ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF TECHNOLOGY

ELECTRICAL & COMPUTER ENGINEERING DEPARTMENT

PERFORMANCE ANALYSIS OF DATA ENCRYPTION USING MULTI-SCROLL CHAOTIC ATTRACTORS & A SHARED IMAGE AS A KEY

By

Alem Haddush Fitwi

Advisor

Professor Dr. Sayed Nouh

A thesis submitted to the school of Graduate studies of Addis Ababa University in partial fulfillment of the requirements for the degree of Masters of Science in Computer Engineering

July 2010
Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

PERFORMANCE ANALYSIS OF DATA ENCRYPTION USING MULTI-SCROLL CHAOTIC ATTRACTORS & A SHARED IMAGE AS A KEY

By

Alem Haddush Fitwi

Advisor

Professor Dr. Sayed Nouh

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

PERFORMANCE ANALYSIS OF DATA ENCRYPTION USING MULTI-SCROLL CHAOTIC ATTRACTORS & A SHARED IMAGE AS A KEY

By

Alem Haddush Fitwi

FACULTY OF TECHNOLOGY

APPROVAL BY BOARD OF EXAMINERS

Dr. Getahun Mekuria	_____
Chairman, Dept. of Graduate Committee	**Signature**
Prof. Dr. Sayed Nouh	_____
Advisor	**Signature**
Dr. Eneyew A.	_____
Internal Examiner	**Signature**
Dr. Tamirat B.	_____
External Examiner	**Signature**

Declaration

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

Alem Haddush Fitwi _____
Name signature

Place: Addis Ababa

_____
Date of submission

This thesis has been submitted for examination with my approval as a university advisor.

Prof. Dr. Sayed Nouh _____
Advisor's name signature

ACKNOWLEDGEMENT

First and foremost, I heartily thank The Omnipresent, Omnipotent, and Omniscient God for helping me finish this thesis work in time without which my thesis would be incomplete.

Next, I would like to pour out my heartiest thanks to Prof. Dr. Sayed Nouh first of all for his willingness to be my advisor, and secondly for his continuous constructive comments and guidance.

I also would like to extend my thanks to all of my colleagues and others who have contributed to the successful completion of my thesis work within the scheduled time frame. Special word of thanks goes to the staffs of the Department of Electrical and Computer Engineering, AAU and the Department of Cryptography, INSA for providing me with all invaluable materials and helpful pieces of advice.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

2D	TWO DIMENSIONAL
3DES	TRIPLE DATA ENCRYPTION STANDARD
AES	ADVANCED ENCRYPTION STANDARD
CE	CHAOTIC ENCRYPTION
CIA	CONFIDENTIALITY, INTEGRITY, AVAILABILITY
CPU	CENTRAL PROCESSING UNIT
DES	DATA ENCRYPTION STANDARDS
DFT	DISCRETE FOURIER TRANSFORM
DSA	DIGITAL SIGNATURE ALGORITHM
ECC	ELLIPTICAL CURVE CRYPTOGRAPHY
FIPS	FEDERAL INFORMATION PROCESSING STANDARD
GF	GALOIS FIELD
GUI	GRAPHICAL USER INTERFACE
HMAC	HASHED MESSAGE AUTHENTICATION CODE
IDEA	INTERNATIONAL DATA ENCRYPTION STANDARD
LCG	LINEAR CONGRUENTIAL GENERATOR
LFSR	LINEAR FEEDBACK SHIFT REGISTER
MD	MESSAGE DIGESTS
MLCGs	MIXED LINEAR CONGRENTIAL GENERATORS
NIST	NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
PKC	PUBLIC KEY CRYPTOGRAPHY
PRNG	PSEUDO RANDOM NUMBER GENERATOR
RGB	RED, GREEN, BLUE
RSA	RIVEST, SHAMIR, AND ADLEMAN
S-BOX	SUBSTITUTION BOX
SHA-1	SECURE HASH ALGORITHM 1

ABSTRACT

Most of the secret key encryption algorithms in use today are designed based on either the feistel structure or the substitution-permutation structure, and those public key encryption algorithms are based on the difficulty of factorizing very large numbers and computation of discrete logarithms. There is, however, a growing tendency of exploiting chaotic natures for cryptographic applications. Therefore, this thesis work focuses on data encryption technique using chaotic natures and a publicly shared image as a key, its randomness and performance evaluation.

A key, from which initial conditions of a hysteresis switched second order linear system are calculated, is generated from the shared image using a full period pseudo random multiplicative LCG. Then, multi-scroll chaotic attractors are generated using a hysteresis switched, second order linear chaotic oscillator. During the generation phase, the balance of ones and zeroes was checked using Monobit test, and the range of values of the constant α within which balanced chaos can be generated was determined. The image of the chaotic attractors is processed and sized according to the size of the plaintext. Then, the plaintext is mixed with the chaos during the enciphering process to obtain a ciphertext. The plaintext can be recovered from the ciphertext during the deciphering process only by mixing the cipher with a chaos generated using the same secret key. As validated by randomness tests specified by NIST and German Federal Office of Information Security (BSI), the chaotic algorithm is very much diffused and random which implies that the cipher formed by mixing the plaintext with it is not prone to statistical or selected cipher attacks. That's, it meets the Shannon's characteristics of a good cipher.

What's more, the performance of the chaotic encryption algorithm is measured and analyzed using such metrics as encryption time, encryption throughput, power consumption, CPU time, memory cost, and key security. Besides, it was compared with such existing encryption algorithms as AES and RSA. Notably, this chaotic encryption algorithm is superior to the AES and RSA in that it has variable key length and its encryption time is not significantly affected by the variation in key length. Then, the performance analysis and simulation results verify that the chaotic based data encryption algorithm is valid and applicable.

Chapter One

INTRODUCTION

1.1 Background of the problem

At present, when the Internet provides essential communication between tens of millions of people as illustrated in figure 1.1 and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with. There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. The fast expansion of computer connectivity necessitates good ways of protecting data and messages from unauthorized tampering or reading. Even the US courts have ruled that there exists no legal expectation of privacy for email. It is thus up to the user to ensure that communications which are expected to remain private actually do so. One of the techniques for ensuring privacy of files and communications is Cryptography [1, 2].

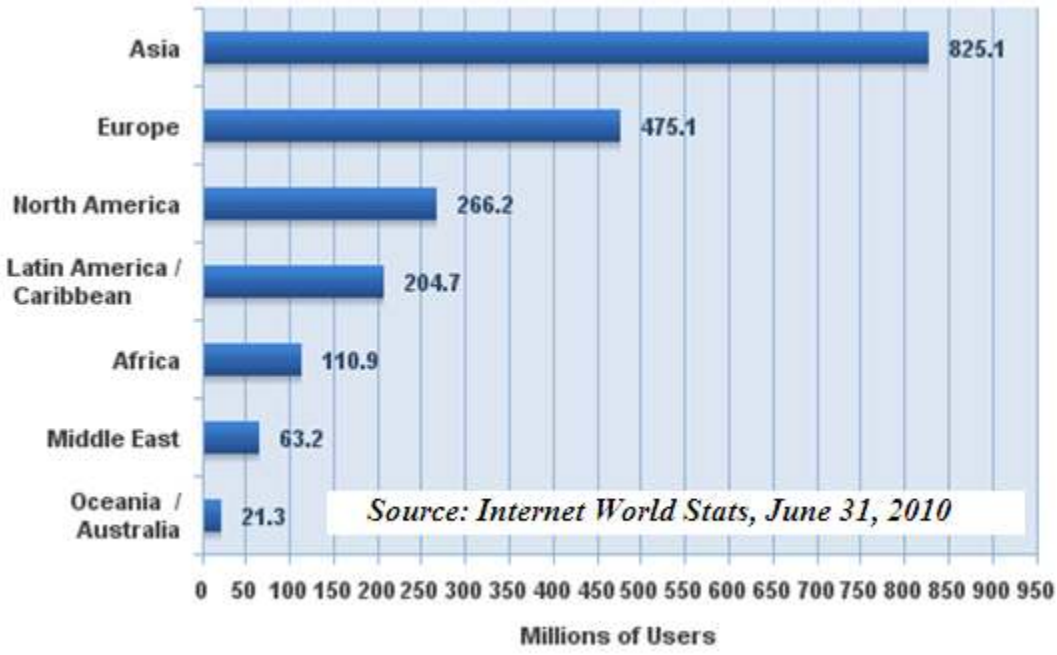


Figure 1.1: World Internet users.

