



ደብረ ብርሃን ዩኒቨርሲቲ
Debre Berhan University



“From the community to the community”

**COLLEGE OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY
MASTERS OF COMPUTER NETWORK AND SECURITY**

*Improving the Execution Speed of Rivest-Shamir-Adleman
Cryptosystem Using Modified Modular Exponentiation Algorithm*

Debebe Kebede Mamo

debebekebede1426@gmail.com

Debre Berhan, Ethiopia

February 2023



ደብረ ብርሃን ዩኒቨርሲቲ
Debre Berhan University



“From the community to the community”

COLLEGE OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY
MASTERS OF COMPUTER NETWORK AND SECURITY

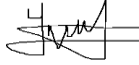

*Improving the Execution Speed of Rivest-Shamir-Adleman
Cryptosystem Using Modified Modular Exponentiation Algorithm*

Debebe Kebede Mamo

Advisor: Yelkal Mulualem (PhD)

This is to certify that the thesis prepared by *Debebe Kebede*, titled: *Improving the Execution Speed of Rivest-Shamir-Adleman Cryptosystem Using Modified Modular Exponentiation Algorithm* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Network and Security complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

<u>Name</u>	<u>Signature</u>	<u>Date</u>
Advisor: <u>Yelkal Mulualem</u>		<u>23/06/2015 E.C</u>
Examiner: _____	_____	_____
Examiner: <u>Asrat Mulatu (Ph.D.)</u>		<u>20/04/2023</u>
Chairperson: _____	_____	_____

Debre Berhan, Ethiopia

February 2023

Dedication

This thesis is dedicated to my mother, Nigatwa Tezera, and my father, Kebede Mamo. Without their endless love and encouragement, I would never have been able to complete my graduate studies. I love you both and I appreciate everything that you have done for me.

Abstract

Nowadays, public-key cryptography, especially Rivest-Shamir-Adleman (RSA), is becoming more and more crucial for e-commerce transactions. Several digital consumer appliances, such as set-top boxes and smart cards, may now send and receive many RSA-encoded messages. However, due to the exponentiation and multiplications of very large numbers, the RSA algorithm is inappropriate for encrypting large messages. Hence to fill this gap, this thesis work proposes an efficient modular exponentiation technique for the RSA cryptosystem which can enhance the execution speed of the standard RSA cryptosystem by applying the square-multiply technique of exponentiation. The MATLABR2019a programming platform is employed for the actual implementation of this study. The proposed work discusses the Standard RSA (SRSA) and Modified RSA (MRSA) cryptographic algorithms by considering different performance evaluation metrics such as encryption time, decryption time, memory usage, throughput, and avalanche effect through a practical implementation using MATLAB. Concerning the encryption and decryption time performance, the MRSA algorithm outperforms the SRSA algorithm by 17.46442% and 18.70409%, respectively. Moreover, the MRSA has a 14.53511% higher avalanche value than the SRSA, showing its greater security. Also, it outperforms the SRSA algorithm in terms of encryption and decryption throughput by 15.80883% and 15.12928% respectively. However, because the MRSA algorithm uses an array list, which uses more memory than the SRSA, it consumes slightly more memory (1.05849%) overall.

Keywords: *Modular Exponentiation, Rivest-Shamir-Adleman (RSA), Encryption and Decryption, Cryptography, Private Key, Public Key, Security*

Acknowledgment

I will always be grateful to God, who is the Creator and the Guardian and to whom I owe my existence.

I would like to express my gratitude and admiration for my adviser, Dr. Yelkal Mulualeman, an Assistant Professor at the University of Gondar. He has been a fantastic mentor for me. His enlightening direction, kind critique, and friendly counsel have been priceless. I genuinely appreciate him for providing his insightful and accurate opinions on a variety of topics.

I also want to express my gratitude to Dr.Eng. Yihenew Wondie, Dr. Esubalew Yitayal, Dr. Samuel Asferaw, Dr. Henock Mulugeta, Dr. Workshet Lamenu, and Mr. Getaneh Awlache for sharing their inspiring ideas with me. Without their assistance, I might not have had the chance to complete my thesis.

My family also deserves special thanks. Words cannot express how grateful I am to them for all the sacrifices they've made on my behalf. Their trust and vote of confidence have kept me going thus far.

Finally, it gives me great pleasure to acknowledge the contributions of my colleagues, whose friendship, cooperation, and understanding were essential to the success of my thesis.

Table of Contents

Abstract	ii
Acknowledgment	iii
List of Figures	iv
List of Tables	v
Acronyms and Abbreviations	vi
Chapter One	1
Introduction.....	1
1.1. Background	1
1.1. Motivation	2
1.2. Statement of the Problem	3
1.3. Research Questions	4
1.4. Research Objectives	4
1.5. Significance of the Study	4
1.6. Scope and Delimitation of the Study.....	5
1.7. Thesis Outline	5
Chapter Two.....	6
Literature Review.....	6
2.1. Introduction	6
2.1. The RSA Algorithm	10
2.2. Working Principles of RSA Algorithm.....	11
2.3. RSA Algorithm with Example	13
2.4. Application of Prime Numbers in Cryptography	16
2.5. Restrictions on the Selection of Prime Numbers	16

2.6.	RSA Digital Signature Scheme	17
2.6.1.	The RSA Digital Signature Algorithms:.....	18
2.7.	Number Theory Behind RSA.....	19
2.7.1.	Prime Generation and Integer Factorization.....	19
2.8.	Practical applications of the RSA Algorithm.....	19
2.9.	Advantages and Disadvantages of the RSA Algorithm	20
2.10.	Execution Speed of RSA Algorithm	20
2.11.	Related Works	21
2.12.	Summary of Related Works	25
Chapter Three.....		27
3.1.	Introduction	27
3.2.	Computational Aspects of the RSA Algorithm.....	27
3.3.	Modular Exponentiation.....	28
3.4.	The Naive Modular Exponentiation Method	29
3.4.1.	Memory Efficient Method	29
3.6.	The Modified Modular Exponentiation Method	31
4.1.	Overview	35
4.2.	Implementation.....	35
4.2.1.	Standard RSA (SRSA) Implementation	36
4.2.2.	Modified RSA(MRSA) Implementation	38
4.3.	Performance Analysis and Results.....	40
4.3.1.	Encryption Time	40
4.3.2.	Decryption Time.....	41
4.3.3.	Memory Utilization	43
4.3.4.	Avalanche Effect	45

4.3.5. Throughput	50
4.4. Discussions.....	53
Chapter Five.....	54
Conclusion and Recommendations.....	54
5.1. Conclusion.....	54
5.2. Recommendations	54
5.3. Contributions.....	55
5.4. Future Work	55
References.....	56

List of Figures

Figure 1.1: A Simple Illustration of Public Key Cryptography	9
Figure 2.2: Public Key Cryptography	10
Figure 2.3: Structure of RSA Algorithm.....	13
Figure 2.4: Flowchart of RSA encryption and decryption operations	15
Figure 3.2: Naive Modular Exponentiation Example	30
Figure 3.3: Flowchart for Computing $ab \bmod n$ Using Modified Method	33
Figure 4.0: Generating a random prime number having a 256-bit size	36
Figure 4.1: Implementation of SRSA Algorithm for p and q values	37
Figure 4.2: ASCII code and Decrypted Message of SRSA	37
Figure 4.3: Implementation of MRSA Algorithm for p and q values.....	38
Figure 4.4: ASCII code and Decrypted Message of MRSA	39
Figure 4.5: Analysis of Encryption Performance (in seconds)	41
Figure 4.6: Analysis of Decryption Performance (in seconds).....	43
Figure 4.7: Snapshot showing Memory consumed by the MRSA.....	43
Figure 4.8: Snapshot showing Memory consumed by the SRSA.....	44
Figure 4.9: Analysis of Memory Usage (MB)	45
Figure 4.10: Analysis of Avalanche Effect (%).....	49
Figure 4.11: Analysis of Encryption Throughput (BPs).....	51
Figure 4.12: Analysis of Decryption Throughput (BPs).....	53

List of Tables

Table 2.0: Summary of Related Works.....	25
Table 2Table 3.1: Result of the Modified Modular Exponentiation Algorithm for $ab \pmod n$, where $a = 240$, $b = 262(10) = 100000110(2)$, $n = 14$	34
Table 4.0: Summary of Encryption Time for the Standard and Modified RSA Algorithm	40
Table 4.1: Summary of Decryption Time for the Standard and Modified RSA Algorithm	41
Table 4.2: Comparison of the encryption time of SRA and MRSA algorithms (%)	42
Table 4.3: Comparison of the decryption time of SRA and MRSA algorithms (%)	42
Table 4.4: Summary of Memory Usage for the Standard and Modified RSA Algorithm	44
Table 4.5: Comparison of the memory consumption of SRA and MRSA algorithms (%)	45
Table 4.6: Avalanche Test of the SRSA	47
Table 4.7: Avalanche Test of the MRSA.....	Error! Bookmark not defined.
Table 4.8: Comparison of the avalanche effect of SRA and MRSA algorithms (%)	49
Table 4.9: Summary of Encryption Time for the Standard and Modified RSA Algorithm Using Different Message Sizes	50
Table 4.10: Summary of Encryption Throughput for the Standard and Modified RSA Algorithm (Byte/second)	51
Table 4.11: Summary of Decryption Time for the Standard and Modified RSA Algorithm Using Different Message Sizes	51
Table 4.12: Summary of Decryption Throughput for the Standard and Modified RSA Algorithm (Byte/second)	52
Table 4.13: Comparison of the encryption throughput of SRA and MRSA algorithms (%).....	52
Table 4.14: Comparison of the decryption throughput of SRA and MRSA algorithms (%).....	52

Acronyms and Abbreviations

BFW	Bit Forwarding
CRT	Chinese Remainder Theorem
LPC	Linear Predictive Code
MATLAB	Matrix Laboratory
MRSA	Modified RSA
MSB	Most Significant Bit
PGP	Pretty Good Privacy
SET	Secure Electronic Transaction
SEW	Substitute and Reward
SFW	Store and Forward
SMTP	Simple Mail Transfer Protocol
SRSA	Standard RSA
SSL	Secure Socket Layer
SSNR	Segmental Signal-to-Noise

Chapter One

Introduction

1.1. Background

The RSA algorithm was introduced in 1977 by Rivest, Shamir, and Adleman at MIT and was first published in 1978. The RSA scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key cryptosystems [1]. RSA has been used extensively in various applications such as banking, telecommunications, and eCommerce. It is often used to secure communications between web browsers and eCommerce sites.

Four key concepts serve as the foundation for cryptography, with the goals of preserving confidentiality, data integrity, authenticity, and non-repudiation. By establishing a set of guidelines that restrict access to particular data, cryptography provides confidentiality. On the other side, data integrity is maintained by making sure that data is accurate and consistent over its entire life cycle. Authentication, on the other hand, helps in verifying the veracity of a datum attribute that some entity asserts to be true. Cryptography also assures that a statement or piece of data's originator cannot refute it in the case of nonrepudiation[2].

With the help of cryptographic technology, it is possible to communicate or transmit data electronically without having to worry about fraud or other forms of deception (confidentiality), while also retaining the message's integrity and the sender's legitimacy. This is accomplished by converting data into various, unintelligible formats (ciphertext or code). Encryption refers to the act of converting data into ciphertext, and decryption refers to the process of converting ciphertext into plaintext, or understandable data. Both encryption and decryption processes are carried out using secret information such as passwords or keys [3].

RSA is the typical public-key cryptosystem for securing the information sent through the insecure channel by using encryption and decryption processes. This algorithm uses a pair of keys for the implementation. One is called the public key, which is known to everyone. The other is called a private key, which is kept secretly by the owner. RSA is of one-way function that is very easy to compute two prime numbers. On the other hand, the inverse which is about integer factorization is

very difficult. If the key length is too small, RSA may be broken by using some attacking techniques such as factoring algorithms. Therefore, the key length should be at least 1024 bits to avoid attacking from third parties. Because of technology which is grown very fast, 1024 bits may not be safe enough. At present, this algorithm is presented to perform with a high key length which is larger than 1024 bits to apply with many applications to ensure a high-security level [4].

The RSA algorithm is based on the simple mathematical principle that —It is relatively easy to multiply two large prime numbers, but it is very difficult to factorize the product into prime factors if the product is supplied [5]. The expectation that the encryption work is the security of RSA that depends on, thus decoding an encrypted message is computationally infeasible for an intruder. Public-key cryptosystems are the most popular, due to both confidentiality and authentication facilities [6]. The message is encrypted with the public key and can only be decrypted by using the private key. So, the encrypted message cannot be decrypted by anyone who knows only the public key, and thus secure communication is possible. In a public-key cryptosystem, the private key is always linked mathematically to the public key. Therefore, it is always possible to attack a public-key system by deriving the private key from the public key. The defense against this is to make the problem of deriving the private key from the public key as difficult as possible. Some public-key cryptosystems are designed such that deriving the private key from the public key requires the attacker to factor in a large number.

The security of RSA relies on the practical difficulty of factoring in the product of two large prime numbers. There is no published method to defeat the system if a large enough key is used. However, if too big a key size is used, the execution speed of the algorithm becomes inefficient [7].

In this paper, we aim to apply a modified modular exponentiation algorithm to reduce the execution time for the encryption and decryption processes of the RSA algorithm.

1.1. Motivation

Public key cryptography and RSA in particular, is increasingly important to e-commerce transactions. Many digital consumer appliances (e.g. set-top boxes and smart cards) are now expected to send and receive RSA-encoded messages. However, the competition requires a long processing time. This is an issue even for gigahertz desktop computers. For embedded systems, which typically employ much slower processors, it is an overwhelming problem. Consumers simply will not tolerate digital set-top boxes that pause for long periods while exchanging secret information with Web sites.

Users will also look dimly upon smart-access devices (smart cards and the associated readers) that force them to wait in the rain for just a little too long while checking their access privileges. Consequently, RSA encryption and decryption speed is becoming important to a variety of embedded devices [8].

Today, the modulus n is typically chosen to be 1024, 2048, 3072, 7680, and 15360-bit numbers [9]. For numbers this large it would take hundreds of thousands of high-speed computers many years to derive p and q from the public keys. For these reasons, the researcher believes that conducting a study associated with the RSA execution speed performance enhancement is one of the hot issues to be studied.

1.2. Statement of the Problem

Some of the drawbacks associated with RSA include the fact that the RSA cryptosystem is relatively slow [10] and thus inappropriate for encrypting large messages; this is due to the exponentiation and multiplications of very large numbers. Whether for encrypting, decrypting, signing, or verifying, the RSA cryptosystem uses a series of modular multiplications. In RSA, both encryption and decryption involve raising an integer to an integer power, **mod** n . Calculating a modular exponent is as simple as calculating b^e directly and then taking that number **modulo** n . Consider trying to compute c , given $b = 5$, $e = 11$, and $n = 326$: $c \equiv 5^{11} \pmod{326}$. A calculator can be used to compute 5^{11} ; this comes out to **48,828,125**. Taking this value **modulo** **326**, the answer c is determined to be **171**. Here b is only one digit in length and that e is only two digits in length, but the value b^e is 8 digits in length. In strong cryptography, b is often at least 1024 bits. Consider the values $b = 7 \cdot 13^{10}$ and $e = 14$, which are both perfectly reasonable. In this example, b is 12 digits in length and e is 2 digits in length, but the value b^e is 169 decimal digits in length.

