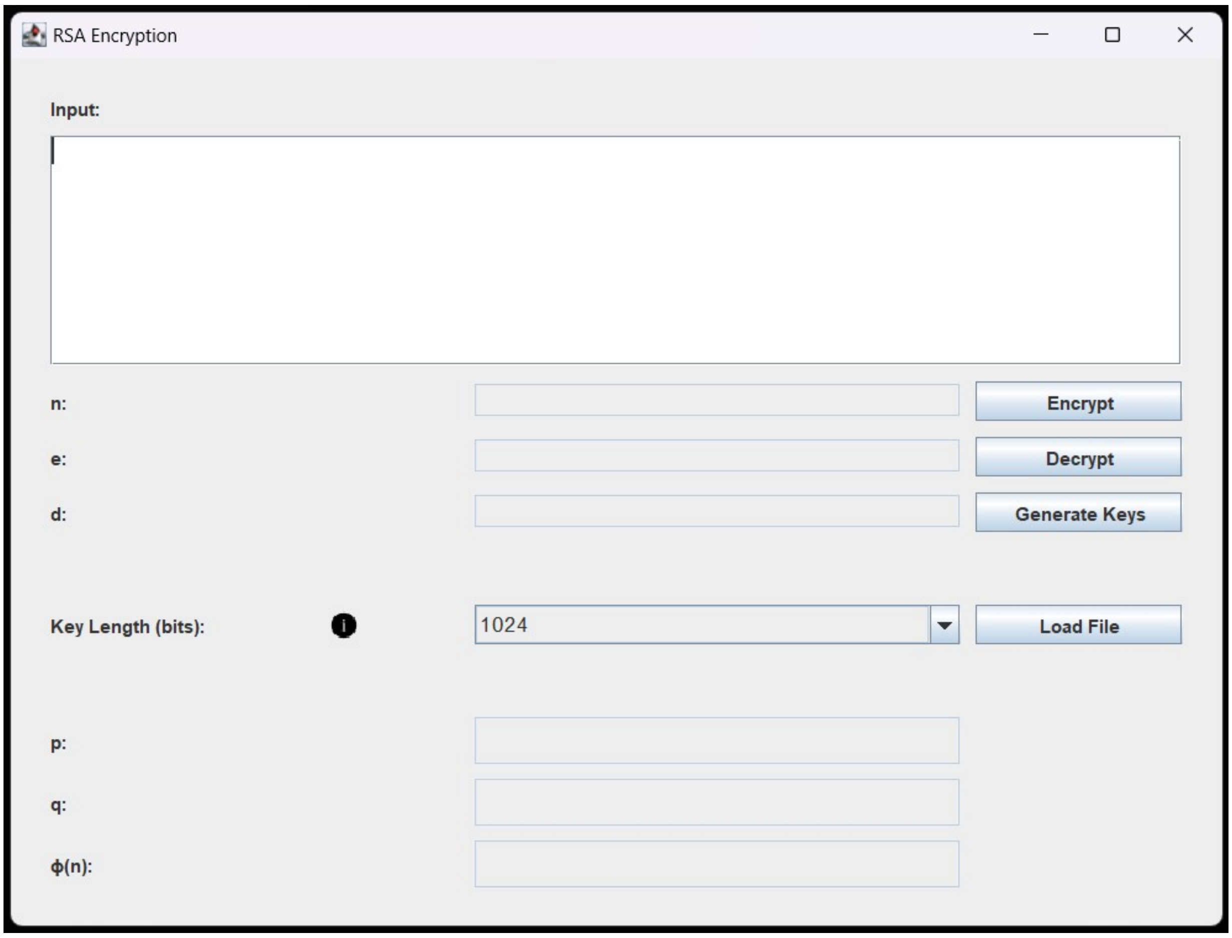


Figure B.1: Graphic Interface for RSA algorithm

Appendix C

ECC Graphic Interface

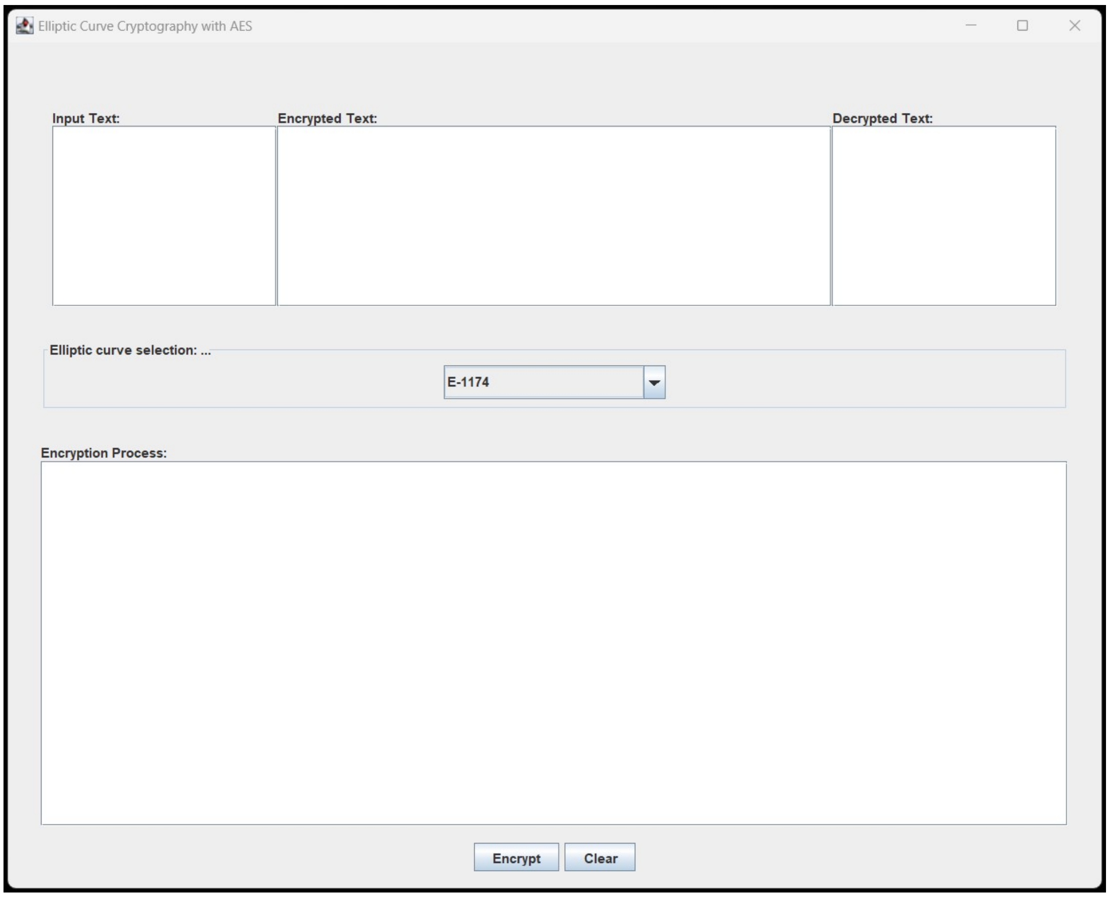


Figure C.1: Graphic Interface for ECC algorithm