Cryptographic algorithms play an astronomical role in information security systems. It is one of the mechanisms used to achieve security goals, which includes Confidentiality, Integrity, and Availability as depicted in figure 1.2, usually referred to as "CIA". In recent years, as the importance and the value of exchanged data over the Internet or other media types have been increasing alarmingly, there has been a search for the best solution to offer the necessary protection against the data thieves' attacks. On the other side, cryptographic algorithms consume a significant amount of such computing resources as CPU time, memory, and battery power. As a consequence, there has been a great interest of designing cryptographic algorithms which are efficient, both performance and resource utilization wises, and secure [1, 2, 3].

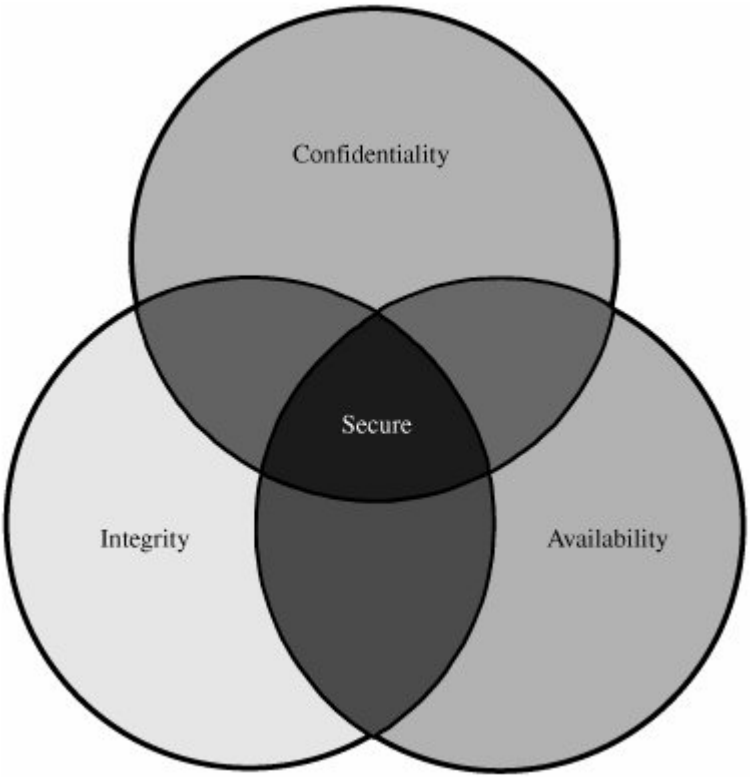


Figure 1.2 : Security goals [2]

1.2 Statement of the problem

At present, secure communication via the internet is a burning issue. There has been a growing need for secure (or reliable), faster, efficient (in resource utilization) and less complex cryptographic algorithms. The currently popular cryptographic algorithms in use are public-key algorithms like Rivest, Shamir, and Adleman (RSA) and Elliptical Curve Cryptography (ECC),

which are based on the computational complexity of "hard" problems, often from number theory, and a secret key algorithm Advanced Encryption standard (AES), which is based on the substitution-permutation network.

However, strong public-key cryptography is often considered to be too computationally expensive for long data encryption, and becomes more extortionate as the key space increases [1, 4]. What's more, the AES, the current in use most popular data encryption algorithm, has fixed input, output and a cipher key (which is multiple of 64). The input and output for the AES algorithm each consists of a sequence of 128 bits. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard [5].

Therefore, there is a ***problem*** of coming up with data encryption model, *based on existing design paradigms*, which is efficient in resource utilization and that satisfies all the security properties (which includes one way encryption, semantic security, and indistinguishability) so as to be resistant to the various attack models. So, to achieve the required level of security and implementation requirements, there is a demand to follow different design approaches.

1.3 Objective

1.3.1 General Objectives

The main goal of this thesis work is to design multi-scroll chaotic cryptographic system for data enciphering which has variable key length, and a key-length independent computational time, and eventually evaluate its performance.

1.3.2 Specific Objectives

The detailed and specific objectives of this thesis work are outlined as follows:

- To generate suitably a key from a shared image using pseudo random number generator called multiplicative linear congruential generator (LCG).

- To generate chaotic attractors using a hysteresis switched linear second order system and optimize them for data encryption
- To devise an XOR mixing technique for data encryption and decryption.
- To identify metrics for performance analysis.
- To perform functional and randomness tests on the chaotic algorithm.
- To experiment and evaluate the performance (for the applicable metrics only) of the design by implementing it on java or matlab as convenient.

1.4 Methodology

The methodology used in doing this thesis comprises four major phases. The first phase encompasses the study of the issues and areas closely pertinent to the thesis work in order to acquire an in-depth understanding and knowledge of the relevant areas where the problem lies. The other three phases elucidate how such major tasks as designing, testing, and performance analysis are carried out. The major phases and the activities performed in each one of them are pictorially portrayed and explicated below in figure 1.3, and they were performed in the order they appear in the figure.

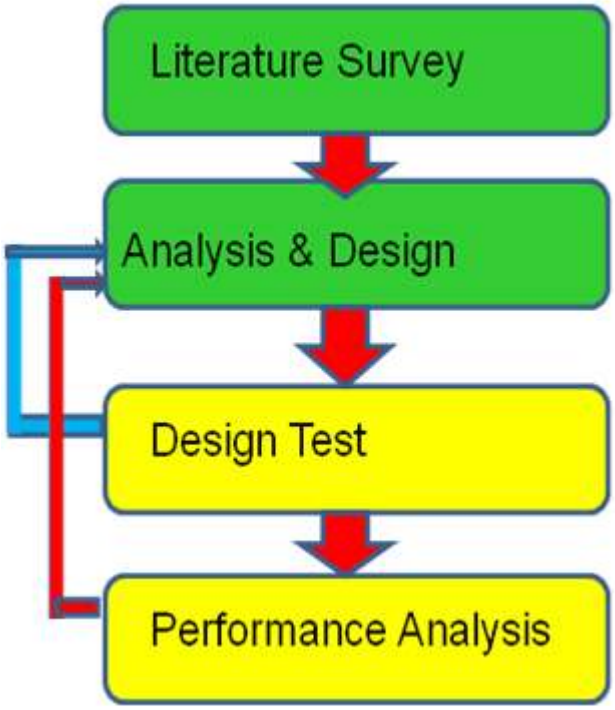


Figure 1.3: Summary of the Methodology

- ✓ **Literature survey:** This very first phase focuses on gaining all the necessary background information required to understand the broader picture of the problem as well as its vicinities. This is accomplished by reading as many literatures as possible in such relevant areas as image processing, generation of chaotic attractors, and various cryptographic algorithms which includes articles, books, research papers, class lecture notes, research publications and information available on the Internet.
- ✓ **Analysis and Design :** This phase involves the study of various cryptographic design paradigms. Various pertinent propositions and their drawbacks available so far, and development of a concept to alleviate some of the problems posed on the security of cryptographic algorithms are studied and analyzed in order to come up with a new perception and design of the problem.
- ✓ **Testing the design:** The third phase is all about the various standard tests and simulations performed on a given cryptographic algorithm. A number of standard tests such as NIST, BSI, and the Canadian standard security tests, and common simulation techniques are studied, and analyzed in order to customize them as to suit the designed chaotic cryptographic algorithm to validate its functionality and security.
- ✓ **Performance Analysis:** At last, a deep study on how metrics are identified and the performance of the designed system is evaluated and analyzed is made. That's, a number of approaches for performance measurement metrics identification of the designed chaotic encryption algorithm are studied so as to select the right approach. Besides, this phase involves data collection and performance evaluation of the designed algorithm and others.

1.5 Organization of the Thesis

The thesis is organized as follows. Chapter two presents all the necessary background information needed to become familiar and conversant with the rudimentary and core concepts of various cryptographic algorithms and chaos generation. Chapter three presents how the cryptographic model is designed which includes key generation, chaos generation, enciphering process, and deciphering process. Chapter four is all about the functional and randomness tests performed on the chaotic generator. Identified metrics, performance measurements of the design

and corresponding results are presented in chapter five. Then, eventually, conclusion, and application of the results are presented in chapter six.

Chapter Two

LITERATURE REVIEW

2.1 Introduction to Cryptography

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. It has two major branches of studies, namely Cryptography (code making) and Cryptanalysis (code breaking). Cryptography, the concern of this thesis work, is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet [1].