Modern computers can perform such calculations; however, the sheer volume of the numbers slows the calculations down significantly. As b and e increase even further to provide better security, the value becomes unwieldy. The amount of time it takes to conduct exponentiation is determined by the operating environment and processor. The majority of modular arithmetic's technological applications involve exponentials with huge numbers. Our proposed method is assumed to reduce the number of operations required to conduct modular exponentiation

drastically while maintaining the same memory footprint as the previous way. It combines the previous method with a more general concept known as binary exponentiation.

1.3. Research Questions

In addressing the above-mentioned gaps, this paper aims to investigate and fill these gaps by answering the following two research questions. 1) How to improve the execution time of the standard RSA cryptosystem? 2) How to implement the proposed algorithm using appropriate cryptography tools and techniques?

1.4. Research Objectives

General Objective

The study's general objective is to improve the execution speed of the RSA using a modified modular exponentiation algorithm.

Specific Objectives

The specific objectives include:

- To demonstrate the performance gap of the RSA algorithm.
- To design an enhanced scheme that conquers the standard RSA performance deficiency.
- To implement the proposed algorithm using appropriate cryptography tools.
- To compare the performance of the suggested RSA computational approach to that of the standard scheme.

1.5. Significance of the Study

Nowadays E-commerce transactions are becoming more and more dependent on public-key cryptography RSA in particular. Many RSA-encoded messages are now able to be sent and received by a large number of digital consumer appliances, including set-top boxes and smart cards [11]. It requires a significant amount of processing time to decode this information, so investigating this research is expected to solve such challenges. The study also has a significant contribution to both academic researchers and practitioners. Academic researchers will be benefited from the theoretical and empirical contribution since it tries to fill the existing literature gap notably on the RSA cryptosystem efficiency defect. More than ever, studies are needed in quantum-crypto-resistant methods for

quantum computers, as existing cryptographic systems will soon be resolved. This study will support the development of crypto processors by assisting in the effective selection of the underlying modular exponentiation unit to reduce hardware cost complexity while increasing the execution speed of the RSA algorithm.

1.6. Scope and Delimitation of the Study

This study is focusing on the improvement of the RSA cryptosystem execution speed during data encryption and decryption while preserving its security. Due to a limited amount of financial resources and time framework, the study is delimited to only the enhancement of standard RSA cryptosystem implementation efficiency. The study is also limited to the enhancement of the RSA algorithm encryption and time rather than its key generation time. Due to limited computer system specifications, the practical implementation is performed by utilizing smaller key sizes ranging from 28 bits to 512 bits. The proposed work also addressed only textual data for practical implementation.

1.7. Thesis Outline

The rest of the document is organized as follows. In Chapter Two review of the literature about our research is presented. Key concepts about the RSA cryptosystem and modular exponentiation algorithm are described. Different studies that are presented in investigating the RSA algorithm and other cryptosystems are also reviewed and compared to one another.

In Chapter Three, we present the methodology of the proposed system. It discusses the design considerations, algorithm design, and major components.

Chapter Four presents an implementation and discussion of the experimental result and its evaluation.

Finally, conclusions, contributions, and future work are presented in Chapter Five.

Chapter Two

Literature Review

2.1. Introduction

The greatest and possibly the only genuine revolution in the history of cryptography is the development of public-key cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation. After millennia of working with algorithms that could be calculated by hand, a major advance in symmetric cryptography occurred with the development of the rotor encryption [12].

Public-key cryptography is a fundamental change from everything that has gone before. For starters, rather than relying on substitution and permutation, public-key algorithms are based on mathematical functions [13]. Asymmetric public-key cryptography, which employs two different keys, is more significant than symmetric encryption, which only requires one key. The usage of two keys has significant effects on key distribution, authentication, and confidentiality.

An attempt to tackle two of the most challenging issues with symmetric encryption—the key distribution problem and the problem of trust between two parties resulted in the development of the idea of public-key cryptography.

For symmetric encryption to work, the two parties to exchange must share the same key and that key must be protected from access by others. Moreover, frequent key changes are typically preferred to reduce the amount of data that could be compromised if an attacker discovers the key. As a result, the key distribution technique—a word that describes the means of delivering a key to two parties that wish to exchange data without exposing the secret to others—is what gives any cryptographic system its strength. Key distribution can be accomplished for two parties A and B in a variety of ways, including the following:

1. A can select a key and hand it to B in person.
2. The key may be chosen and physically delivered to A and B by a third party.
3. If A and B have recently and previously used a key, one of them may send the

other a new key that has been encrypted with the old key.

4. If **A** and **B** each have an encrypted connection to a third-party **C**, **C** may deliver a key on the encrypted channel to **A** and **B**.

For options 1 and 2, a key must be manually delivered. This is a realistic requirement for link encryption as each link encryption device will only exchange data with its partner at the other end of the link. However, for end-to-end encryption via a network, awkward manual delivery. Any device in a distributed system exchanges between a specific host or terminal and numerous other hosts and terminals as time goes on. Therefore, each device requires a set of keys that are dynamically given. In a wide-area distributed system, the issue is particularly challenging.

The second problem of symmetric cryptography apparently unrelated to the first is that of digital signatures. Electronic messages and documents would require the equivalent of signatures used in paper documents if encryption were to be widely employed, not just in military situations but also for commercial and private purposes. In other words, could a method be developed to conclusively prove that a digital message was transmitted by a certain individual? Compared to authentication, this requirement is a little bit broader.

To tackle the above problems of symmetric cryptography, public key cryptography/asymmetric key cryptography/ is introduced. Asymmetric algorithms utilize a pair of two keys: one for encryption and the other, for decryption. These algorithms have the following significant features:

- Given just the encryption key and the cryptographic algorithm, it is computationally infeasible to figure out the decryption key.
- The two related keys can be used interchangeably for encryption and decryption.

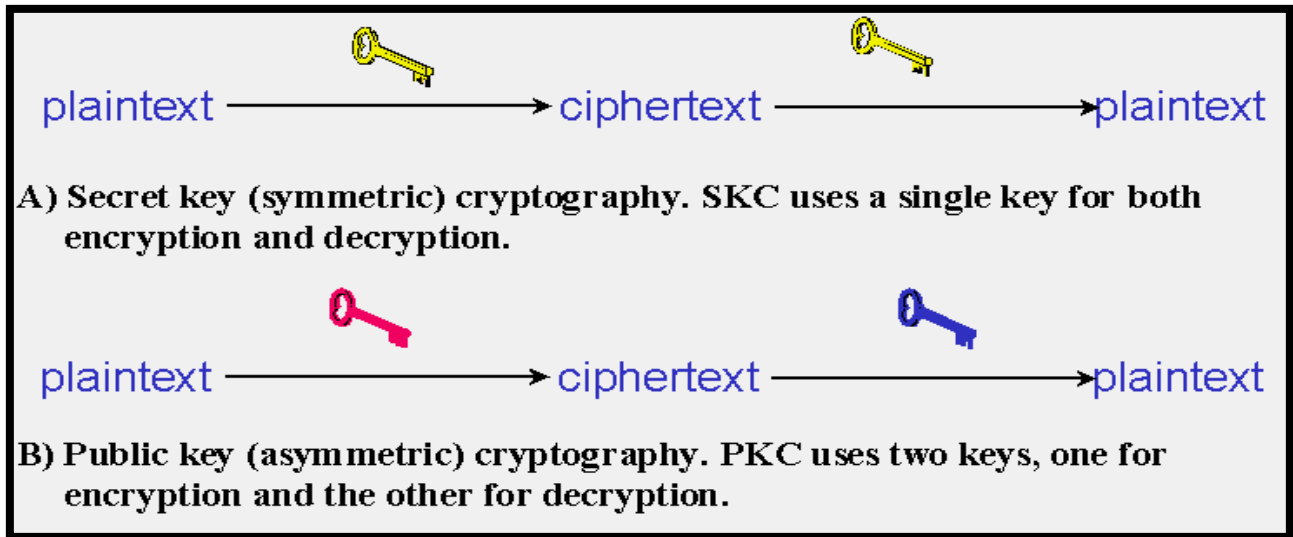


Figure 2.1: Secret Key (Symmetric) and Public Key (Asymmetric) Cryptography [14]

There are six components to a public-key encryption scheme.

Plaintext: The readable message or data that is sent to the algorithm as input.

The encryption algorithm: The method used to transform data into an encrypted message (cipher text).

Public and private keys: This is a pair of keys that have been chosen so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.

Cipher text: This is the message that was produced as a scrambled output. The plaintext and the key will determine this. Two separate keys will result in two distinct ciphertexts for the same message.

Decryption algorithm: This algorithm generates the original plaintext from the ciphertext and the matching key.

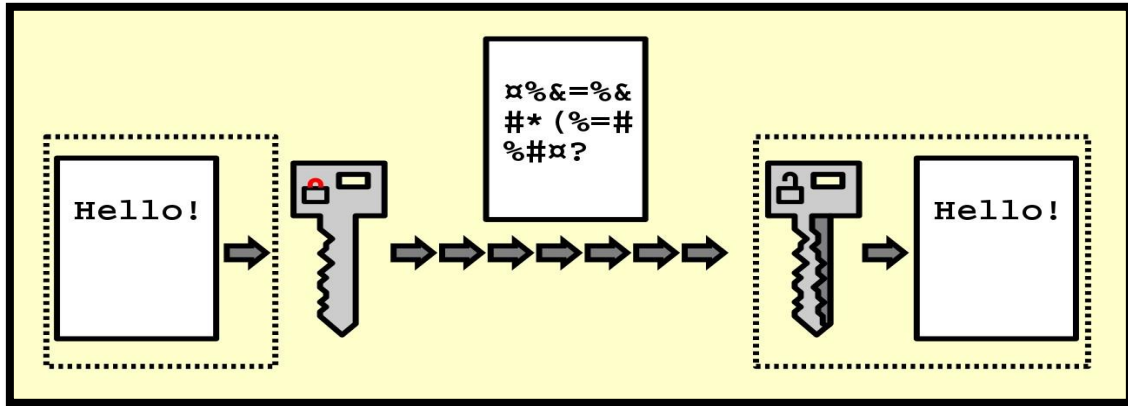


Figure 1.1: A Simple Illustration of Public Key Cryptography [15]

The following are the essential steps of public key cryptography:

1. For the encryption and decryption of messages, each user generates a pair of keys.
2. One of the two keys is stored by each user in a public register or another easily accessible file. This is the public key. The second key remains a secret. Each user keeps a collection of public keys they have acquired from other users, as shown in Figure 2.2.
3. Bob encrypts the message using Alice's public key if he wishes to send her a confidential message.
4. When Alice receives the message, she uses her private key to decrypt it. Because only Alice has access to her private key, no other recipient will be able to decrypt the message.

In this method, private keys are generated locally by each participant and never need to be shared because everyone has access to the public keys. Incoming communication is secure as long as a user's private key is kept private and secure. A system can update its private key at any time and publish the associated public key to replace its previous public key.

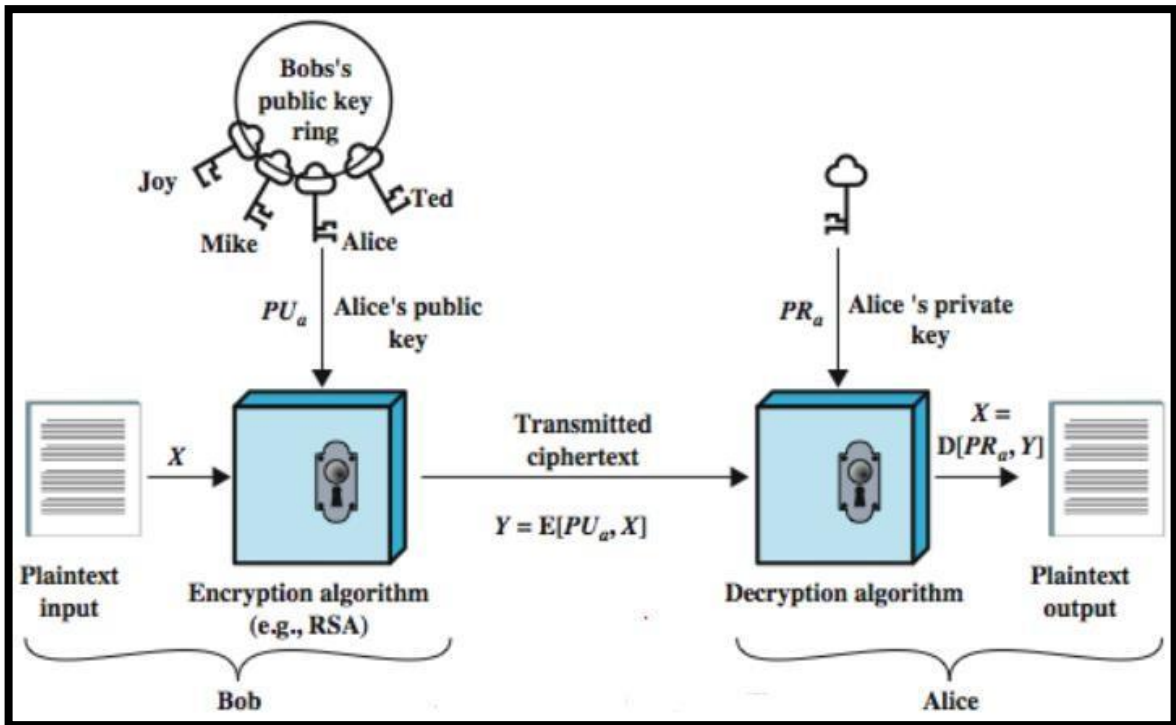


Figure 2.2: Public Key Cryptography [1]

2.1. The RSA Algorithm

The RSA public key cryptosystem was invented by R. Rivest, A. Shamir, and L. Adleman. The RSA cryptosystem is built on the dramatic difference between how simple it is to find large primes and how challenging it is to factor the product of two large prime numbers.

The RSA cryptosystem is built on the dramatic difference between how simple it is to find large primes and how challenging it is to factor the product of two large prime numbers. The algorithm is based on modular exponentiation. Numbers e , d , and N are chosen with the property that if A is a number less than N , then $(Ae \bmod N)d \bmod N = A$. This indicates that we can use e to encrypt A and d to decrypt it. On the other hand, we can decrypt with e and encrypt with d . (though doing it this way round is usually referred to as signing and verification). The pair of numbers (e, N) is known as the public key and can be published. Whereas the pair of numbers (d, N) is known as the private key and must be kept secret. The terms –public exponent,|| –private exponent,|| and –modulus|| are used to describe the numbers e , d , and N , respectively. The modulus length is meant when key lengths are discussed about RSA. The algorithm uses different keys for encryption and

decryption is said to be asymmetric. Anyone with access to the public key can encrypt messages, but only the owner of the secret key can decrypt them. Conversely, the owner of the secret key can encrypt messages that anyone with the public key can decrypt. If such messages are successfully decrypted, only the owner of the secret key could have encrypted them, according to the successful decryption process. The digital signature technique is based on this reality.

2.2. Working Principles of RSA Algorithm

The private key in RSA is kept secret, but the public key is distributed to everyone in the framework. The RSA algorithm has four steps: key generation, key distribution, message encryption, and message decryption.

Key Generation

- i. The receiver arbitrarily chooses two huge prime integers **p** and **q** of similar key lengths.

To determine whether a number is prime, two alternative primality tests can be applied. One is known as the Miller-Rabin Primality Test, while the other is the Fermat Primality Test.

In The American National Standards Institute (ANSI X9.31), there are more strict rules for creating strong primes and additional restrictions on **p** and **q** to reduce the likelihood of known techniques being used against the algorithm. This topic is the subject of numerous arguments. Most likely, it would be best to simply use a longer key length [16].

- ii. The receiver computes the common module **n** such that $n=p*q$
- iii. It computes the Euler's totient function: $\phi(n) = (p-1)(q-1)$.
- iv. Choose a number, **e**, less than **n**, that has no common factors (other than 1) with $\phi(n)$.

In this case, **e** and **n** are said to be relatively prime. The letter **e** is used since this value will be used in encryption. In practice, the most frequent choices for **e** are **3**, **17**, and **65537** ($2^{16}+1$). These are Fermat primes, often known as F0, F2, and F4 respectively. They were chosen because they speed up the process of modular exponentiation.

- v. Find a number, **d**, such that **ed - 1** is exactly divisible (that is, with no remainder) by $\phi(n)$.

The letter **d** is used because this value will be used in decryption. Put another way, given **e**, we choose **d** such that **ed mod n = 1**. This is known as modular inversion.

The Extended Euclidean Algorithm can be used to calculate **d = e⁻¹ mod phi**, or **d = (1/e) mod phi**, to determine the value of **d**.

- vi. The receiver will publish **e** and **n** which are his public keys and keeps **d** as a secret key (private key).

Key Distribution

Suppose that Bob wishes to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message, and Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (**n, e**) to Bob via a reliable, but not necessarily secret, route. Alice's private key (**d**) is never distributed.

Message Encryption

Suppose Alice wants to send Bob a bit pattern represented by the integer number **m** (with **m < n**). To encode, Alice performs the exponentiation **m^e**, and then computes the integer remainder when **m^e** is divided by **n**. In other words, the encrypted value, **c**, of Alice's plaintext message, **m**, is

$$c = m^e \bmod n$$

The bit pattern corresponding to this cipher text **c** is sent to Bob.

Message Decryption

To decrypt the received cipher text message, **c**, Bob computes

$$m = c^d \bmod n$$

which requires the use of his private key (**n, d**).

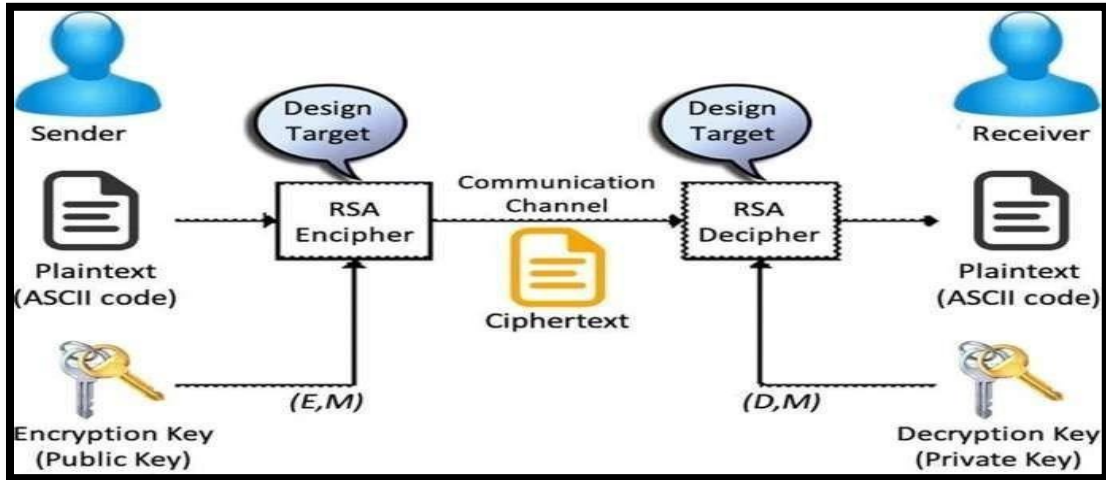


Figure 2.3: Structure of RSA Algorithm [17]

2.3. RSA Algorithm with Example

Step 1: Select primes $p=11$, $q=3$.

Step 2: $n = p \cdot q = 11 \cdot 3 = 33$ $\phi = (p-1)(q-1) = 10 \cdot 2 = 20$

Step 3: Choose $e=3$

Check $\gcd(e, p-1) = \gcd(3, 10) = 1$ (i.e. 3 and 10 have no common factors except 1), and check $\gcd(e, q-1) = \gcd(3, 2) = 1$ therefore $\gcd(e, \phi) = \gcd(e, (p-1)(q-1)) = \gcd(3, 20) = 1$

Step 4: Compute d such that $ed \equiv 1 \pmod{\phi}$

i.e. compute $d = e^{-1} \pmod{\phi} = 3^{-1} \pmod{20}$

i.e. find a value for d such that ϕ divides $(ed-1)$

i.e. find d such that 20 divides $3d-1$.

Simple testing ($d = 1, 2, \dots$) gives $d = 7$

Check: $ed-1 = 3 \cdot 7 - 1 = 20$, which is divisible by ϕ .

Step 5: Public key = $(n, e) = (33, 3)$

Private key = $(n, d) = (33, 7)$.

This is the smallest possible value for the modulus n for which the RSA Algorithm works.

Now say we want to encrypt the message $m = 7$,

Step 6: Encrypt this message using $c = m^e \bmod n$, $c = 7^3 \bmod 33 = \mathbf{13}$

Step 7: To check decryption we compute $m = c^d \bmod n = 13^7 \bmod 33 = \mathbf{7}$.

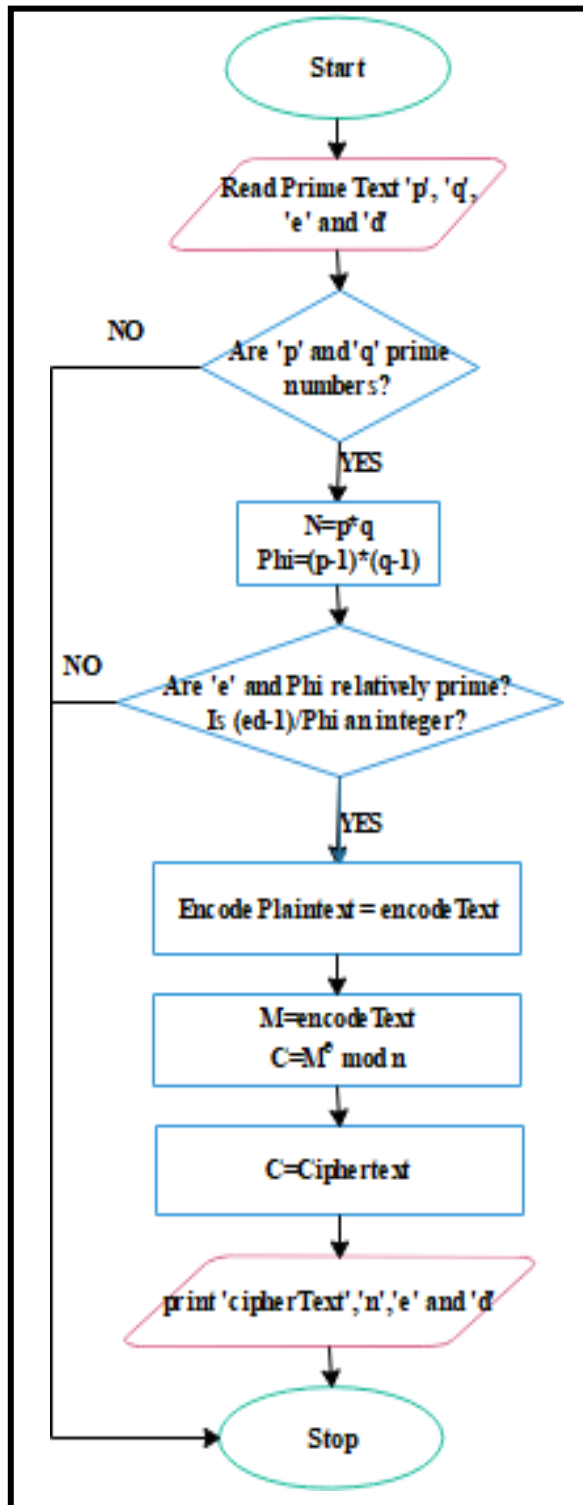


Figure 2.4: Flowchart of RSA encryption and decryption operations

2.4. Application of Prime Numbers in Cryptography

More precisely, some crucial cryptographic algorithms like RSA heavily rely on the fact that prime factorization of huge integers takes a long time. We have a "secret key" made up of those two prime numbers that are used to decrypt the message and a "public key" made up of the product of those two huge prime numbers. Everyone can use the public key to encrypt messages for us, but only we have access to the prime factors and are able to decrypt the messages. We can make the public key available to the general public. Given the state of the art of number theory, everyone else would have to factor the number, which takes too long to be useful.

2.5. Restrictions on the Selection of Prime Numbers

- i) It is best to choose \mathbf{p} and \mathbf{q} as primes so that it is computationally infeasible to factor $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$. The main requirement for \mathbf{p} and \mathbf{q} is that they must roughly have the same bit length and be sufficiently large to avoid the elliptic curve factoring algorithm. For instance, each of \mathbf{p} and \mathbf{q} should be roughly 1024 bits long if a 2048-bit modulus \mathbf{n} is to be used.
- ii) Another restriction on the primes \mathbf{p} and \mathbf{q} is that the difference $\mathbf{p} - \mathbf{q}$ should not be too small. If $\mathbf{p} - \mathbf{q}$ is small, then $\mathbf{p} \approx \mathbf{q}$ and hence $\mathbf{p} \approx \sqrt{\mathbf{n}}$. As a result, \mathbf{n} might be efficiently factored by just doing trial division by all odd numbers close to \mathbf{n} .
- iii) In addition to these constraints, in cryptography, it is recommended by several scholars that \mathbf{p} and \mathbf{q} be strong primes. The following three requirements must all be met for a prime \mathbf{p} to be considered a strong prime:
 - (a) $\mathbf{p} - 1$ has a large prime factor, denoted \mathbf{r} ;
 - (b) $\mathbf{p} + 1$ has a large prime factor; and
 - (c) $\mathbf{r} - 1$ has a large prime factor.

In number theory, a strong prime is a prime number that is greater than the arithmetic mean of the closest prime above and below (in other words, it's closer to the following than to the preceding prime). Or, to state it algebraically, writing the prime number series as $(p_1, p_2, p_3, \dots) = (2, 3, 5, \dots)$, p_n is a strong prime if $p_n > (p_{n-1} + p_{n+1})/2$. For instance, the seventh prime number is **17**; the sixth and eighth prime numbers, **13** and **19**, add up to **32**, which is half of **64**, and since **17** is greater than **32**, it is a strong prime.

A prime can be a strong prime both in the number theoretic sense and the cryptographic sense. For example, since the arithmetic mean of its two nearby primes is 62 less, the number 439351292910452432574786963588089477522344331 is a strong prime in the number theoretic sense.

This number, **439351292910452432574786963588089477522344330**, would be a strong prime in the sense of cryptography without the aid of a computer because it has the large prime factors **1747822896920092227343** (and the number one less has the large prime factor **1683837087591611009**), and **439351292910452432574786963588089477522344332**. (and in turn, the number one less than that has the large prime factor **105646155480762397**).

These numbers would be challenging to factor manually, even with methods more sophisticated than trial division. These integers factor almost instantly in a modern computer algebra system. A strong prime in terms of cryptography must be substantially bigger than this example.

2.6. RSA Digital Signature Scheme

A digital signature is a method of authentication that lets the sender of a message attach a code that serves as a signature. The message's hash is typically used to create the signature, which is then encrypted using the creator's private key. The message's integrity and origins are guaranteed by the signature. A digital signature is used to secure the authenticity of digital messages or documents. A message is signed by a secret key of the sender to produce a signature and the signature is verified against the message by a public key. Thus, anyone can verify the signatures, but only one party holding the secret key can sign messages. Digital signatures are used widely in e-commerce applications, banking applications, software distribution, and in other cases where jurisdiction is involved and it

is important to detect permanent addresses. The digital signature provides three types of services such as:

Authentication: This is a procedure to ensure that received messages come from a valid source. It must verify the signature's time and date as well as its creator.

Message integrity: requires that the contents be verified at the time of the signature and not changed while the data is being transferred. We cannot obtain the same signature if the message has been changed.

Non-repudiation: This term refers to the inability of the signer (or sender) to claim having signed a message or document [18].

Public and private keys are generated by the sender in the RSA digital signature algorithm. To sign the digital message, the private key is used. After then, the recipient will receive the signed message. The recipient generates a new verification value from the signed message that was received to validate the authenticity of digitally signed data by using the sender's public key. The recipient then compares the verified value to the value of the original message. The message is validated if the two values matches.

2.6.1. The RSA Digital Signature Algorithms:

Key generation algorithm (generated by the receiver, Bob)

Alice must do the following:

1. Choose two prime numbers (p , q) randomly, secretly, and roughly of the same size.
2. Compute the modulus n as follows: $n = p * q$.
3. Compute the $\Phi(n)$ as follows: $\Phi(n) = (p-1) * (q-1)$.
4. Choose the key e , such that $1 < e < \Phi(n)$, and $\text{GCD}(e, \Phi(n)) = 1$.
5. Compute the private key d , such as $d = e^{-1} \text{ mod } \Phi(n)$.

Send the public (n , e) to Bob.

Signature and verification algorithms Signature (sender - Alice)

Alice must do the following:

6. Determine the message m to be signed such that $0 < m < n$.

7. Sign the message as follows: $S = m^d \bmod n$.
8. Send the signatures with the message m to Bob (receiver).
Verification (receiver - Bob) - Bob must do the following:
9. Obtain the keys (d, n) .
10. Obtain s, m from Alice.
11. Compute v as follows: $v = s^e \bmod n$.
12. Verify the message m as follows: is $m=v$?

2.7. Number Theory behind RSA

2.7.1. Prime Generation and Integer Factorization

Two basic facts and one conjecture in number theory prepare the way for today's RSA public-key cryptosystem.

FACT 1. Prime generation is easy: Finding a random prime number of a specific size is simple.

It is simple to determine whether a number is a prime, even a large prime since prime numbers of any size are very common.

There are numerous effective techniques to multiply two large numbers.

CONJECTURE 3. Factoring is hard: Given such an n , it appears to be quite hard to recover the prime factors p and q .

Finding the factors of a large number generally takes a lot of time, even though the problem has been studied for hundreds of years. The most efficient methods available now are much faster than the straightforward strategy of testing each potential factor one at a time [19].

2.8. Practical applications of the RSA Algorithm

RSA has been extensively applied in several domains. The algorithm continues to this day to be the most widely used in commercial systems. It is used:

- To protect web traffic, in the SSL protocol (Security Socket Layer),

- To ensure email confidentiality and authenticity in PGP (Pretty Good Privacy)
- To ensure a remote SSH connection (Secure Shell).
- Moreover, it is a crucial component of the SET protocol (Secure Electronic Transaction), which is used in modern payment systems.

Most digital data, information, and phone security applications use RSA. The RSA has the benefit of being a trusted and secure system, but it also has the drawback of being incredibly slow in processing data. This is the reason behind its usage in hybrid cryptographic systems, which combine public key algorithms (RSA) for the secure transmission of the symmetric key (or session key) required for message encryption and decryption [20].

2.9. Advantages and Disadvantages of the RSA Algorithm

The RSA algorithm is renowned for its improved convenience and security. Private keys are not disclosed or transmitted using the RSA technique. Contrarily, with a secret-key system, keys can only be exchanged manually or through a communication channel. This is due to the fact that the same key is used for encryption and decryption in secret-key systems, and there is a possibility that an adversary could discover this secret key during transmission.