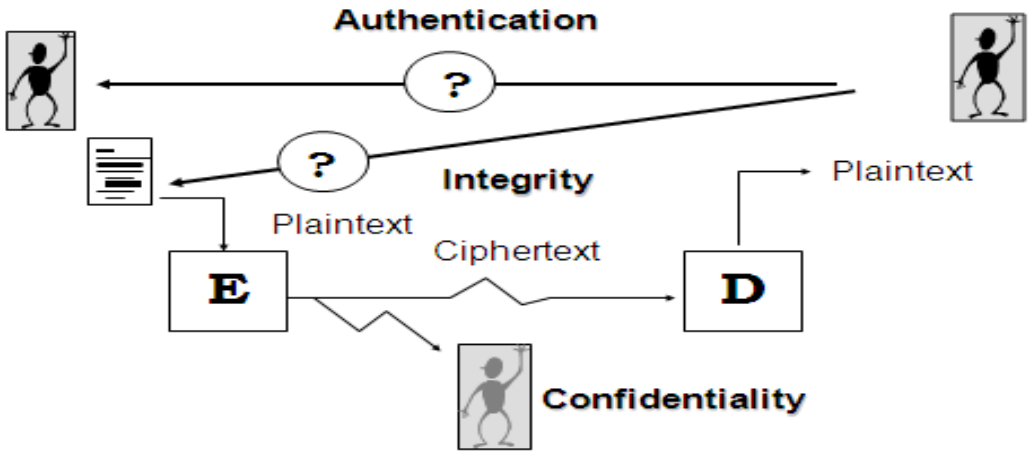


Figure 2.1: Security Issues [6]

Within the context of any application-to-application communication, as depicted in figure 2.1 there are some specific security requirements that cryptography must provide, including [1,2,7,8]:

- *Authentication:* The process of proving one's identity.
- *Privacy/confidentiality:* Ensuring that no one can read the message except the intended receiver.
- *Integrity:* Assuring the receiver that the received message has not been altered in any way from the original.

Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described below. In all cases, the initial unencrypted data is referred to as *plaintext*. It is encrypted into *ciphertext*, which will in turn usually be decrypted into usable plaintext [1, 2, 7]. Generally, security attacks are of two categories namely passive and active attacks. Cryptographic algorithms are designed to usually prevent passive attacks on information.

2.1.1 Secret Key Cryptography

It is a type of cryptography where a single key is used for both encryption and decryption processes. As portrayed in figure 2.2, the sender uses the key to encrypt the plaintext and sends the cipher-text to the receiver.

The receiver applies the same key to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called *symmetric encryption*. With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key. Secret key cryptography algorithms include: DES, 3DES, AES, IDEA, etc [1].

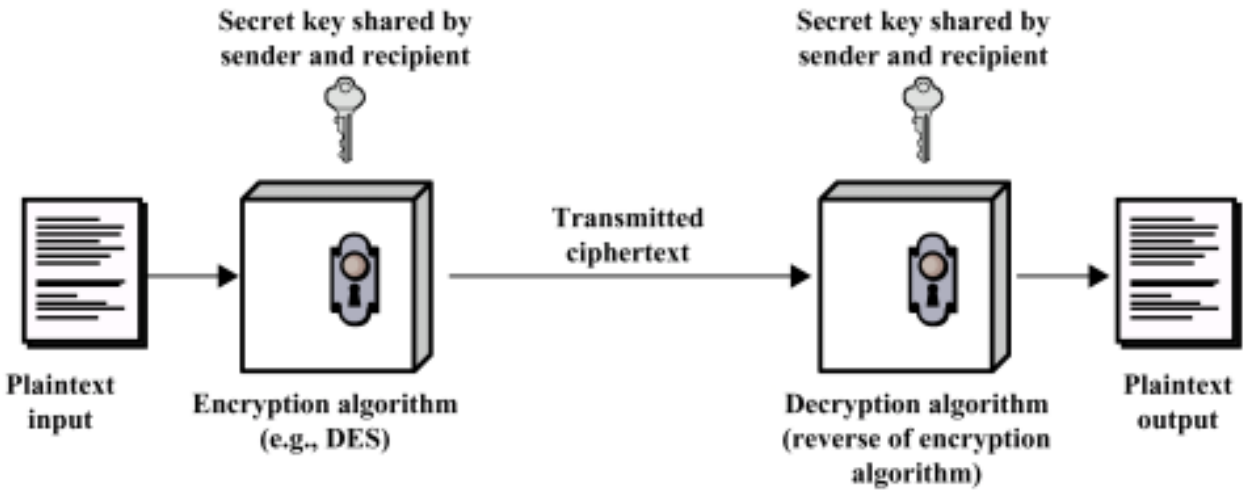


Figure 2.2**:** Secret Key Cryptography [7].

2.1.2 Public-Key Cryptography (PKC)

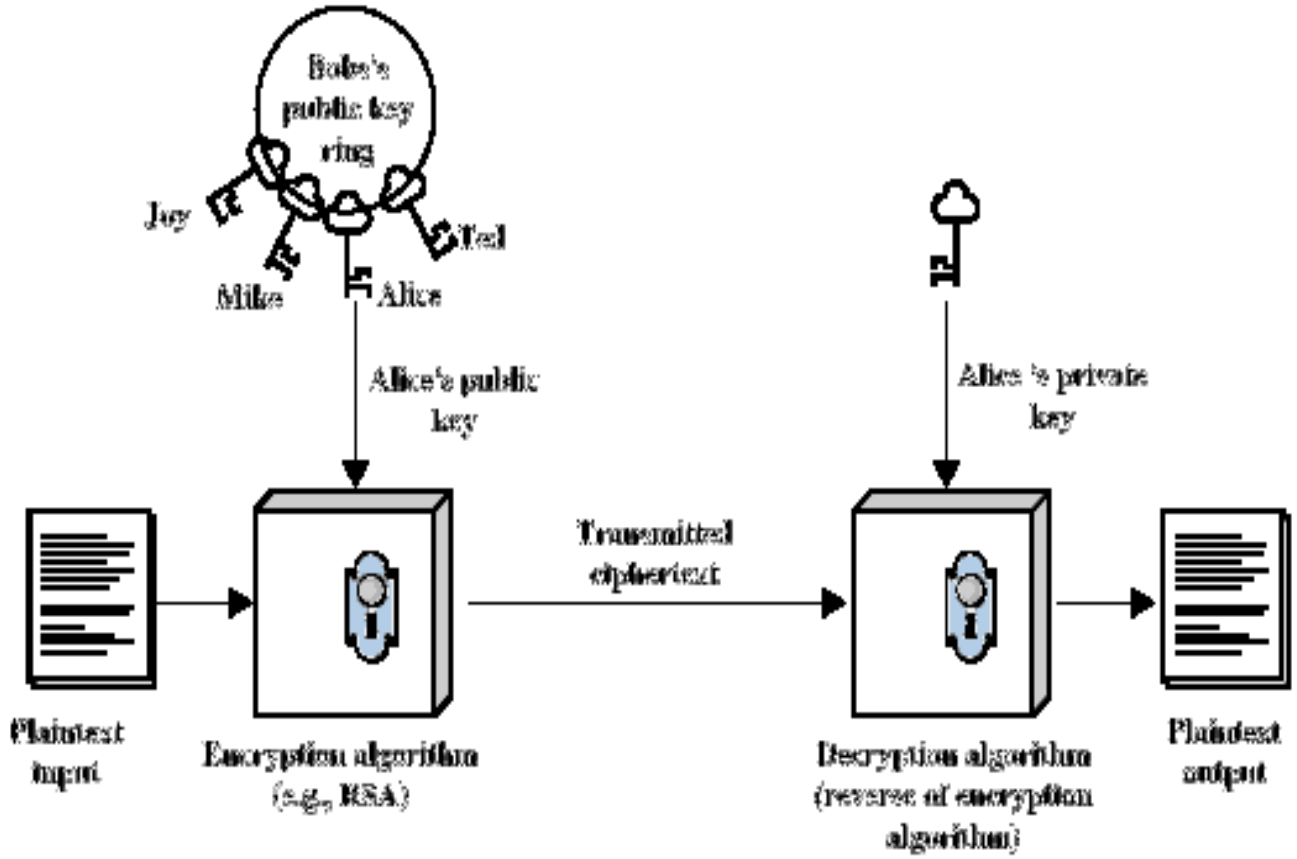


Figure 2.3: Public Key Cryptography [7].

Public-key cryptography, that uses two mathematically related asymmetrical keys (one key for encryption and the other for decryption) as depicted in figure 2.4, has been said to be the most significant new development in cryptography in the last four decades.

Modern PKC was first described publicly by Stanford University professor Martin Hellman and a graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key [1,2,7,8,9]. Public-key cryptography algorithms that are in use today for key exchange or digital signatures include: RSA, Deffie-Hellman, DSA, EIGamal, ECC, etc [1].