Digital signatures can be provided by public-key systems. There is no way to revoke these digital signatures. In contrast, authentication in secret-key systems necessitates either the disclosure of some crucial secrets or the engagement of a third party. This could lead to the sender rejecting the previously verified message by asserting that one of the parties involved has compromised the shared secret. The fundamental drawback of public-key cryptography is speed. Because efficiency and security are always trade-offs. The ideal key size for RSA is tiny, however, this compromises security in numerous ways. Numerous secret-key techniques are quicker than asymmetric encryption. Private-key encryption is used in conjunction with public-key encryption.

2.10. Execution Speed of RSA Algorithm

Electronic commerce applications like online banking and shopping are quickly gaining popularity due to the ongoing expansion of network infrastructure. RSA public-key

cryptosystem is an essential encryption algorithm for authenticating the identity of the person at the other end of a communication line and for exchanging secret data safely and is also the existent standard. Before the emergence of more powerful computers and cryptanalysis tools, 512-bit RSA encryption was thought to be secure enough. Before the emergence of more powerful computers and cryptanalysis tools, 512-bit RSA encryption was thought to be secure enough. However, 2048-bit RSA is now strongly advised for high-security systems. Large-speed RSA hardware accelerators are crucial for smartcards in electronic commerce systems, especially for authentication servers with a high volume of accesses.

The bottleneck of the RSA operation is its modular exponentiation for very long numbers. In hardware, RSA is around 1000 times slower than DES. With a 512-bit modulus, the fastest Very Large-Scale Integration (VLSI) hardware implementation of RSA has a throughput of 64 kilobits per second. In software also, DES is around 100 times faster than RSA [21].

These figures might slightly change as technology advances, but RSA will never be as quick as symmetric algorithms because public key encryption uses longer keys than symmetric encryption.

2.11. Related Works

K. Somsuk, T. Chiawchanwattana, and C. Sanemueang (2018) investigated to speed up the RSA decryption process with large sub-exponents using improved Chinese Remainder Theorem (CRT). Here the basic idea of CRT is, to speed up the RSA decryption process. Suppose Bob has the private key \mathbf{d} and also knows the primes \mathbf{p} and \mathbf{q} ($\mathbf{n}=\mathbf{p}*\mathbf{q}$) and compute with \mathbf{p} and \mathbf{q} rather than \mathbf{n} .

Step1: Modular representation of cipher text \mathbf{y} in \mathbf{p} and \mathbf{q} . Compute this as follows:

$$\mathbf{y_p} = (\mathbf{y} \bmod \mathbf{p})$$

$$\mathbf{y_q} = (\mathbf{y} \bmod \mathbf{q})$$

Step 2: Modular exponentiation with \mathbf{p} and \mathbf{q} . Compute exponents as follows:

$$\mathbf{d_p} = (\mathbf{d} \bmod (\mathbf{p}-1))$$

$$\mathbf{dq} = (\mathbf{d} \bmod (\mathbf{q}-1))$$

Now compute the exponentiation of the plain text \mathbf{xp} and \mathbf{xq} which are represented in modulo \mathbf{p} and \mathbf{q} individually as follows:

$$\mathbf{xp} = (\mathbf{yp}^{\mathbf{dp}} \bmod \mathbf{p})$$

$$\mathbf{xq} = (\mathbf{yq}^{\mathbf{dp}} \bmod \mathbf{q})$$

Step 3: Assemble the final result \mathbf{x} and do a couple of more complications as follows:

$$\mathbf{cp} = \mathbf{q}^{-1} \bmod \mathbf{p} \text{ or } \mathbf{cp} * \mathbf{q} \equiv \mathbf{1} \bmod \mathbf{p}.$$

Read as \mathbf{cp} equals \mathbf{q} inverse of modulo \mathbf{p} ; or \mathbf{cp} times \mathbf{q} is congruent to $\mathbf{1}$ modulo \mathbf{p} (i.e. Which number do I have multiply by \mathbf{q} in such a way that its remainder is $\mathbf{1}$ when divided by \mathbf{p}).

Similarly, compute for \mathbf{cq} as follows:

$$\mathbf{cq} = \mathbf{p}^{-1} \bmod \mathbf{q} \text{ or } \mathbf{cp} * \mathbf{p} \equiv \mathbf{1} \bmod \mathbf{q}.$$

Read as \mathbf{cq} equals \mathbf{p} inverse of modulo \mathbf{q} ; or \mathbf{cp} times \mathbf{p} is congruent to $\mathbf{1}$ modulo \mathbf{q} (i.e. Which number do I have multiply by \mathbf{q} in such a way that its remainder is $\mathbf{1}$ when divided by \mathbf{p}). Here the value of \mathbf{cp} and \mathbf{cq} is whole numbers, not fractions.

$\mathbf{x} = (\mathbf{q} * \mathbf{cp}) \mathbf{xp} + (\mathbf{p} * \mathbf{cq}) \mathbf{xq} \bmod \mathbf{n}$; The underlined cofficeints can be precomputed.

It can be shown that the CRT method can speed up the decryption operation of the RSA algorithm by a factor of four. However, in applying RSA-CRT encryption operation, the encryption algorithm does not change. Because the public key exponent \mathbf{e} is the same as the standard RSA.

The improved technique to speed up RSA's decryption process with large values of \mathbf{dp} and \mathbf{dq} is presented by modifying some processes of CRT-RSA. The proposed method suits to solve the problem with high values of \mathbf{dp} and \mathbf{dq} which are chosen before finding \mathbf{d} . Additionally, ciphertext and sub-exponents must be changed to implement their proposed method [6].

S. Vollala, V. V. Varadhan, K. Geetha, and N. Ramasubramanian (2014) presented an efficient modular multiplication algorithm for public-key cryptography like the RSA

Diffie and Hellman and the ElGamal schemes. They proposed an efficient algorithm for computing $a^x \bmod n$ and $a^x b^y \bmod n$. In the study, they proposed four algorithms to evaluate modular exponentiation. Bit forwarding (BFW) algorithms to compute $a^x \bmod n$, and to compute $a^x b^y \bmod n$ two algorithms namely Substitute and reward (SRW), Store, and Forward (SFW) were proposed.

The proposed algorithms namely BFW1 and BFW2 refer to bit forwarding 1 bit and bit forwarding 2 bits respectively. These algorithms perform a smaller number of multiplications in comparison with the binary exponential algorithm. In BFW1 if there are two consecutive ones in the exponent, one bit can be forwarded thereby reducing one multiplication for each pair of consecutive ones. The second algorithm BFW2 is a two-bit forward binary exponentiation method, in which if there are three consecutive ones in the exponent then two bits at a time can be forwarded so that it reduces two multiplications for every three consecutive ones [22].

F. H. M. S. Al-Kadei, H. A. Mardan, and N. A. Minas (2020) conducted research to speed up image encryption using the RSA algorithm. Three experiments were developed to examine the execution time of encryption and decryption processes. Moreover, some programming techniques were used to speed up these processes.

The image is taken as input, in the Properties section, the image properties are displayed, such as the file name, image resolution, and image size. After the image is loaded, the HEX function extracts the image's HEX code, and the HEX code is converted to encrypted text depending on the RSA settings. Conversely, the encryption text is loaded, the RSA algorithm is applied, the text is decrypted, and the resulting string is converted to an image.

In experiment 1, a picture of 240 Kb and resolution 303 * 538 was selected. This image was chosen for RSA algorithm coding. In this algorithm, a key length of 1024 Kb was used. In this experiment, the execution time is recorded with the public key length for its calculation with the decryption process of the file itself to compare the time of the encryption and decryption process.

In Experiment 2, they tested the system designed by calculating the time required to

perform encryption and decryption of the image. In this image, they selected a picture of a certain size to encode it using the RSA algorithm. In this experiment they choose a general key for encryption of a certain length, then they have recorded the results in both processes, and then compare them with the results of the second experiment to compare with each other to conclude.

In Experiment 3, they made software modifications that reduce the time it takes to encrypt and decrypt data [23].

S. Vollala, B. S. Begum, A. D. Joshi, and N. Ramasubramanian (2016) Presented the modular exponential techniques in higher radix namely, radix-4 and radix8. These algorithms are compatible with the hardware implementations. The modular multiplications involved in modular exponentiation are evaluated with the help of the Montgomery multiplication method. They have introduced modifications in the existing Montgomery method and named as Adapted Montgomery Multiplication (AMM) Technique [24].

Abouelkheir, E., & El-Sherbiny, S. (2022) In this work, a new modification of the RSA (Rivest–Shamir–Adleman) algorithm has proposed to enhance the performance of conventional RSA up on application in audio cryptosystems. The performance was investigated by measuring some audio quality metrics. The metrics that were estimated during the experimental test include the Execution Time, Segmental Signal-to-Noise Ratio (SSNR), Linear Predictive Code measure (LPC), Cepstral Distance Measure (CD), and Mean Square Error between the original signal and the encrypted signal [25].

Issah Mubasir, Alhassan Abdul-Barik, and Alhassan Salamudeen (2021) Proposed the use of the Rivest-Shamir-Adleman (RSA) algorithm to implement a system for encrypting text files of any length (by breaking long messages into valid blocks and encrypting each block) capable of being transmitted using a Simple Mail Transfer Protocol (SMTP). Probable primes, 3048 bits in length are generated to be used in the generation of public-private key pairs for encryption and decryption [26].

Venkatesh, K., Pratibha, K., Annadurai, S., & Kuppusamy, L. (2019) Presented a reconfigurable architecture for pre-computation methods to compute modular exponentiation and thereby speeding up RSA and Diffie-Hellman like protocols [27].

2.12. Summary of Related Works

Table 2.0: Summary of Related Works

Sr.#	Paper Title	Authors	Description and findings	Tools and techniques used	Shortcomings
1	Speed Up Image Encryption by Using RSA Algorithm	F. H. M. S. Al-Kadei, H. A. Mardan and N. A. Minas (2020)	In this paper, the authors have developed three experiments to examine the execution time of the processes of RSA encryption and decryption and compare the results together to elicit improved points and discuss them.	Java	Limited to only the encryption of images.
2	High-Radix Modular Exponentiation for Hardware Implementation of Public-Key Cryptography	S. Vollala, B. S. Begum, A. D. Joshi and N. Ramasubramanian(2016)	This paper presented the modular exponential techniques in higher radix namely, radix-4 and radix-8.	Mont-gomery Multiplication (AMM) Technique	The problem of overflowing a finite computation range
3	Speed up RSA's Decryption Process with Large sub Exponents using Improved CRT	K. Somsuk, T. Chiawchanwattana, and C. Sanemueang (2018)	Proposed to speed up RSA's decryption process by modifying some processes of CRT-RSA.	Improved CRT	It is only applicable for decryption.

4	Enhancement of Speech Encryption/Decryption Process Using RSA Algorithm Variants	Abouelkheir, E., & El- Sherbiny, S. (2022)	The dynamic keys with five primes were employed to increase security and speed.	MATLAB	Even though the proposed work uses five prime numbers to boost the algorithm's security level, it lengthens the system's running time.
5	Fast Implementation of the Rivest-Shamir-Adleman (RSA) Algorithm with Robust Packet Data Loss Detection Function	Issah Mubasir, Alhassan Abdul-Barik and Alhassan Salamudeen (2021)	This study proposed a method for encrypting text files of arbitrary size using the Rivest-Shamir-Adleman (RSA) algorithm (by breaking long messages into valid blocks and encrypting each block) capable of being transmitted using a Simple Mail Transfer Protocol (SMTP).	Java	The system's limitation is that it only encrypts text files.
6	Reconfigurable Architecture to Speed-up Modular Exponentiation	Venkatesh, K., Pratibha, K., Annadurai, S., & Kuppusamy, L. (2019).	In this paper, they have presented a reconfigurable architecture for pre-computation methods to compute modular exponentiation and thereby speeding up RSA and Diffie-Hellman like protocols.	VHDL hardware description language and Sim SE simulator	Instead of considering both the hardware and software aspects of modular multiplication, the study largely concentrated on the hardware aspect.

Chapter Three

Proposed Computing Model of Modular Exponentiation for RSA Cryptosystem

3.1. Introduction

This chapter is devoted to explaining the methods that were followed by this study. It describes the methodologies that were employed, the rationale behind the selection, and the proposed system.

3.2. Computational Aspects of the RSA Algorithm

The modular exponentiation technique is a crucial but time-consuming method used in a wide range of scientific investigations and real-world applications, particularly those involving modern cryptology.

In RSA, both encryption and decryption involve raising an integer to an integer power, mod n . If the exponentiation is done over the integers and then reduced modulo n , the intermediate values would be enormous. To implement the RSA Algorithm for huge integer values, we need to focus on the calculation of the remainder value for the exponential function of both encryption and decryption. The first modular exponentiation rule states that we should not compute $C = M^e \pmod{n}$ by first exponentiation M^e and then dividing to get the remainder $C = (M^e) \% n$. At each exponentiation step, the temporary results must be reduced modulo n . This is due to the large space needed for the binary number M^e . We need bits to store M^e , assuming M and e each have 256 bits.

$$\log_2(M^e) = e \cdot \log_2(M) \approx 2^{256} \cdot 256 = 2^{264} \approx 10^{80}$$

This number is roughly equivalent to the total number of particles in the universe. We don't have the means to store it.

Let's determine the number of modular multiplications required to calculate $M^e \pmod{n}$. The naive method of calculating $C = M^e \pmod{n}$ is to start with $C = M \pmod{n}$ and continue executing the modular multiplication operations $C = C \cdot M \pmod{n}$ until $C = M^e \pmod{n}$ is obtained. The naive method would be difficult for large values of e because it

requires $e - 1$ modular multiplication to compute $C = M^e \pmod n$. For example, if we need to calculate $M^{15} \pmod n$, this method computes all powers of M until 15:

$$M \rightarrow M^2 \rightarrow M^3 \rightarrow M^4 \rightarrow M^5 \rightarrow M^6 \rightarrow M^7 \rightarrow \dots \rightarrow M^{15} \text{ [28]}$$

3.3. Modular Exponentiation

Public key cryptography requires raising big powers to bases and then reducing the result using modulo functions, which is computationally expensive. This method is referred to as modular exponential. In practice, the modular exponential function must be both fast and efficient for the RSA algorithm to be effective. Modular exponentiation is exponentiation performed over a modulus. It has applications in computer science, particularly in public-key cryptography, where it is used in both Diffie-Hellman Key Exchange and RSA public/private keys. Figure 3.0 shows the simple modular operations.

$$\begin{array}{l} \hline (u+v)\pmod n = ((u \pmod n) + (v \pmod n)) \pmod n \\ \hline (u-v)\pmod n = ((u \pmod n) - (v \pmod n)) \pmod n \\ \hline (u*v)\pmod n = ((u \pmod n) * (v \pmod n)) \pmod n \\ \hline \end{array}$$

Figure 3.0 Properties of Modular Arithmetic

The remainder obtained by multiplying an integer b (the base) by an exponent e (the exponent) and dividing the result by a positive integer n (the modulus) is known as modular exponentiation; it is represented by the formula $c = b^e \pmod n$. It follows from the definition of division that $0 \leq c < n$.

For instance, given $b = 5$, $e = 3$ and $n = 13$, dividing $5^3 = 125$ by 13 leaves a remainder of $c = 8$. Modular exponentiation can be performed with a negative exponent e by finding the modular multiplicative inverse d of b modulo n using the extended Euclidean algorithm. That is $c = b^e \pmod n = d^{-e} \pmod n$, where $e < 0$ and $b * d \equiv 1 \pmod n$.

Even for extremely big integers, modular exponentiation is an effective computation method. On the other hand, it is said to be challenging to compute the modular discrete logarithm, which involves determining the exponent e when given b , c , and n . The one-way function characteristic of modular exponentiation makes it a candidate for use in cryptography methods.

3.4. The Naive Modular Exponentiation Method

The most way to determine a modular exponent is to determine $\mathbf{b^e}$ directly, then take this number and modulo n . Suppose we're attempting to determine \mathbf{c} with $\mathbf{b = 7}$, $\mathbf{e = 13}$, and $\mathbf{n = 437}$: $\mathbf{c \equiv 7^{13} \pmod{437}}$.

To calculate 7^{13} , one may use a calculator; the result is 96,889,010,407. The solution \mathbf{c} is found to be $\mathbf{273}$ by taking this value modulo 437.

3.4.1. Memory Efficient Method

This technique of modular multiplications is applied repeatedly to make each operation relatively faster, saving time as well as the efficiency of memory. This formula uses the property: $\mathbf{(a * b) \pmod{m} = [(a \pmod{m}) * (b \pmod{m})] \pmod{m}}$.

Algorithm 3.0 Computing $\mathbf{a^b \pmod{n}}$ Using the Naive Approach

1. Set $\mathbf{c = 1}$, $\mathbf{e' = 0}$.
2. Increase $\mathbf{e'}$ by 1.
3. Set $\mathbf{c = (b * c) \pmod{n}}$.
4. If $\mathbf{e' < e}$, go to step 2. Else, \mathbf{c} contains the correct solution to $\mathbf{c \equiv b^e \pmod{n}}$.

This algorithm simply counts up $\mathbf{e'}$ by ones until $\mathbf{e'}$ reaches \mathbf{e} , doing a multiply by \mathbf{b} and performing a modulo operation each time it adds one (to ensure the results stay small). This technique uses $\mathbf{e-1}$ modular multiplications, which makes it inefficient.

The larger value of \mathbf{e} in cryptography determines the security, and efficiency depends on how efficiently these modular multiplications and modular exponential functions can be solved. If we use this naive technique, the plaintext, cipher text, or even partial cipher text, which is intended to be of high value, will need a lot of modular multiplications.

For instance: with $m = 7$, $e = 13$, and $n = 437$, the naive modular exponential will solve $m^e \bmod n$ as shown in figure 3.2. and gives the result as **273**.

1.	$e' = 1 * c = (1 * 7) \bmod 437 = 7 \bmod 437 = 7.$
2.	$e' = 2 * c = (7 * 7) \bmod 437 = 49 \bmod 437 = 49.$
3.	$e' = 3 * c = (49 * 7) \bmod 437 = 343 \bmod 437 = 343.$
4.	$e' = 4 * c = (343 * 7) \bmod 437 = 2401 \bmod 437 = 216.$
5.	$e' = 5 * c = (216 * 7) \bmod 437 = 1512 \bmod 437 = 201.$
6.	$e' = 6 * c = (201 * 7) \bmod 437 = 1407 \bmod 437 = 96.$
7.	$e' = 7 * c = (96 * 7) \bmod 437 = 672 \bmod 437 = 235.$
8.	$e' = 8 * c = (235 * 7) \bmod 437 = 1645 \bmod 437 = 334.$
9.	$e' = 9 * c = (334 * 7) \bmod 437 = 2338 \bmod 437 = 153.$
10.	$e' = 10 * c = (153 * 7) \bmod 437 = 1071 \bmod 437 = 197.$
11.	$e' = 11 * c = (197 * 7) \bmod 437 = \bmod 437 = 68.$
12.	$e' = 12 * c = (68 * 7) \bmod 437 = 476 \bmod 437 = 39.$
13.	$e' = 13 * c = (39 * 7) \bmod 437 = 273 \bmod 437 = \mathbf{273}$

Figure 3.2: Naive Modular Exponentiation Example

3.5. Modular Exponentiation and Roots

Given the background of Prime Generation and Integer Factorization facts and conjecture in chapter two, n will hereafter denote the product of two large, randomly generated primes. Let e be an odd integer between 3 and $n-1$ that is relatively prime to $p-1$ and $q-1$, and let m and c be integers between 0 and $n-1$. This implies that the following equation is true:

$$\text{GCD}(e, (p-1)*(q-1))=1$$

Two more facts and one more conjecture is the basis for the encryption and decryption processes in the RSA public-key cryptosystem:

FACT4. Modular exponentiation is easy: Given n , m , and e , it's easy to compute $c = m^e \bmod n$

Formally, the value $\mathbf{m}^e \bmod \mathbf{n}$ is obtained by multiplying e copies of \mathbf{m} , dividing the result by \mathbf{n} , and keeping the remaining part. This computation may seem to be expensive because it involves $e-1$ multiplications by \mathbf{m} with increasing intermediate results, followed by a division by \mathbf{n} . But, two optimizations simplify the process.

The number of multiplications can be reduced to no more than twice the size of e in binary by using an appropriate sequence of previous intermediate values instead of only \mathbf{m} .

The intermediate results are kept at the same size as \mathbf{n} by dividing and taking the leftover value after each multiplication [29].

By performing a modular exponentiation procedure with another odd integer \mathbf{d} between 3 and $\mathbf{n}-1$, the value \mathbf{m} can be extracted from the result \mathbf{c} . For this \mathbf{d} , in particular, the following is true for all \mathbf{m} :

FACT5. Modular root extraction – the reverse of modular exponentiation – is easy given the prime factors: Given \mathbf{n} , \mathbf{e} , \mathbf{c} , and the prime factors \mathbf{p} and \mathbf{q} , it's easy to recover the value \mathbf{m} such that $\mathbf{c} = \mathbf{m}^e \bmod \mathbf{n}$.

$$\mathbf{m} = (\mathbf{m}^e)^d \bmod \mathbf{n} \text{ and}$$

If \mathbf{e} , \mathbf{p} , and \mathbf{q} are known, it is simple to calculate the integer \mathbf{d} .

CONJECTURE6. Modular root extraction is otherwise hard: Given only \mathbf{n} , \mathbf{e} , \mathbf{c} , but not the prime factors, it appears to be quite hard to recover the value \mathbf{m} .

We need to calculate \mathbf{d} from the values of \mathbf{e} , \mathbf{p} , and \mathbf{q} , where \mathbf{d} is the multiplicative inverse of \mathbf{e} modulo (\mathbf{n}).

3.6. The Modified Modular Exponentiation Method

Most of the time, a problem arises that requires exponentiation, and usually, a crude approach to compute these values suffices. However, every rule has an exception, so knowing a few tips to ramp up efficiency. This is where the naive method is replaced by rapid modular exponentiation, which offers a considerably more effective solution to the problem.

In disciplines like cryptography and primality testing, such methods' efficiency is essential.

Based on our proposed approach, the modular exponentiation algorithm, expressed as $a^b \bmod n$, is computed using the following pseudo code.

Note: The integer b is expressed as a binary number $b_k b_{k-1} \dots b_0$

Algorithm 3.1 Pseudo code for Computing $a^b \bmod n$ Using Modified Method

```
    c ← 0; f ← 1
    for i ← k downto 0
        do c ← 2 * c
           f ← (f * f) mod n
        if bi = 1
            then c ← c+1
                f ← (f * a) mod n
    return f
```

Due to its oscillations between the two operations, this technique uses the square-multiply approach. Suppose we want to calculate $c \equiv a^b \bmod n$:

- Write the letter "**b**" in binary form.
- Starting from the left (MSB-Most Significant Bit): for each '1' in 'b' square the result then multiply by 'a' and for each '0' square the result only.
- Calculate the modular of the result at each step.

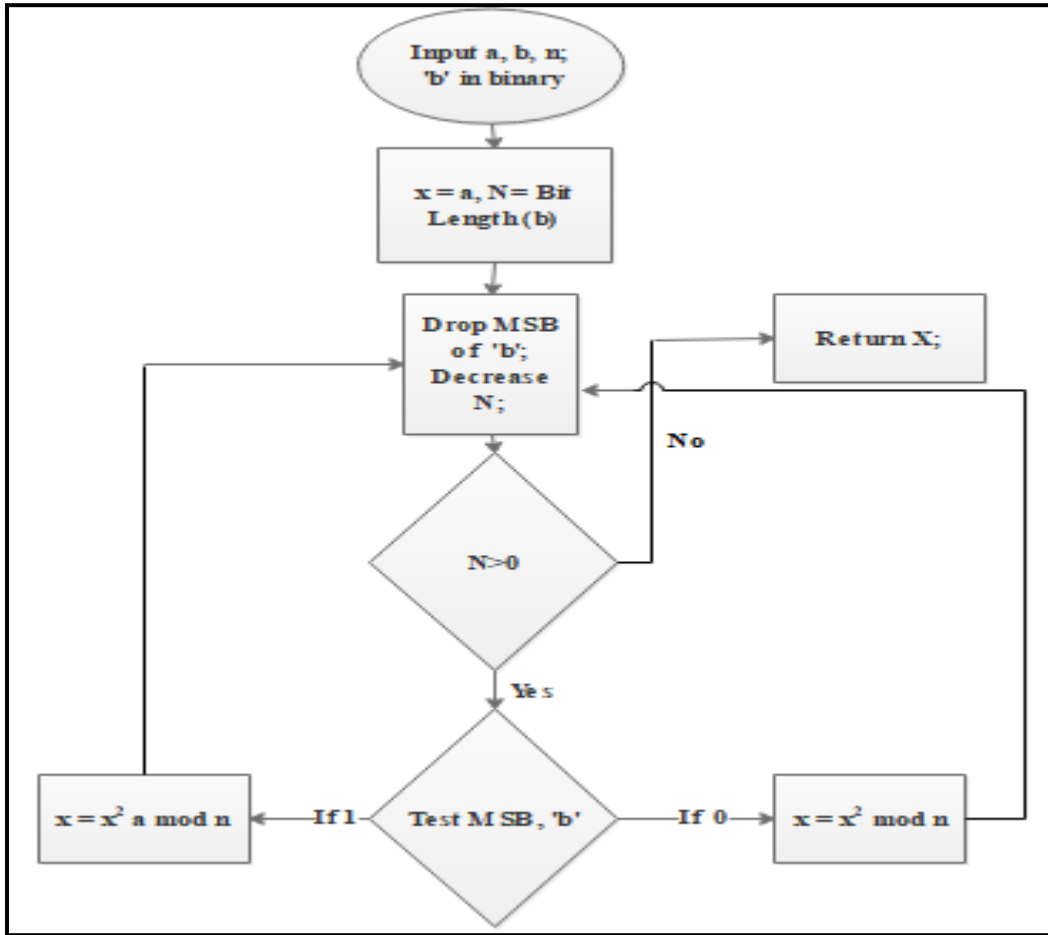


Figure 3.3: Flowchart for Computing $a^b \bmod n$ Using Modified Method

Example:

Table 3.1: Result of the Modified Modular Exponentiation Algorithm for $a^b \bmod n$, where $a = 240$, $b = 262(10) = 100000110(2)$, $n = 14$

i	8	7	6	5	4	3	2	1	0
bi	1	0	0	0	0	0	1	1	0
c	1	2	4	8	16	32	65	131	262
f	240	4	2	4	2	4	4	4	2

Therefore, the computed value of $a^b \bmod n = 240^{262} \bmod 14$ using our modified modular exponentiation approach is **2**. The variable **c** is not needed; it is included for explanatory purposes. The final value of **c** is the value of the exponent.

Modular exponentiation as an application is interesting due to its significance in cryptography. This modified modular exponentiation method can also be applied to the Diffie-Hellman Key exchange algorithm.

Chapter Four

Implementation, Performance Analysis, and Results

4.1. Overview

This Chapter details the purpose of modifications made to the standard RSA encryption and decryption processes with the aid of experimental implementation and analysis. Here the principle is clarified by using artificially small parameters. The method, however, is generally applicable to all suitably chosen parameters.

4.2. Implementation

The practical implementation is performed using MATLABR2019a programming platform installed on windows 10 pro that was running on Intel(R) Core (TM) i7-4610M CPU @ 3.00GHz 8 GB RAM and 64-bit Operating System. The selection of this programming platform is due to the fact that MATLAB toolboxes are professionally developed, thoroughly examined, and well-documented. MATLAB comes with a huge library of predefined functions that provides tested and prepackaged solutions to many technical tasks. It provides a vast library of mathematical operations for numerical integration, cryptography, statistics, linear algebra, and more [30].

Using a prime generation process based on the Miller-Rabin Method, the tests are performed and analyzed by generating initial random primes for each of the cases. The test is carried out for the various bit sizes of initial primes, which range from 28, 56, 128, 256, and 512. For the help of performance evaluation and analysis, we have supplied the same input values for both the standard and modified work in each experimental test. To minimize the effect of random errors, we performed repeated measurements using the same inputs until we get precise values and then we take their average. The prime numbers are generated randomly using the **Linux OpenSSL** command. Figure 4.0 shows an illustration to generate a random prime number having a 256-bit size.

```
debe@debe-VirtualBox:~$ openssl
OpenSSL> prime -generate -bits 256
102979233234342051657689998763634022637841593360229597674584728652922896985123
OpenSSL> █
```

Figure 4.0: Generating a random prime number having a 256-bit size

4.2.1. Standard RSA (SRSA) Implementation

Here the naive approach of modular multiplications was applied to perform the encryption and decryption process of the RSA cryptosystem. Initially, the standard RSA algorithm has been implemented using MATLAB2019a programming language. As an illustration of this implementation, an experimental output having a total bit size of 512 is employed for prime numbers p and q (256 bits for each) as shown in figure 4.1. The computational steps were performed as follows:

Step 1: At the beginning, we run the Matlab program's **Standard_RSR.m** code. Here Matlab 2019a-based simulation tool is used.

Step 2: Then, we entered two large prime numbers, p , and q , and calculated their product n , which is the modulus for encryption and decryption.

Step 3: The calculated public and private RSA keys are displayed in figure 4.2 after entering the values and pressing enter key.

Step 4: Figure 4.2 also shows that when asked to "Enter the message," we can enter any text, in this case, we input the message "**Hi**" as an illustration.