2.1.3 Hash Functions

Hash functions, also called *message digests* and *one-way encryption*, are algorithms that, in some sense, use no key. But, as depicted in figure 2.4, a fixed-length hash value is computed based upon the plaintext and a secret value or key attached to it. During transmission only the message concatenated with the hash is transmitted. The secret value is removed just after the required hash value is computed! This makes it impossible for either the contents or length of the plaintext to be recovered. It is a one way cryptographic algorithm in that the original message can't be recovered using a key from the hashed value.

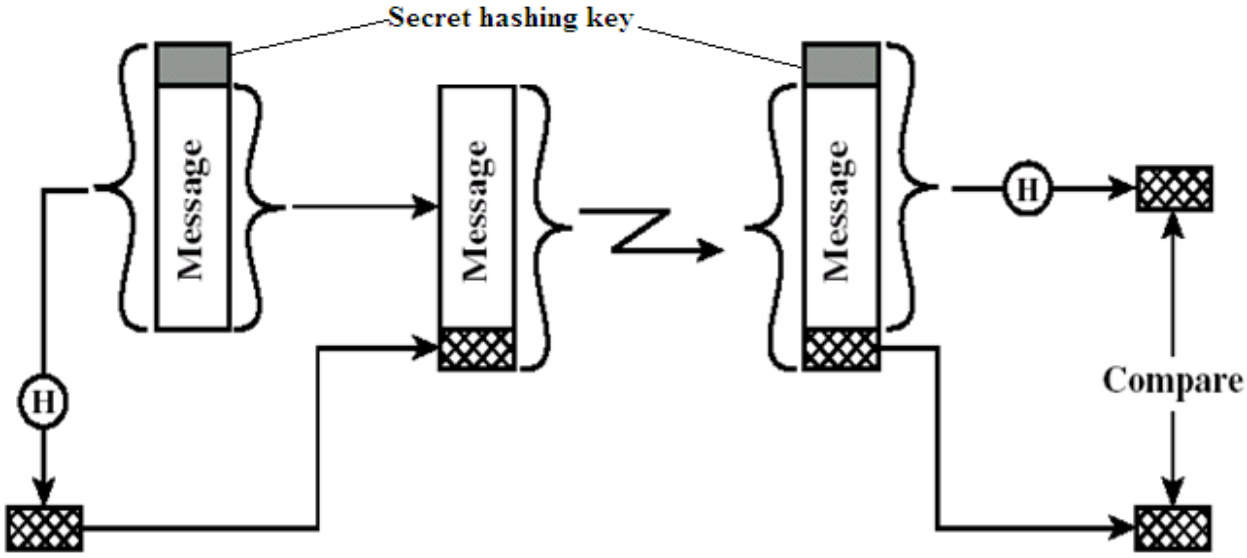


Figure 2.4: Hash Function (One Way Cryptography) [6].

The secret value is added just to increase the security of the hash. Hash algorithms are typically used to provide a *digital fingerprint* of a file's contents often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file [1, 7].

2.2 Data Encryption Standard, DES

The Data Encryption Standard (DES) is a block cipher (a form of shared secret encryption) that was selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) for the United States in 1976 and which has subsequently enjoyed widespread use internationally. It is based on a symmetric-key algorithm that uses a 56-bit key and a feistel structure. DES is now considered to be insecure for many applications. This is *chiefly due to the 56-bit key size being too small*. In January, 1999, Distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes. Consequently, DES has been withdrawn as a standard by the National Institute of Standards and Technology (formerly the National Bureau of Standards) and was finally superseded by the Advanced Encryption Standard (AES) on 26 May 2002 [1, 10, 11, 12, 13,14, 15, 16].

2.3 Advanced Encryption Standard, AES

The Advanced Encryption Standard (AES) is a substitution-permutation structure based encryption standard adopted by the U.S. government. It was announced by National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001 after a 5-year standardization process. The AES ciphers have been analyzed extensively and are now used worldwide, as was the case with its predecessor, DES. Until May 2009, the successful published attacks against the full AES were side-channel attacks on some specific implementations and Monte Carlo simulation on the first four rounds. The input and output for the AES algorithm each consist of sequences of 128 bits. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard [1, 5, 8, 10, 11, 17].

2.4 Rivest, Shamir and Adleman (RSA) Algorithm

RSA, which stands for Rivest, Shamir and Adleman who first publicly described it, is an algorithm for public-key cryptography. It is believed to be secure given sufficiently long keys and the use of up-to-date implementations. As of 2010, the largest (known) number factored by a general-purpose factoring algorithm was 768 bits long, using a state-of-the-art distributed implementation. RSA keys are typically 1024–2048 bits long. Some experts believe that 1024-bit keys may become breakable in the near term; few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if n is sufficiently large. If n is 300 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. To make it more secure, the key size can be made longer; however, as the key size increases, it becomes more expensive computationally [4, 9, 18].

2.5 Elliptical Curve Cryptography, ECC

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. An ECC with a key-length greater than 112-bit said to be secure but slow when used for data encryption. As the key size increases, encryption using ECC becomes computationally more expensive [4, 9, 19, 20].

2.6 Multi-Scroll Chaotic Attractors

“Chaos" means "a state of disorder", but the adjective "chaotic" is defined more precisely in chaos theory. For a dynamical system to be classified as chaotic, it must be sensitive to initial conditions, and topologically mixing (means the system should evolve over time so that any given region of its phase space will eventually overlap with any other given region so as to be random and unpredictable). An ***attractor*** is a set towards which a dynamic system evolves over time. That is, points that get close enough to the attractor remain close even if slightly disturbed. Geometrically, an attractor can be a point, a curve, or even a complicated set with a fractal structure known as a *strange attractor* [21]. Figures 2.5 and 2.6 depict examples of chaos

generated by uncontrolled (Turbulence in the tip vortex from an airplane wing) and controlled (The ***Lorentz attractor,*** the best-known chaotic attractor) systems, respectively.

Figure 2.5: Turbulence in the tip vortex from an airplane wing [21].

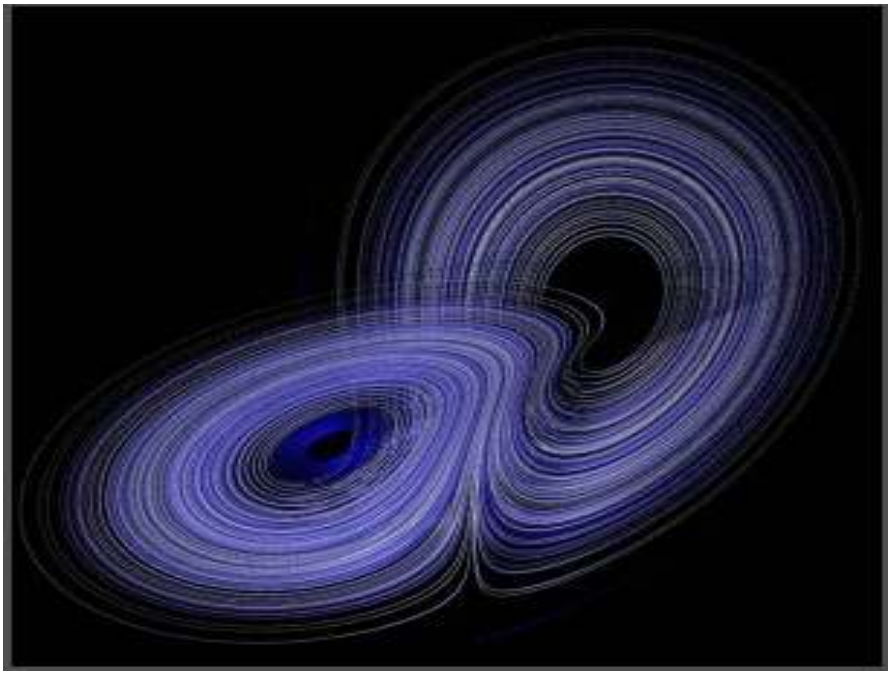


Figure 2.6: The *Lorentz attractors* [21].

Over the last two decades, chaotic oscillators have been found to be useful with great potential in many technological disciplines such as information and computer sciences, biomedical engineering, power systems protection, encryption and communications, etc.

Recently, there has been some increasing interest in exploiting chaotic dynamics for real-world engineering applications, in which much attention has been focused on effectively generating chaos from simple systems by using simple controllers. Then a survey has been made on a number of techniques which have been developed for generating chaotic attractors and their application (related works) in papers [22, 23, 24, 25, 26, 27, 28, 29, 30]. The related works explored in this work includes biometric encryption using multi-scroll attractors and image encryption using chaotic maps. They, however, don't discuss how secure the key is, how the key is exchanged, and how good it is as compared with existing cryptographic algorithms. What's more, the related works done so far don't define the secure range of constant values, and the security of the cipher is not tested well. That's, the chaotic generators are not tested against standard cryptographic randomness tests. For instance, paper [24] proposes fingerprint image encryption using chaos. However, it doesn't achieve the security properties that a cryptographic algorithm should satisfy. Besides, the key generation and exchange is insecure and not different from existing schemes. It is also difficult to realize it in practical implementations unless some suitable techniques for the extraction of finger features are used.