Step 5: After entering the message, the program will display the ASCII code, Ciphertext of the ASCII message, Decrypted ASCII message, and Decrypted Message as the entered message.

```
MATLAB R2019a
HOME PLOTS APPS EDITOR PUBLISH VIEW
C:\Users\hp\Documents\MATLAB
Command Window
Implementation of the Standard RSA Algorithm
Please Enter a Large Prime Number for p: 112428278689657166897681503455592410258705395370639294210643915906815280087489
Please Enter a Large Prime Number for q: 94075467037394908318964446059690911083847332213682933131189177566407220942131
```

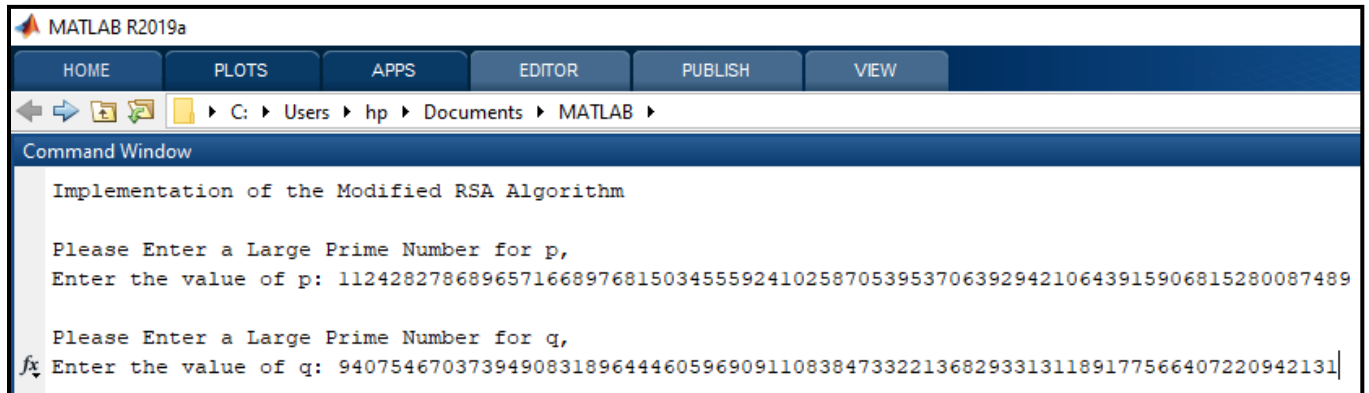
Figure 4.1: Implementation of SRSA Algorithm for p and q values

```
MATLAB R2019a
HOME PLOTS APPS EDITOR
C:\Users\hp\Documents\MATLAB
Command Window
The value of (N) is: 1.057674282593989e+154
The public key (e) is: 5
The value of (Phi) is: 1.057674282593989e+154
The private key (d) is: 1.078827768245869e+155
The Value of p:
  1.1243e+77
The Value of q:
  9.4075e+76
Enter the message: Hi
ASCII Code of the Entered Message:
  72  105
Encrypted /Ciphertext/ of the Entered Message:
  72  28
Decrypted ASCII of Message:
  72  105
Decrypted /Plaintext/ of the Message: Hi
Elapsed time is 1.281019 seconds.
```

Figure 4.2: ASCII code and Decrypted Message of SRSA

4.2.2. Modified RSA (MRSA) Implementation

Likewise, the proposed work was implemented using the same programming platform and equal input size of data to that of the standard RSA cryptographic approach. However, in this case, we have applied the square-multiply / fast modular exponentiation/ approach to encrypt and decrypt the supplied message by converting the public exponent(e) and secret exponent(d) into the binary number format. Figure 4.4 shows the experimental output of the modified RSA cryptosystem which is tested using the same inputs as that of the standard RSA.

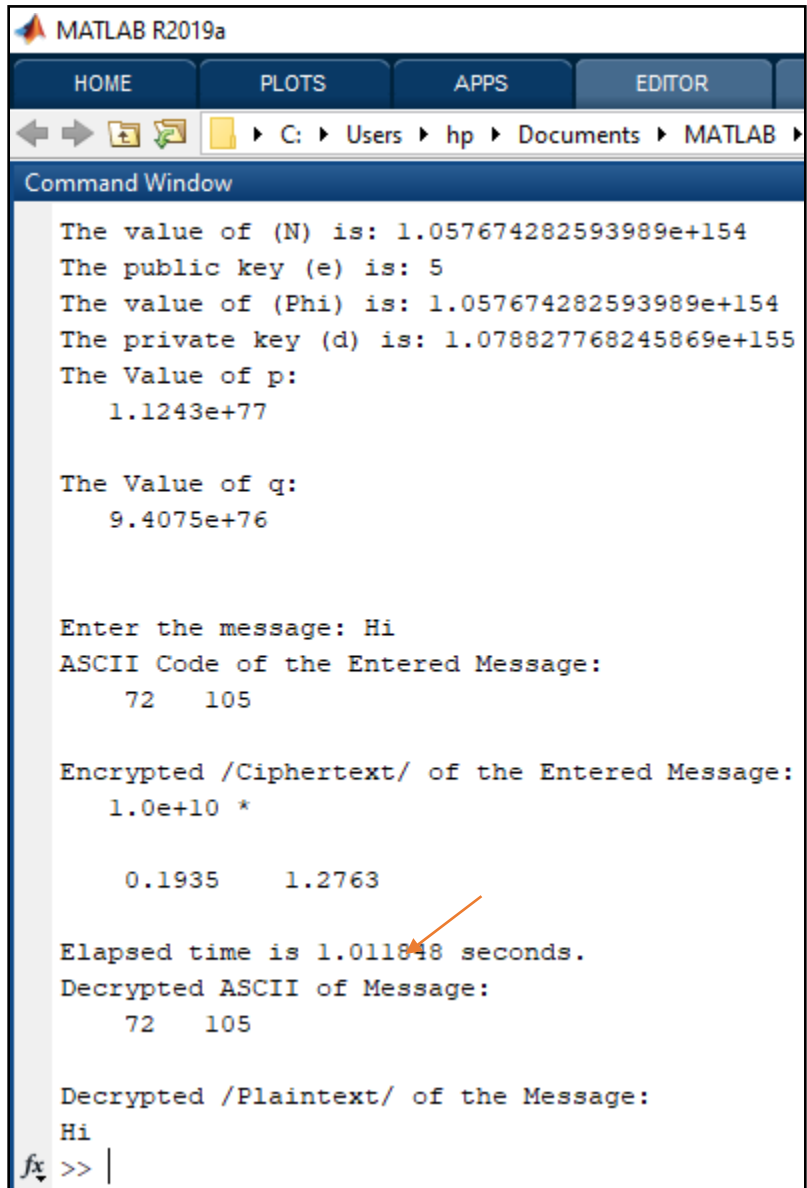


```
MATLAB R2019a
HOME PLOTS APPS EDITOR PUBLISH VIEW
C:\Users\hp\Documents\MATLAB
Command Window
Implementation of the Modified RSA Algorithm

Please Enter a Large Prime Number for p,
Enter the value of p: 112428278689657166897681503455592410258705395370639294210643915906815280087489

Please Enter a Large Prime Number for q,
fx Enter the value of q: 94075467037394908318964446059690911083847332213682933131189177566407220942131|
```

Figure 4.3: Implementation of MRSA Algorithm for p and q values



```
MATLAB R2019a
HOME PLOTS APPS EDITOR
C:\Users\hp\Documents\MATLAB
Command Window
The value of (N) is: 1.057674282593989e+154
The public key (e) is: 5
The value of (Phi) is: 1.057674282593989e+154
The private key (d) is: 1.078827768245869e+155
The Value of p:
    1.1243e+77
The Value of q:
    9.4075e+76
Enter the message: Hi
ASCII Code of the Entered Message:
    72    105
Encrypted /Ciphertext/ of the Entered Message:
    1.0e+10 *
    0.1935    1.2763
Elapsed time is 1.011848 seconds.
Decrypted ASCII of Message:
    72    105
Decrypted /Plaintext/ of the Message:
Hi
>> |
```

Figure 4.4: ASCII code and Decrypted Message of MRSA

4.3. Performance Analysis and Results

Our suggested RSA algorithm was compared to the standard RSA in terms of performance analysis metrics like Key Encryption and Decryption Time, Throughput, Memory Utilization, and Avalanche Effect. Before using any algorithm or technique for a real-world application, these performance parameters have to be considered.

4.3.1. Encryption Time

Encryption time is the amount of time needed to transform plaintext into ciphertext. The length of the key used, the type of mode employed, and the size of the plain text blocks are all factors that affect how quickly data is encrypted. The proposed work uses seconds as the measurement unit. The algorithm's performance is shown by the encryption time, which also indicates how quickly the system will react to inputs. Figure 4.5 shows the comparison of encryption time in seconds among the Standard RSA (SRSA) and Modified RSA (M RSA) algorithms. The result shows that the Modified RSA algorithm takes less amount of encryption time than the Standard one. From Table 4.2, it is found that the M RSA algorithm has improved the encryption performance of SRSA by 17.46442 %.

Table 4.0: Summary of Encryption Time for the Standard and Modified RSA Algorithm

Bit Size	28 Bit	56 Bit	128 Bit	256 Bit	512 Bit
Time (Seconds)					
SRSA	1.034064	1.036526	1.072717	1.144744	1.205565
M RSA	1.008957	1.008999	1.010322	1.011292	1.019878

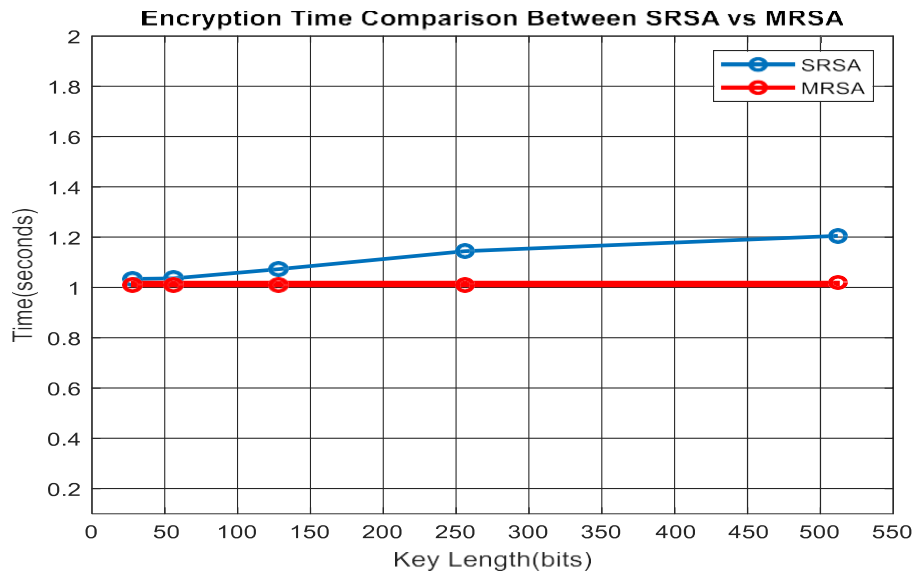


Figure 4.5: Analysis of Encryption Performance (in seconds)

4.3.2. Decryption Time

Decryption time is the amount of time it takes an algorithm to decipher a given cipher text and return the original plaintext. The performance of the system is also affected by the reverse encryption procedure. The algorithm determines the numerous useful applications in the real world based on the decryption time. Figure 4.6 shows the comparison of decryption time in seconds among the Standard RSA (SRSA) and Modified RSA (MRSA) algorithms. Likewise, the encryption time, this result also shows that the Modified RSA algorithm takes less amount of encryption time than the Standard RSA algorithm. According to Table 4.3, the MRSA algorithm has an 18.70409 % improvement over the SRSA algorithm in terms of decryption performance.

Table 4.1: Summary of Decryption Time for the Standard and Modified RSA Algorithm

Bit Size	28 Bit	56 Bit	128 Bit	256 Bit	512 Bit
Time (Seconds)					
SRSA	1.036230	1.037971	1.072871	1.173844	1.22741
MRSA	1.008957	1.009952	1.010431	1.011325	1.059358

We have constructed the comparison in Table 4.2 and Table 4.3 based on the encryption and decryption time of the Standard and Modified RSA algorithms in Tables 4.0 and Table 4.1.

Table 4.2: Comparison of the encryption time of SRA and MRSA algorithms (%)

Key Length (bits)	MRSA	SRSA	Time Difference
28	1.008957 sec	1.034064 sec	5.834393 %
56	1.008999 sec	1.036526 sec	6.308298 %
128	1.010322 sec	1.072717 sec	13.08824 %
256	1.011292 sec	1.144744 sec	26.1322 %
512	1.019878 sec	1.205565 sec	35.95899 %
Average	1.01189 sec	1.098723 sec	17.46442 %

Table 4.3: Comparison of the decryption time of SRA and MRSA algorithms (%)

Key Length (bits)	MRSA	SRSA	Time Difference
28	1.008957 sec	1.036230 sec	6.254945 %
56	1.009952 sec	1.037971 sec	6.496501 %
128	1.010431 sec	1.072871 sec	13.107 %
256	1.011325 sec	1.173844 sec	31.22943 %
512	1.059358 sec	1.22741 sec	36.43259 %
Average	1.020005 sec	1.109665 sec	18.70409 %

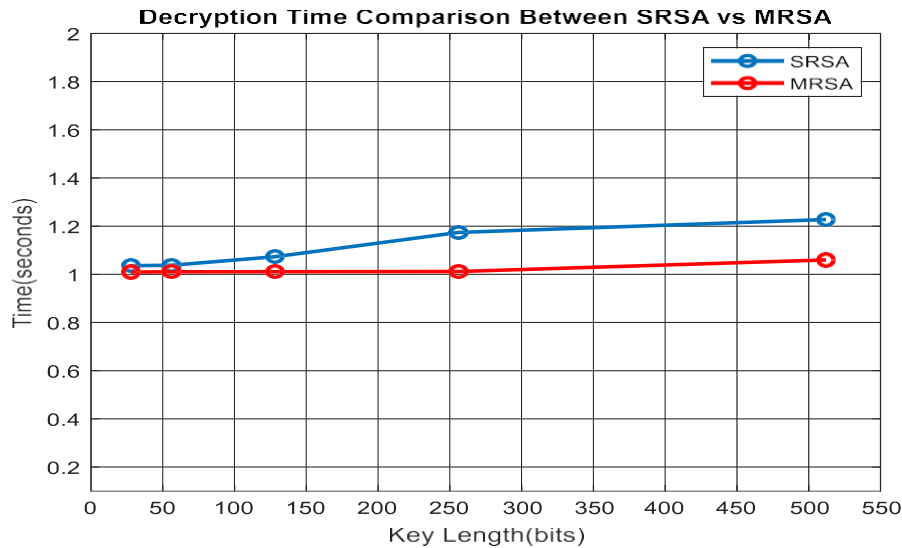


Figure 4.6: Analysis of Decryption Performance (in seconds)

4.3.3. Memory Utilization

The quantity of memory required for the encryption or decryption process is referred to as the memory consumed, and it primarily depends on the internal structure of the algorithm. Based on memory use, this metric is crucial in determining whether the method is space-efficient. Figure 4.7 and Figure 4.8 shows a sample experimental output for a 1280-byte message encrypted using 512 key lengths. According to Table 4.5, it is observed that the memory consumption of the MRSA algorithm is slightly higher by 1.05849 % than the SRSA. As seen in Figure 4.9, the Modified RSA algorithm uses more memory than the Standard RSA algorithm because to implement the modified system, we have utilized an array list which consumes more space. As a result, it's not recommended to utilize the Modified RSA algorithm for memory-constrained-based systems.

```
Decrypted /Plaintext/ of the Message: Video provides a powerful way to help you prove y
Memory Consumption By the Modified RSA Algorithm:
-----
Memory consumed by the MATLAB process in bytes:
1.4353e+09
```

Figure 4.7: Snapshot showing Memory consumed by the MRSA

```
Decrypted /Plaintext/ of the Message: Video provides a powerful way to help you pr
Memory Consumption By the Standard RSA Algorithm:
-----
Memory consumed by the MATLAB process in bytes:
1.4139e+09
```

Figure 4.8: Snapshot showing Memory consumed by the SRSA

Table 4.4: Summary of Memory Usage for the Standard and Modified RSA Algorithm

Input Size (Bytes)	Algorithms	Evaluation Process	Memory Usage(MB)	Key Length (bits)
1280	SRSA	Encryption & Decryption	1329.3	56
	MRSA		1353.6	
1280	SRSA	Encryption & Decryption	1355.9	64
	MRSA		1373.7	
1280	SRSA	Encryption & Decryption	1393.2	128
	MRSA		1413.2	
1280	SRSA	Encryption & Decryption	1435.8	256
	MRSA		1442.0	
1280	SRSA	Encryption & Decryption	1442.3	512
	MRSA		1446.4	

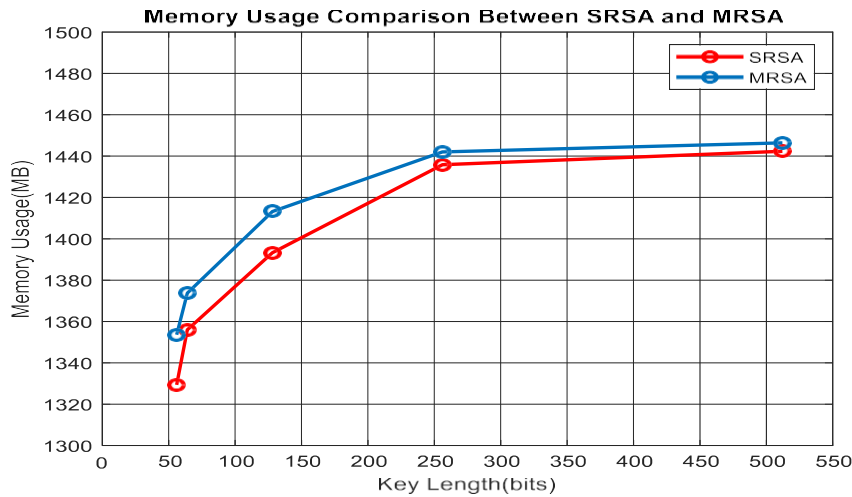


Figure 4.9: Analysis of Memory Usage (MB)

Table 4.5: Comparison of the memory consumption of SRA and MRSA algorithms (%)

Input Size (Bytes)	Key Length (bits)	MRSA	SRSA	Memory Consumption Difference (%)
1280	56	1353.6 MB	1329.3 MB	-1.82803 %
	64	1373.7 MB	1355.9 MB	-1.31278 %
	128	1413.2 MB	1393.2 MB	-1.43554 %
	256	1442.0 MB	1435.8 MB	-0.43182 %
	512	1446.4 MB	1442.3 MB	-0.28427 %
Average		1405.78 MB	1391.3 MB	-1.05849 %

4.3.4. Avalanche Effect

The avalanche effect is one of the key features of cryptography. It occurs when an input data set is minimally modified by changing one or more bits, dramatically altering the output. The avalanche effect suggests that a slight change in the plaintext (or key) should result in a significant change to the cipher text.

The purpose of the avalanche effect is to make it more difficult to do cipher text analysis while trying to come up with an attack because even a small change might result in a significant change. It aids in evaluating an algorithm's level of security.

$$\text{Avalanche Effect} = \frac{\text{Number of Flipped Bits in Cipher text}}{\text{Number of Bits in Cipher text}} * 100\% \quad (1)$$

To perform the test, we change some plaintext characters. Table 4.6 and Table 4.7 show the analysis of the Avalanche Effect due to some character changes in plaintext from different positions when the key is constant (512-bit).

If we flip in some characters (bits) in the plaintext, for example, from the word "**CRYPTOGRAPHY**", we get different cipher text outputs as shown in Table 4.6 and Table 4.7. From the result, we can see that the cipher text is very strong for very simple plaintext character changes in both algorithms.

Figure 4.8 shows the avalanche effect of SRSA and MRSA. Although more than half of the bits in the cipher text are changed in both algorithms, the MRSA exhibits a 14.53511% higher avalanche value than the SRSA. Therefore, we can deduce that the Modified RSA algorithm exhibits a relatively higher avalanche effect than the Standard RSA algorithms which depicts the secureness of the algorithm.

Table 4.6: Avalanche Test of the SRSA

Plaintext	Ciphertext	# of changed characters	# of flipped bits in ciphertext	Avalanche Effect (%)
CRYPTOGRAPHY 0100001101010010010110010 1010000010101000100111101 0001110101001001000001010 100000100100001011001	&Zà 00011110001001100001100 10001001101011010111000 000001111000001000	1	33	33/64*100%= 51.56%
CrYPTOGRAPHY 0100001101110010010110010 1010000010101000100111101 0001110101001001000001010 1000001001000010110010010 0000]UÀg 00111010100101011101110 10001110011101000100110 1101110111011001111100			
CRyPTOGrAPHY 0100001101010010011110010 1010000010101000100111101 0001110111001001000001010 1000001001000010110010010 0000]¼éj 01011101101111001001010 01110100101101010000011 101010011001111100	2	34	34/64*100%= 53.13%
CRyPToGrAPHY 0100001101010010011110010 1010000010101000110111101 0001110111001001000001010 1000001001000010110010010 0000	:Yèwg 00111010100101011101110 10001110011101000100110 1101110111011001111100	3	36	36/68*100= 54.94%
CRyPToGrAPHy 0100001101010010011110010 1010000010101000110111101 00011101110010 0100000101010000010010000 1111001	u+9Ñ6iÓ 01110101001010111011101 00011100111010001001101 101110111011010011100	4	38	38/68*100%= 55.88%

Table 4.7: Avalanche Test of the MRSA

Plaintext	Ciphertext	# of changed characters	# of flipped bits in ciphertext	Avalanche Effect (%)
CRYPTOGRAPHY 0001111000100110000110010 0010011010110101110000000 01111000001000	□&□□Zà□□ 00011110001001100001100 10001001101011010111000 000001111000001000			
CrYPTOGRAPHY 0100001101110010010110010 1010000010101000100111101 0001110101001001000001010 1000001001000010110010010 0000	5ÖÄÑŷñà 00110101110101011100010 01101000110100101111100 01111000001000	1	37	37/61=60.66%
CRyPTOGrAPHY 0100001101010010011110010 1010000010101000100111101 0001110111001001000001010 1000001001000010110010010 0000	□áöeã°□nèô 10001110111000011111011 00110010111110101101110 10100000110110111011101 00011110100	2	54	54/81=66.67%
CRyPToGrAPHY 0100001101010010011110010 1010000010101000110111101 0001110111001001000001010 1000001001000010110010010 0000	□áöeãDdxÛô 10001110111000011111011 00110010111100110010001 00011001000111100011011 00011110100	3	51	51/80=63.75%
CRyPToGrAPHy 0100001101010010011110010 1010000010101000110111101 00011101110010 0100000101010000010010000 1111001	□áöeãDdxÛ□ 10001110111000011111011 00110010111100110010001 00011001000111100011011 00100010100	4	50	50/81=61.73%

Table 4.8: Comparison of the avalanche effect of SRSA and MRSA algorithms (%)

# of changed characters	SRSA	MRSA	Avalanche Value Difference (%)
1	51.56 %	60.66 %	15.00165 %
2	53.13 %	66.67 %	20.30898 %
3	54.94 %	63.75 %	13.81961 %
4	55.88 %	61.73 %	9.476754 %
Average	53.8775 %	63.2025 %	14.53511% %

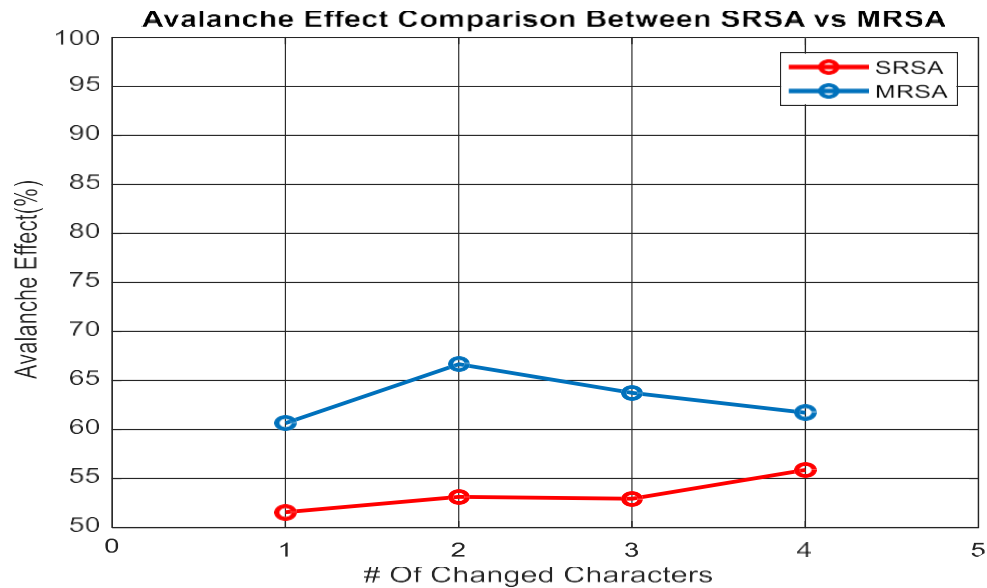


Figure 4.10: Analysis of Avalanche Effect (%)

4.3.5. Throughput

Throughput is the measure of an algorithm's capacity to process data in a given amount of time. The higher the throughput, the more efficient the algorithm is, and the less battery power it uses because throughput and battery usage are inversely correlated. Figure 4.9 and Figure 4.10 show the comparison of encryption throughput and decryption throughput in Byte per second (Bps) respectively among the Standard RSA (SRSA) and Modified RSA (M RSA) algorithms. According to Table 4.13 and Table 4.14, the M RSA algorithm has 15.80883 % and 15.12928 % performance improvements over the SRSA algorithm in terms of encryption and decryption throughput respectively.

$$\text{Encryption Throughput} = \frac{\text{Plain Text}}{\text{Encryption Time}} \quad (2)$$

$$\text{Decryption Throughput} = \frac{\text{Plain Text}}{\text{Decryption Time}} \quad (3)$$

Table 4.9: Summary of Encryption Time for the Standard and Modified RSA Algorithm Using Different Message Sizes

Plaintext Size (Byte)	386	788	1340	2110	3070
Algorithms	Encryption Time (Seconds)				
SRSA	1.231376	1.25715	1.272265	1.310744	1.374953
M RSA	1.019957	1.050202	1.079777	1.117768	1.161019

Table 4.10: Summary of Encryption Throughput for the Standard and Modified RSA Algorithm

Algorithms	Encryption Throughput (Bps)				
SRSA	313.47	626.81	1053.24	1609.77	2232.80
MRSA	378.45	750.33	1241.00	1887.69	2644.23

(Byte/second)

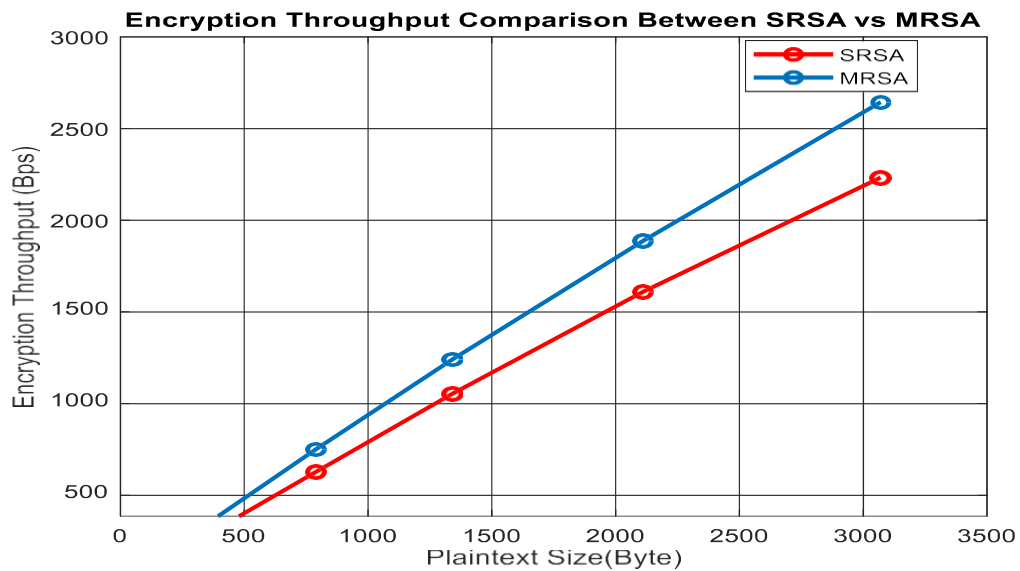


Figure 4.11: Analysis of Encryption Throughput (BPs)

Table 4.11: Summary of Decryption Time for the Standard and Modified RSA Algorithm Using Different Message Sizes

Plaintext Size (Byte)	386	788	1340	2110	3070
Algorithms	Decryption Time (Seconds)				
SRSA	1.234020	1.236230	1.278710	1.331844	1.39741
MRSA	1.028957	1.05952	1.090431	1.1301325	1.18935

Table 4.12: Summary of Decryption Throughput for the Standard and Modified RSA Algorithm (Byte/second)

Algorithms	Decryption Throughput (Bps)				
SRSA	312.89	637.42	1047.93	1584.27	2196.92
MRSA	375.14	743.73	1228.87	1867.04	2581.24

Table 4.13: Comparison of the encryption throughput of SRA and MRSA algorithms (%)

Plaintext Size (Byte)	SRSA	MRSA	Throughput Difference (%)
386	313.47 Bps	378.45 Bps	17.17004 %
788	626.81 Bps	750.33 Bps	16.46209 %
1340	1053.24 Bps	1241.00 Bps	15.12973 %
2110	1609.77 Bps	1887.69 Bps	14.72276 %
3070	2232.80 Bps	2644.23 Bps	15.55954 %
Average	1167.218 Bps	1380.34 Bps	15.80883 %

Table 4.14: Comparison of the decryption throughput of SRA and MRSA algorithms (%)

Plaintext Size (Byte)	SRSA	MRSA	Throughput Difference (%)
386	312.89 Bps	375.14 Bps	16.5938 %
788	637.42 Bps	743.73 Bps	14.29417 %
1340	1047.93 Bps	1228.87 Bps	14.7241 %
2110	1584.27 Bps	1867.04 Bps	15.14536 %
3070	2196.92 Bps	2581.24 Bps	14.88897 %
Average	1155.886 Bps	1359.204 Bps	15.12928 %

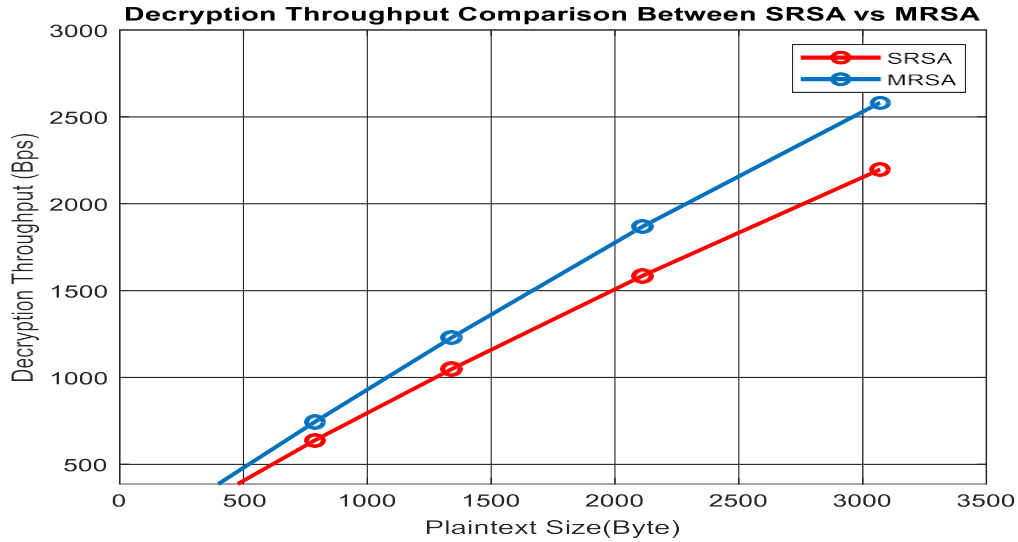


Figure 4.12: Analysis of Decryption Throughput (BPs)

4.4. Discussions

In this chapter, we presented the experimental aspects associated with both the standard and the improved model of the RSA cryptosystem. The main objective of this work was to implement a modified modular exponentiation algorithm to speed up the RSA's execution time. In this research work, different performance evaluation metrics such as encryption, decryption time, memory utilization, avalanche effect, encryption, and decryption throughput were employed to perform the investigation. While comparing the results obtained using the baseline model and the improved model related to different performance evaluation metrics, we found various experimental outcomes. The experimental results show that the MRSA algorithm has a **17.46442 %** and **18.70409%** improvement over the SRSA algorithm in terms of encryption and decryption time performance. The MRSA also exhibits a **14.53511%** higher avalanche value than the SRSA which depicts its secureness. It also **15.80883 %** and **15.12928 %** performance improvements over the SRSA algorithm in terms of encryption and decryption throughput respectively. However, the memory consumption of the MRSA algorithm is slightly higher by **1.05849 %** than the SRSA because to implement it, an array list that consumes more space is employed.