Then, eventually, after surveying as many papers as possible about the efficient generation of chaotic attractors, I have selected "*Hysteresis Switched Linear Second Order System*" for chaos generation in my designed system. This chaotic oscillator is chosen because it can create chaos easily using a second order linear system; it is easy to implement with small circuitries; and it is efficient to generate n-scroll chaotic attractors [22, 23, 25, 26, 27, 28, 29, 30].

The block diagram portrayed in figure 2.7 represents a linear second-order system with a feedback of hysteresis series [30].

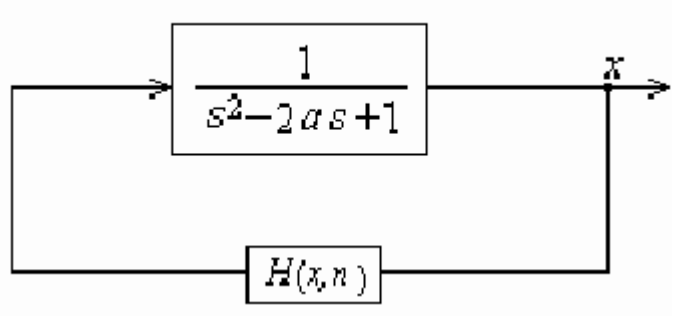


Figure 2.7: Second Order Chaotic Oscillator [30].

The hysteresis switched second order system depicted in figure 2.7 can be described by

$$\begin{cases} \dot{x} = y \\ \dot{y} = -x + 2\alpha y + H(x,n) \end{cases} \quad \text{...(2.1)}$$

where x and y are state variables, α is a positive constant, H(x, n) is a hysteresis-series described by

$$H(x,n) = \sum_{i=1}^{n} h_i(x) \quad \text{...(2.2)}$$

and

$$h_i(x) = \begin{cases} 1 & for\ x > i-1 \\ 0 & for\ x < i \end{cases} \quad \text{...(2.3)}$$

The solution of system (2.1), assuming the solution to be complex, can be expressed as

$$\begin{cases} X(t) = e^{\alpha t}[X(0)\cos(\beta t) + \frac{1}{\beta}(Y(0) - \alpha X(0))\sin(\beta t)] \\ Y(t) = e^{\alpha t}[Y(0)\cos(\beta t) + \frac{\alpha}{\beta}(Y(0) - \alpha X(0) - \frac{\beta^2}{\alpha}X(0))\sin(\beta t)] \end{cases} \quad \text{...(2.4)}$$

Where $\beta=\sqrt{1-\alpha^2}$, n is number of scrolls, and X(0) & Y(0) are the initial conditions. With suitable parameters, system (2.1) can create (n+1)-scroll chaotic attractors from the system of solutions in 2.4 [30]. System (2.1) is fully described and solved in chapter three of this thesis report.

Chapter Three

DEISGN AND ANALYSIS OF THE CHAOTIC ENCRYPTION SYSTEM

3.1 Design overview

The design of a chaotic based crypto-system comprises five major tasks as delineated in figure 3.1. The tasks include image processing, key generation, generation of chaotic attractors, enciphering process, and deciphering process. What's more, the type of techniques used to manage the secret key of the designed chaotic crypto-system, and to provide a digital fingerprint of the shared image to check its integrity are presented.

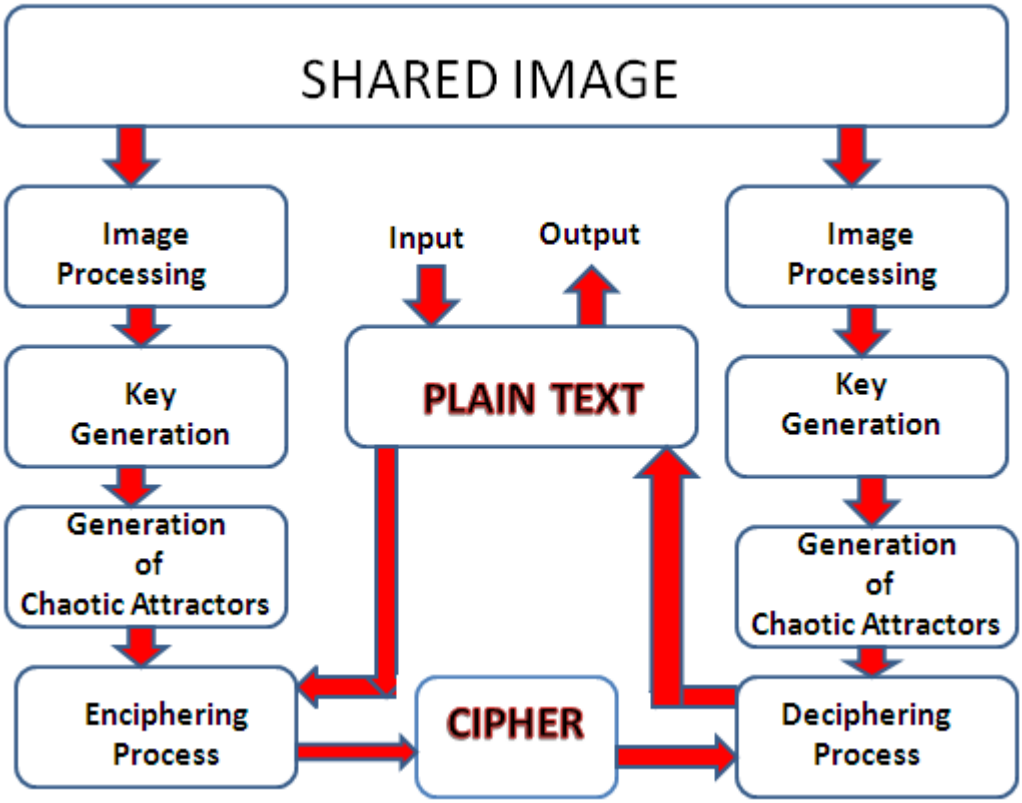


Figure 3.1: Chaotic Crypto-System

3.2 Shared Image

In this crypto-system, the same RGB image, in lieu of the secret key itself, is publicly shared amongst all communicating (sending and receiving) parties from which the secret key is extracted. It is publicly shared by communicating parties just like a public key of a public-key encryption, only the information required to extract the key from the image is communicated secretly. This gives a better cover-time to the message or information being communicated secretly as compared to the scheme of communicating the secret key itself. Keys having lengths less than the image size (width*length) are extracted from this shared image. The shared image used in this thesis work and from which a secret key is extracted is the one portrayed in the figure 3.2. The minimum key length allowed in this chaotic algorithm (designed in this thesis work) is 128 bits for it is the minimum strong secure key length as explicated in section 3.3.1. Above it, it can be of any length as long as it is less than or equal to the size of the shared image. However, whenever there is a need for key length greater than the shared image size, the image can be resized. That's, its size can be made larger as needed.

Figure 3.2: Shared image from which a secret key is extracted (261x326)

The shared image is then processed to make it convenient to extract the secret key from its pixel values. The image processing here comprises such processes as ***image reading***, ***converting to grayscale***, and ***grabbing the pixel values*** of the grayscale image.

Figure 3.3: RGB Image read in from a file.