Chapter Five

Conclusion and Recommendations

5.1. Conclusion

In this thesis work an improvement of the RSA algorithm is presented. The implementation of the proposed technique was performed using the MATLAB programming platform. In the proposed algorithm, we have attempted to improve the encryption and decryption time of the standard RSA algorithm. The new algorithm based on modified RSA has been proposed and designed by applying the square-multiply technique of modular exponentiation.

The proposed cryptosystem is implemented and its performance is evaluated using various metrics in both encryption and decryption processes. The results obtained demonstrated that the modified cryptosystem improves the existing modular exponentiation algorithms, significantly reducing the encryption and decryption execution time. However, our algorithm uses more memory than the Standard RSA algorithm that is because to implement the modified system, we have utilized an array list which consumes more space. The Modified RSA algorithm also exhibits a higher encryption and decryption throughput when compared to the Standard RSA algorithm. The Modified RSA algorithm also exhibits considerably higher avalanche values when compared to the Standard RSA algorithm, which shows how secure the algorithm is.

5.2. Recommendations

Additional features, improvements, and modifications should be incorporated to come up with an effective and efficient cryptographic algorithm. We recommend utilizing larger key lengths of prime numbers p and q to support the large transfer of messages and better security of the system. Furthermore, the proposed work is recommended for communication domains where speed performance and throughput with better are vital issues. However, when storage is a vital issue of the domain, the standard RSA cryptosystem is the feasible choice.

5.3. Contributions

The followings are some of the contributions that can be made by this research work:

- The proposed RSA cryptosystem uses a simple mathematical technique of computation that most individuals easily understand.
- A huge amount of data can be transferred between the CPU and the memory in a considerable time.
- Better real-time system performance is achieved by the simultaneous processing of data while the program is running.

5.4. Future Work

We encourage ourselves to concentrate on the implementation of the RSA cryptosystem, which has a large key size, in the upcoming work, since it will dramatically increase public-key cryptography's security. Furthermore, we want to extend our efforts on improving the RSA algorithm's key generation time. Future research will also consider digital signatures, such as Pretty Good Privacy (PGP), which are mathematical schemes for representing the authenticity of messages. This will increase the security and authenticity of the data. The scheme could also be extended to encrypt other multimedia files such as image, audio, and video files.

References

- [1] Stallings, W. (2006). *Cryptography and network security principles and practices* 4th edition.
- [2] Chen, F., Wang, J., Li, J., Xu, Y., Zhang, C., & Xiang, T. (2022). TrustBuilder: A non-repudiation scheme for IoT cloud applications. *Computers & Security*, *116*, 102664.
- [3] Pournaghi, S.M., Bayat, M. & Farjami, Y. MedSBA: a novel and secure scheme to share medical data based on blockchain technology and attribute-based encryption. *J Ambient Intell Human Comput* **11**, 4613–4641 (2020). <https://doi.org/10.1007/s12652-020-01710-y>
- [4] Dhakar, R. S., Gupta, A. K., & Sharma, P. (2012, January). Modified RSA encryption algorithm (MREA). In *2012 second international conference on advanced computing & communication technologies* (pp. 426-429). IEEE.
- [5] Castelvechi, Davide (2020-10-30). "Quantum-computing pioneer warns of complacency over Internet security". *Nature*. 587 (7833): 189. Bibcode:2020Natur.587..189C. doi:10.1038/d41586-020-03068-9. PMID 33139910. S2CID 226243008.
- [6] Somsuk, K., Chiawchanwattana, T., & Sanemueang, C. (2018, October). Speed up RSA's Decryption Process with Large sub Exponents using Improved CRT. In *2018 International Conference on Information Technology (InCIT)* (pp. 1-5). IEEE.
- [7] Castelvechi, Davide (2020-10-30). "Quantum-computing pioneer warns of complacency over Internet security". *Nature*. 587 (7833): 189. Bibcode:2020Natur. 587..189C. doi:10.1038/d41586-020-03068-9. PMID 33139910. S2CID 226243008
- [8] Acosta, A. J., Addabbo, T., & Tena-Sánchez, E. (2017). Embedded electronic circuits for cryptography, hardware security and true random number generation: an overview. *International Journal of Circuit Theory and Applications*, *45*(2), 145-169.
- [9] Yusfrizal, Y., Meizar, A., Kurniawan, H., & Agustin, F. (2018, August). Key management using combination of Diffie–Hellman key exchange with AES encryption. In *2018 6th International Conference on Cyber and IT Service Management (CITSM)* (pp. 1-6). IEEE
- [10] Kartit, Zaid (February 2016). "Applying Encryption Algorithms for Data Security in Cloud Storage, Kartit, et al"

- [11] Nwoye, Chinedu. (2015). Design and Development of an E-Commerce Security Using RSA Cryptosystem. *International Journal of Innovative Research in Information Security (IJIRIS)*. Volume 6. 5-17].
- [12] R. Imam, Q. M. Areeb, A. Alturki and F. Anwer, "Systematic and Critical Review of RSA Based Public Key Cryptographic Schemes: Past and Present Status," in *IEEE Access*, vol. 9, pp. 155949-155976, 2021, doi: 10.1109/ACCESS.2021.3129224.
- [13] Underwood, R.G. (2022). Public Key Cryptography. In: *Cryptography for Secure Encryption*. Universitext. Springer, Cham. https://doi.org/10.1007/978-3-030-97902-7_9
- [14] Kessler, G. C. (2015). An overview of cryptography.
- [15] Johannes Landin (14:29, 27 October 2020).| Vectorized version of File:Public_key_encryption_keys.png|
- [16] Barker, E. and Roginsky, A. (2019), Transitioning the Use of Cryptographic Algorithms and Key Lengths, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, [online], <https://doi.org/10.6028/NIST.SP.800-131Ar2> (Accessed February 28, 2023)
- [17] Abid, R., Iwendi, C., Javed, A.R. *et al.* An optimised homomorphic CRT-RSA algorithm for secure and efficient communication. *Pers Ubiquit Comput* (2021).
- [18] Chia, Jason; Chin, Ji-Jian; Yip, Sook-Chin (2021-09-16). "Digital signature schemes with strong existential unforgeability". *F1000Research*. 10: 931. doi:10.12688/f1000research.72910.1. S2CID 239387758
- [19] Jahan, I., Asif, M., & Rozario, L. J. (2015). Improved RSA cryptosystem based on the study of number theory and public key cryptosystems. *American Journal of Engineering Research (AJER)*, 4(1), 143-149.
- [20] Sahu, S., Singh, J., & Ashraf, J. (2015). Encryption & Decryption of Text Data with RSA cryptography using MATLAB. *International Journal of Science & Engg*, 3, 104-110.
- [21] E.F. Brickell, -Survey of Hardware Implementations of RSA,| *Advances in Cryptology—CRYPTO '89 Proceedings*, Springer-Verlag, 1990, pp. 368–370.
- [22] S. Vollala, V. V. Varadhan, K. Geetha and N. Ramasubramanian, "Efficient modular multiplication algorithms for public key cryptography," 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 74-78, doi:

10.1109/IAdCC.2014.6779297.

[23] F. H. M. S. Al-Kadei, H. A. Mardan and N. A. Minas, "Speed Up Image Encryption by Using RSA Algorithm," *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2020, pp. 1302-1307, doi: 10.1109/ICACCS48705.2020.9074430.

[24] S. Vollala, B. S. Begum, A. D. Joshi and N. Ramasubramanian, "High-radix Modular Exponentiation for hardware implementation of Public-Key Cryptography," *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, Pune, India, 2016, pp. 346-350, doi: 10.1109/CAST.2016.7914992.

[25] Abouelkheir, E., & El-Sherbiny, S. (2022). Enhancement of Speech Encryption/Decryption Process Using RSA Algorithm Variants. *Human-centric Computing and Information Sciences*, 12.

[26] Mubasir, I., Abdul-Barik, A., & Salamudeen, A. (2021). Fast Implementation of the Rivest-Shamir-Adleman (RSA) Algorithm with Robust Packet Data Loss Detection Function. *Asian Journal of Engineering and Applied Technology*, 10(2), 19-24.

[27] Venkatesh, K., Pratibha, K., Annadurai, S., & Kuppusamy, L. (2019, October). Reconfigurable architecture to speed-up modular exponentiation. In *2019 International Carnahan Conference on Security Technology (ICCST)* (pp. 1-6). IEEE.

[28] M.R. Gauthama Raman, Dr. S. Kaja Mohideen, 2014, Modified Modular Exponentiation for a Faster Implementation of RSA algorithm on FPGA, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCICCT – 2014 (Volume 2 – Issue 05).

[29] S. Sharma, J.S. Yadav, P. Sharma. Modified RSA Public Key Cryptosystem Using Short Range Natural Number Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering*, volume 2, Issue 8, August 2012.

[30] L. Yu, "Matlab Programming Environment Based on Web," *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Chongqing, China, 2018, pp. 509-512, doi: 10.1109/ITOEC.2018.8740716

Appendices

Appendix A- MATLAB Sample Code for the Standard RSA Algorithm

```
clc;
disp('Implementation of the Standard RSA Algorithm');
close all;
p = input('\nPlease Enter a Large Prime Number for p: ');
q = input('\nPlease Enter a Large Prime Number for q: ');
[Pk,Phi,d,e] = initialize(p,q);
disp('The Value of p: ');
disp(p);
disp('The Value of q: ');
disp(q);
M = input('\nEnter the message: ','s');
x=length(M);
c=0;
for j= 1:x
    for i=0:122
        if strcmp(M(j),char(i))
            c(j)=i;
        end
    end
end

disp('ASCII Code of the Entered Message:');
disp(c);

% % %Encryption
for j= 1:x
    cipher(j)= crypt(c(j),n,e);
end
disp('Encrypted /Ciphertext/ of the Entered Message:');
disp(cipher);

% % %Decryption
for j= 1:x
    message(j)= crypt(cipher(j),Pk,d);
end
tic
disp(['Decrypted /Plaintext/ of the Message: ' c]);
user = memory;
disp('Memory Consumption By the Standard RSA Algorithm:');
disp('-----');
disp('Memory consumed by the MATLAB process in bytes:');
disp(user.MemUsedMATLAB)
    pause(1)
    toc
```

```
function mc = crypt(M,N,e)
k = 65535;
c = M;
cf = 1;
cf=mod(c*cf,N);
for i=k-1:-1:1
    c = mod(c*c,N);
    j=k-i+1;
end
mc=cf;
end
```

Appendix B- MATLAB Sample Code for the Modified RSA Algorithm

```
% RSA = Ron Rivest, Adi Shamir, and Leonard Adleman
%% %% MATLAB Code By:
Debebe Kebede (BSc.) &
Yilekal Mulualem (PhD.)
  clc;
status=0;

%% Implementation of the Modified RSA Algorithm

disp('Implementation of the Modified RSA Algorithm');
[Pk,Phi,d,e] = initialize(p,q);

%% Necessary Inputs:

while status == 0
  disp(' ');
  disp('Please Enter a Large Prime Number for p,');
  p = input('Enter the value of p: ');
  status = isprime(p);
end

status=0;

while status == 0
  disp(' ');
  disp('Please Enter a Large Prime Number for q,');
  q = input('Enter the value of q: ');
  status = isprime(q);
end

%% Calculation of Public and Private Keys

[n,Phi,d,e] = rsa(p,q);

disp('The Value of p: ');
disp(p);
disp('The Value of q: ');
disp(q);

M = input('\nEnter the message: ','s');

x=length(M);  %Size of Entered Message

for j= 1:x
  cipher(j)= crypt(c(j),n,e);
end

disp('Encrypted /Ciphertext/ of the Entered Message:');
disp(cipher);
```



```

disp('Decrypted ASCII of Message:');
disp(c);

disp(['Decrypted /Plaintext/ of the Message: ' c]);
user = memory;
disp('Memory Consumption By the Modified RSA Algorithm:');
disp('-----');
disp('Memory consumed by the MATLAB process in bytes:');
disp(user.MemUsedMATLAB)
status=0;

seperater=" ";

for j= 1:x
    if(j==x)
        temp=num2str(cipher(j));
        txstring = strcat(txstring,temp);
    else
        if(j==1)
            txstring=num2str(cipher(j));
            txstring = strcat(txstring,seperater);
        else
            temp=strcat(num2str(cipher(j)),seperater);
            txstring = strcat(txstring,temp);
        end
    end
end

%% function to check that input numbers p and q are prime numbers

function result = isprime(number)
result=true;
% check if number is a nonnegative integer
if floor(number)~=number || number<0
    result=false;
    return
end

end

```

```

%% Function to Calculate and Assign Values using p and q
% Calculates Phi, n, d (used for decryption/ private key), e (used for
encryption/ public key)
function [n,Phi,d,e] = rsa(p,q)
clc;
disp('Intaializing:');
n=p*q;
Phi=(p-1)*(q-1);
%Calculate the value of e
x=2;e=1;
while x > 1
    e=e+1;
    x=gcd(Phi,e);
end
%Calculate the value of d
i=1;
r=1;
while r > 0
    k=(Phi*i)+1;
    r=rem(k,e);
    i=i+1;
end
d=k/e;
clc;
disp(['The value of (N) is: ' num2str(n)]);
disp(['The public key (e) is: ' num2str(e)]);
disp(['The value of (Phi) is: ' num2str(Phi)]);
disp(['The private key (d) is: ' num2str(d)]);
end

%% Function to perform cryptography operations - Encryption and Decryption
using Keys
% Public for Encryption, Private for Decryption
function mc = crypt(M,N,e)
e=decimaltobinary(e);
k = 65535;
c = M;
cf = 1;
cf=mod(c*cf,N);
for i=k-1:-1:1
    c = mod(c*c,N);
    j=k-i+1;
    if e(j)==1
        cf=mod(c*cf,N);
    end
end
mc=cf;
end

```

```

%% Function to convert Decimal into Binary to Carry out Encryption
Decryption mathematical operations
function a = decimaltobinary(d)
i=1;
a=zeros(1,65535);
while d >= 2
    r=rem(d,2);
    if r==1
        a(i)=1;
    else
        a(i)=0;
    end
    i=i+1;
    d=floor(d/2);
end
if d == 2
    a(i) = 0;
else
    a(i) = 1;
end
x=[a(16) a(15) a(14) a(13) a(12) a(11) a(10) a(9) a(8) a(7) a(6) a(5) a(4)
a(3) a(2) a(1)];
end

```

DEBRE BERHAN UNIVERSITY
COLLEGE OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY
MASTERS OF COMPUTER NETWORK AND SECURITY

**Signed Declaration
Sheet**

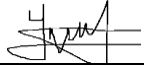
I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____ Signature: _____ Date: _____

Confirmed by advisor:

Name: Yelkal Mulualem (PhD)

Signature:  _____ Date: 23/06/2015 E.C