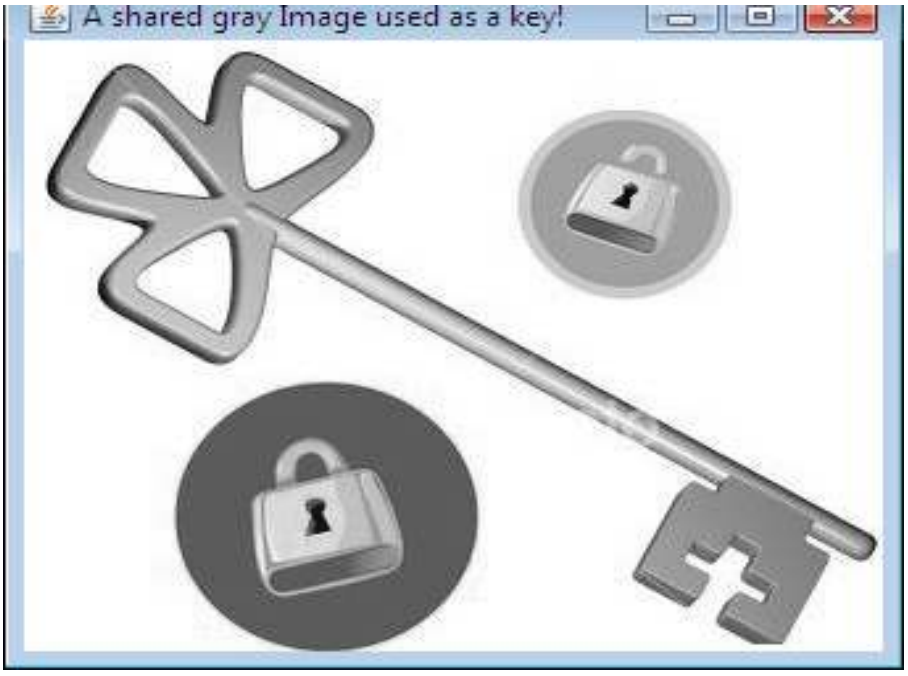


Figure 3.4: Grayscale Image.

The RGB image in figure 3.3 is read in using the method ***readRGBImage(),*** and its important attributes, particularly its width and height, are accessed as follows:

$$Image\ width,\ w=image.getWidth() \quad \text{---(3.1)}$$

$$Image\ height,\ h=image.getHeight() \quad \text{---(3.2)}$$

At last, the RGB image is converted to a grayscale, portrayed in figure 3.4, using the method ***convertTogray()*** from which pixel values, ranging from 0 to 255, are grabbed into a two dimensional array.

$$ImagePixel[w][h]= readGrayImagePixels(grayImage) \quad \text{---(3.3)}$$

3.3 Key Generation

Any secret key of length less than the size of the shared image can be extracted from the two dimensional pixel values of the shared image stored in the 2D array, ImagePixel[w][h].

$$Key.length <= ImagePixel.length,\ or$$

$$Key.\ length <= w*h \quad \text{…(3.4)}$$

Where the values of w, and h are obtained in equations 3.1 and 3.2 respectively, and ImagePixel in equation 3.3.

In this thesis work, the key is extracted from the 2D pixel values of the grayscale image using a full period pseudo random number generator (PRNG). The PRNG used here is called linear congruential generator, LCG, constructed using values defined in GF (m) with a period of m-1. Then, the extracted key, keyExtract, is converted to binary values, and finally substituted using a seven-bit input and five-bit output S-boxes (substitution boxes) to obtain the final enciphering key, keyFinal.

3.3.1 Deciding the size of the Secret Key

The security of this chaotic cryptographic algorithm depends on the secrecy of the enciphering and deciphering key. One potential attack, if the algorithm is made known, is to try all possible deciphering keys and to eliminate all incorrect ones. This is often referred to as a "Brute-force attack", or "Exhaustive key search attack". To withstand this type of attack a large key space is required. Considering a k-bit key, all the 2^k keys are tried one-at-a-time on the ciphertext c until

it decrypts to the known plaintext *m*. Practical security of a "good cipher" depends on the size of the key space: *2^{40} operations is easy, 2^{64} is on the border of practicality, 2^{80} is not feasible and 2^{128} is very strong indeed!* [6, 9]. Hence, exhaustive cryptanalysis with a single decrypter chip is usually infeasible. As a result, the attacking strategy comprises a trade-off between hardware and computing time. For example, decrypting simultaneously with 10^6 DES chips would require about 10^{11} decryptions per chip for exhaustive cryptanalysis. Currently, there exist chips that can do more than 10^6 decryptions/s. Thus, exhaustive cryptanalysis of DES was done in 10^5 sec. ≈ 28 hours [1,10,13]. There are about 3.16×10^7 seconds in a year (about 2^{24} seconds). Below is shown the approximate key size needed to withstand an exhaustive key search for *one year:*

Type of attack	Key size
A human effort at 1 key/s	25 bits
A processor at 10^6 keys/s	45 bits
1000 such processors	55 bits
10^6 processors/chips at 10^6/s	65 bits

Table 3.1: Time to break a given key size

Key Size (bits)	Number of Keys	Time Required at 1 Decryption/ μs	Time Required at 10^6 Decryption/μs	Time Required at 3.6×10^{51} Dec/μs
32	$2^{32} = 4.0 \times 10^9$	2^{31} μs = 35.8 minutes	2.15 ms= 6.8×10^{-11} yrs	6×10^{-37} ps
56	$2^{56} = 6.3 \times 10^{16}$	2^{55} μs = 1000 years	10 hrs=1.14×10^{-3} yrs	4.5×10^{-30} ps
64	$2^{64} = 1.6 \times 10^{19}$	2^{63} μs = 2.5×10^5 years	0.25 years	1.2×10^{-27} ps
80	$2^{80} = 1.0 \times 10^{24}$	2^{79} μs = 1.6×10^{10} years	16,000 years	7.5×10^{-23} ps
128	$2^{128} = 2.5 \times 10^{38}$	2^{127} μs = 4.0×10^{24} years	4.0×10^{18} years	1.8×10^{-8} ps
168	$2^{168} = 2.5 \times 10^{50}$	2^{167} μs = 4.0×10^{36} years	4.0×10^{30} years	19 ns
256	$2^{256} = 6.3 \times 10^{76}$	2^{255} μs = 3.5×10^{63} years	3.5×10^{57} years	9.7×10^7 years
512	$2^{512} = 4.0 \times 10^{153}$	2^{511} μs = 2.0×10^{140} years	2.0×10^{134} years	1.1×10^{88} years
1024	$2^{1024} = 1.6 \times 10^{307}$	2^{1023} μs = 5.4×10^{294} years	5.4×10^{288} years	1.5×10^{243} years

Table 3.1 shows various key sizes and the time needed to break each of them by exhaustive key search attack or brute force analysis using decryption chips of different capabilities. Although, currently 10^{14} operations is regarded as a limit for computational feasibility [4], the key size used in this chaotic algorithm should be 128 bits or greater in order for the algorithm to be strongly secure.

3.3.2 Generation of a Secret Key using a full period LCG

LCG was discovered by D.H. Lehmer. He discovered that the residues of successive powers of a number have good randomness properties. Lehmer obtained that the n^{th} number in the sequence by dividing the n^{th} power of an integer *a* by another integer *m*, and taking the remainder [31]. That's,

$$X_n = a^n \bmod m \qquad \text{---(3.5)}$$

The expression used to compute X_n after computing X_{n-1} is therefore given below.

$$X_n = a^n X_{n-1} \bmod m \qquad \text{---(3.6)}$$

The parameters *a* and *m* are called multiplier and modulus, respectively. Lehmer's choice for these parameters were $a = 23$ and $m = (10^8 + 1)$ [31].

Many of the current in-use random-number generators are a generalization of the Lehmer's proposal and have the following form [31]:

$$X_n = (aX_{n-1} + b) \bmod m \qquad \text{---(3.7)}$$

Here, the X_n's are integers between 0 and m-1. Constants *a* and *b* are non-negative. These types of generators are called Mixed Linear Congruential Generators (MLCGs) or simply Linear Congruential Generators (LCGs). The word mixed implies the use of both multiplication (by *a*) and addition (with *b*) operations.

In this thesis work, a Multiplicative LCG, where the value of *b* in equation 3.7 is zero, is used. The Multiplicative LCG is modeled by equation 3.8. It is more efficient than mixed LCGs in terms of processor times required for computation [31].

$$X_n = (aX_{n-1}) \bmod m \qquad \text{---(3.8)}$$

In the case of the multiplicative LCG, with a proper choice of the multiplier *a*, it is possible to obtain a period of (*m-1*), which is almost equal to the maximum possible length m. Besides,

unlike to a mixed LCG, X_n obtained from multiplicative LCG can never be zero if m is a prime. The values of X_n lie between 1 and m-1, and any multiplicative LCG with a period of ***m-1*** is called a ***full period generator***. All values of the multiplier are not equally good. A multiplicative LCG will be a full period generator if and only if the multiplier *a* is a primitive root of the modulus m. By definition, *a* is said to be a primitive root of m if and only if $a^n \bmod m \neq 1$ for n=1, 2, 3, …, *m-2* where *m* is usually selected to be prime and its size is decided based on the size of the sequence to be generated [31].

In this work, I have used a full period multiplicative LCG where the multiplier $a= 69,621$, and the modulus is $m= (2^{31}-1)$. Its period is $m-1=2^{31}-2$. It is delineated in equation 3.9 below.

$$X_n= (69,621X_{n-1}) \bmod (2^{31}-1) \qquad \text{---(3.9)}$$

Where X_o is called the ***seed*** used to initialize the random-number generator. The seed can assume any integer value greater than zero. This generator is used in this work due to the fact that it is one of the best two LCGs in terms of implementation efficiency and randomness as studied by Fishman and Moore [31].

For different values of the seed, X_o, different sequences containing 2^{31} -2 random integer values ranging from 1 to 2^{31}-2 can be generated. In cases where the value of the seed, X_o, is very large, it may cause overflow when multiplied with $a=69,621$. Therefore, the random integer values are generated using Schrage law described in [31]. Hereunder is given the algorithm for the selected generator to generate 17 random numbers between 1 and 2^{31}-2 inclusive.

Table 3.2: Algorithm for PRNG [31].

```
Const
     a=69,621;
     m=2^31-1=2147483647;
     q=30845;
     r=23902;
     Xo=1;
Var
     Xmq, X, Xnew, Xdq: long;
Begin
     X:= Xo;
     for i:=1 to 17 Do
          Begin
             Xdq:=X/q;
             Xmq:=X%q;
```

```
            Xnew:=a *Xmq -r* Xmq ;
            if(Xnew >0)
                  X:= Xnew;
            else
                  X:= Xnew+m;
            end;
            Random:=X;
         end;
end;
```

Two sequences of 17 random numbers generated using the above algorithm for Xo=1, and Xo=10000, respectively, are listed below:

X_{nw}=[69621 552116347 1082396834 201323037 1832878655 1219051368 874078441 971035822 1699755902 1619285207 1953863635 1883480414 143449980 1332099030 837788288 2002546328 344571154];

X_{nh}=[696210000 2130497210 650759120 1038192761 13622855 1396499628 525966710 1584651913 225953995 825371620 830129594 1372555810 2128207451 149237659 549173053 178271725 1147770212];

Table 3.3: Algorithm for key extraction

```
Const
     n=17; //key.length=17*8=136 bits
     w=261; // defined in equation 3.1
     h=326; // defined in equation 3.2
Var
     keyExtract: long;
     wIndex, hIndex:int;
Begin
     Index:=1;
     for i:=1 to n Do
         Begin
              w-Index:= Xnw [i]%w;
              h-Index:= Xnh [i]%h;
              keyExtract[i] =ImagePixel[wIndex][ hIndex];
         end;
  end;
```

The random numbers generated using the algorithm in table 3.2 are used as ***indices*** to extract a key from the 2D pixel values of the grayscale image depicted in figure 3.4. The algorithm used

to extract 136-bit key, *keyExtract*, from the 2D array of pixel values stored in ImagePixel[w][h] is illustrated in table 3.3.

The 136-bit extracted key, *keyExtract*, is therefore given below represented in decimal (base 10) numeration system. It is made of seventeen pixel values (each one byte long) extracted from the grayscale image.

$$keyExtract = 141721431491931106276051 1 \quad \ldots(3.10)$$

Then, the extracted key is converted to binary values for convenience using the *convertToBinary()* method as follows:

$$keyBinary[len] = convertToBinary(keyExtract) \quad \ldots(3.11)$$

where len is the key length in bits (or number of integer pixels extracted from the shared image). In this case len, the length of the binary key, is 136 bits.

3.3.3 Key Substitution using S-Boxes

After dividing the binary key, *keyBinary[136]*, into blocks of size 49 bit each, each block is divided into seven 7-bit pieces before they are processed by the *S-boxes*, or *substitution boxes*. Each of the seven S-boxes replaces its seven input bits with five output bits according to a non-linear transformation, provided in the form of a lookup table. The S-boxes strengthen the security of the key; that's, substituted bits are used instead of the actual bits randomly extracted from the shared image thereby increasing the efforts of cryptanalysts who try to infer the key using brute force analysis or selected cipher attack. The key is first padded with 0's to make its length multiple of 49, the block size.

The S-Boxes in this algorithm serve more or less the same purpose as the S-Boxes used in DES and AES; they are however different from those used in DES and AES. Here seven S-Boxes are used. Each of them is constructed using defined transformation of values in GF (2^5) comprising 4 unique rows and 32 columns. Each raw comprises 32 elements starting from 0 to 31 in a thoroughly random sequence as illustrated below.

Table 3.4: S-Boxes/Substitution Boxes

```
S1=[14 21 7 28 18 16 27 1 3 17 8 13 25 4 5 22 30 15 19 24 31 6 20 11 2 0 29
    10 12 9 23 26;
    24 16 18 0 13 11 19 27 5 12 20 1 7 26 4 8 30 15 21 28 23 31 9 17 29 10 2
    22 6 25 3 14;
    28 19 23 1 25 24 27 21 30 18 15 26 20 13 2 14 29 0 7 5 8 16 17 9 31 11 4
    6 10 12 3 22;
    6 15 18 3 0 27 23 11 12 25 20 4 31 5 9 7 14 1 24 17 21 30 16 10 8 2 13 29
    19 26 28 22];

S2=[26 2 31 8 22 15 25 9 18 12 29 24 21 3 20 11 14 0 13 27 30 19 16 4 17 6 1
    10 28 7 5 23;
    1 26 3 0 13 4 25 6 14 18 15 24 21 9 12 19 30 5 11 27 28 20 23 22 2 7 29
    16 10 31 8 17;
    14 22 2 10 4 16 8 21 18 27 19 17 28 15 11 9 20 24 29 0 6 23 12 1 3 13 7
    26 31 5 30 25;
    15 1 23 11 21 25 30 12 8 18 4 3 9 29 22 14 0 5 24 19 26 13 16 28 10 31 20
    7 6 27 2 17];


S3=[13 11 27 10 16 12 30 5 29 17 31 8 24 3 6 4 0 7 20 2 26 22 15 23 19 21 25
    28 18 14 1 9;
    10 4 21 20 31 1 18 3 29 28 12 19 5 15 22 14 17 7 30 24 13 11 2 0 26 9 23
    25 8 16 27 6;
    10 20 21 23 24 26 1 31 27 17 2 0 16 25 22 29 9 14 11 3 19 5 28 6 7 30 15
    13 4 18 8 12;
    2 16 21 17 27 7 15 22 28 8 18 23 24 25 12 10 26 14 0 9 31 19 3 29 20 6 5
    4 1 11 13 30];

S4=[8 24 6 23 26 28 1 0 5 4 11 15 18 19 31 7 29 17 2 21 14 10 30 20 16 27 13
    25 12 22 3 9;
    28 25 2 29 1 12 27 17 3 7 11 6 5 26 13 31 14 21 0 23 9 24 19 10 16 20 18
    15 22 4 8 30;
    5 10 31 28 26 25 13 0 18 19 23 22 3 30 4 16 7 8 15 6 2 27 24 1 21 11 29
    20 9 12 17 14;
    22 31 2 21 27 23 26 0 24 3 25 28 4 7 18 29 14 9 11 5 17 13 8 12 19 15 1
    30 16 20 6 10];

S5=[18 16 9 23 10 19 8 28 14 6 17 13 2 1 12 25 30 29 24 31 7 5 27 3 15 11 0
    21 22 26 20 4;
    17 23 28 15 0 11 21 25 19 30 7 13 22 27 18 24 16 31 8 12 10 6 3 5 1 29 9
    2 4 20 14 26;
    11 24 20 9 28 6 7 23 19 21 1 18 27 0 30 26 2 22 10 31 8 13 29 3 25 12 14
    17 15 5 16 4;
    10 28 0 21 23 18 7 27 12 14 8 11 29 26 22 4 31 1 3 6 2 19 9 17 20 25 15 5
    30 24 16 13];

S6=[20 2 11 29 30 18 7 22 1 13 9 23 25 6 5 14 15 28 27 17 8 4 31 24 26 12 21
    0 10 3 19 16;
    21 15 14 0 8 13 4 9 1 23 12 25 20 22 17 16 28 10 30 29 24 19 6 7 26 27 5
    2 31 3 11 18;
    5 28 10 6 15 24 30 7 3 12 0 29 19 25 13 1 22 8 26 27 14 23 2 18 4 31 9 11
    17 16 20 21;
    25 4 8 15 23 14 7 5 24 18 1 17 27 26 31 22 6 21 30 2 16 3 10 13 9 29 19
```

```
    12 11 20 0 28];

S7=[13 28 7 4 24 19 3 2 31 14 29 0 23 1 9 11 20 17 15 16 22 6 8 21 25 18 26
    12 10 5 27 30;
    25 24 1 2 30 8 20 27 11 23 6 12 0 28 19 26 15 3 5 21 9 17 31 16 18 7 29
    13 10 14 4 22;
    11 10 4 24 20 5 18 30 17 8 23 15 25 27 19 31 2 7 3 14 28 12 9 16 1 0 21 6
    22 13 26 29;
    21 28 5 24 12 31 2 3 11 17 30 7 0 10 14 1 4 23 16 26 29 27 19 22 18 13 20
    6 9 8 15 25];
```

The input bits are used as addresses in tables called **S-boxes**. Each group of seven bits will give us an address in a different **S**-box. The first and last bits of the 7-bit input indicate row number, and the other 5 bits gives you the number of column. Located at that address will be a 5-bit number. This 5-bit number will replace the original 7 bits. The net result is that the seven groups of 7 bits are transformed by the seven S-Boxes into seven groups of 5 bits for 35 bits total. Rows are numbered from 0 to 3 using the first and last bits from the 7-bit input (2^2=4), and columns are numbered from 0 to 31 using the middle 5 bits of the 7-bit input (2^5=32). For instance, S_{1in} (010011) is transformed to S_{1out} (5)= S_{1out} (00101) in such a way: the first and last bits (01) gives the row number, which is row=1. The other bits (1001) gives the column number, which is column=9. Then look at the number in S_1 indexed by row=1, and column=9; that's, $S_1(1,9)=5$, the output of the S_1-box for the given input bits. Therefore, the final substituted secret key is obtained using the following algorithm:

Table 3.5: Algorithm for S-Box transformation

```
Const


        blockSize:=49;//bits
        keyLength:=136;//bits
Var

        blockNumber,padVal:long;
        SBoxInput, sboxOutput ,row, column: long;
        SBoxBinary, keyFinal, keyPad:long;
        Counter:=0,ctr:=0,c:=0: int;

Begin

   blockNumber:=keyLength/blockSize;
   padVal:=keyLength%blockSize;
   keyPad:=keyBinary;
   if(padVal~=0)
       for h:=1 to blockSize-padVal
```

```
          keyPad(keyLength+h):=0;
      end;
end;


while(counter~= blockNumber) Do
  begin
           for i:=1 to 7 and j=1 to 7  Do
               begin
                  SBoxInput(i,j):=keyPad(ctr);
                  Ctr:=ctr+1;
               end;

           for j=1 to 7 Do
              begin
                row:= SBoxInput(j,1)*2+ SBoxInput(j,7);
                column:= SBoxInput(j,2)*16+ SBoxInput (j,3)*8+
                         SBoxInput(j,4)*4+sbox(j,5)*2+ SBoxInput (j,6);

                  if(j==1)
                      sboxOutput:=S1(row,column);
                  elseif(j==2)
                      sboxOutput:=S2(row,column);
                  elseif(j==3)
                      sboxOutput:=S3(row,column);

                  elseif(j==4)
                      sboxOutput:=S4(row,column);

                  elseif(j==5)
                      sboxOutput:=S5(row,column);

                  elseif(j==6)
                      sboxOutput:=S6(row,column);
                elseif(j==7)
                      sboxOutput:=S7(row,column);
                  end;
                SBoxBinary:=convertToBinary(SBoxOuput); // to 5 bits each
                cc:=1:int;
                for i=1 to 5 DO
                  Begin
                      keyFinal(c):= SBoxBinary(cc-i);
                      cc:=cc+1;
                      c:=c+1;
                   end

            end;
         end;
       end;
```

3.4 Generation of Chaotic Attractors

In this thesis work, the required chaotic attractors are generated using a hysteresis switched second order linear system. The generation process comprises calculation of initial conditions from keys generated in section 3.3, and solving the second order linear system using the concept of second order homogeneous differential equations.

3.4.1 Hysteresis Switched Second Order Linear System

There are many techniques of generating chaos, but in this thesis work, for reasons described in the literature review section, a system called "Hysteresis Switched second order linear system" is used. It is a chaotic oscillator triggered only by initial conditions. It has no inputs except the initial conditions, X_o and Y_o. Then once triggered by the initial values, it keeps on oscillating and generating chaotic attractors for a time t, and moves from one attractor to another depending on the value of n (number of scrolls) provided due to the feedback hysteresis series as depicted in figure 2.7. The mathematical description of the hysteresis switched system depicted in figure 2.7 is given below:

$$\begin{cases} \dot{x} = y \\ \dot{y} = -x + 2\alpha\, y + H(x,n) \end{cases} \qquad \text{...(3.12)}$$

where X_o, and Y_o are the initial conditions, α is a positive constant, x and y are state variables, H(x, n) is a hysteresis series described in equations 3.13 and 3.14, and n is the number of scrolls.

$$H(x,n) = \sum_{i=1}^{n} h_i(x) \qquad \text{...(3.13)}$$

and

$$h_i(x) = \begin{cases} 1 & for\ x > i-1 \\ 0 & for\ x < i \end{cases} \qquad \text{...(3.14)}$$

3.4.2 Solution of Second Order Linear System

Derivating both sides of the first equation of system (3.12), produces the following:

$$\frac{d^2x}{dt^2} = \frac{dy}{dt} \qquad \text{---(3.15)}$$

Then the derivated y in equation (3.15) is substituted by the equivalent expression given in the second equation of system (3.12). Hence, equation (3.15) becomes:

$$\frac{d^2x}{dt^2} = \frac{dy}{dt} = -x + 2\alpha\frac{dx}{dt} \qquad \text{---(3.16)}$$

Rearranging equation (3.16), gives out

$$\frac{d^2x}{dt^2} - 2\alpha\frac{dx}{dt} + x = 0 \qquad \text{---(3.17)}$$

Equation (3.17) is a homogeneous equation of the form $a\frac{d^2x}{dt^2} + b\frac{dx}{dt} + cx = f(x)$ Where f(x)=0.

Therefore equation (3.17) is solved as follows, letting $x = e^{mt}$, and $D = \frac{dx}{dt}$, we obtain

$$\begin{aligned} &aD^2x + bDx + cx = 0 \\ &x(aD^2 + bD + c) = 0 \\ &e^{mt}\left(aD^2 + bD + c\right) = 0 \end{aligned} \qquad \text{---(3.18)}$$

Then, if e^{mt} is to be a solution, the characteristic equation has to be equated to zero ($am^2 + bm + c = 0$). Then, the expression for m is given below.

$$m = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \text{---(3.19)}$$

Using the values m_1 and m_2, obtained from equation (3.19), the solutions are $x_1 = e^{m_1 t}$ and $x_2 = e^{m_2 t}$. Then, combining both solutions to obtain the general solution gives out the following.

$$x = c_1x_1 + c_2x_2 = c_1e^{m_1 t} + c_1e^{m_2 t} \qquad \text{---(3.20)}$$

If the system is to generate chaos, its solution must be complex; that's, for $a=1$, $b=-2\alpha$, and $c=1$ (obtained from equation 3.17), $m = \alpha \pm i\beta$, where $\beta = \sqrt{1-\alpha^2}$ as proved below.

$$m = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \alpha \pm i\beta$$
$$m = \frac{2\alpha \pm \sqrt{4\alpha^2 - 4}}{2} = \alpha \pm i\beta$$
$$m = \frac{2\alpha \pm 2\sqrt{\alpha^2 - 1}}{2} = \alpha \pm i\beta \qquad \text{---(3.21)}$$
$$m = \alpha \pm \sqrt{\alpha^2 - 1} = \alpha \pm i\beta$$
$$m = \alpha \pm \sqrt{-1(1-\alpha^2)} = \alpha \pm i\beta$$
$$m = \alpha \pm i\sqrt{(1-\alpha^2)} = \alpha \pm i\beta$$

Euler's formulae,

$$e^{i\theta} = \cos\theta + i\sin\theta$$
$$e^{-i\theta} = \cos\theta - i\sin\theta \qquad \text{---(3.22)}$$

If $m_1 = \alpha + i\beta$ and $m_2 = \alpha - i\beta$, then using the Euler's formulae in (3.22), the general solution in (3.21) becomes

$$x = c_1 e^{(\alpha + i\beta)t} + c_2 e^{(\alpha - i\beta)t}$$
$$x = e^{\alpha t}(c_1 e^{i\beta t} + c_2 e^{-i\beta t})$$
$$x = e^{\alpha t}[(c_1 + c_2)\cos \beta t + (c_1 - c_2)\sin \beta t]$$

Let A= $(c_1 + c_2)$ and B=$(c_1 - c_2)$, then

$$x(t) = e^{\alpha t}[A \cos \beta t + B \sin \beta t] \qquad \text{---(3.23)}$$

The solution for the state variable y is then obtained as follows: as given in the first equation of system (3.12), $y(t) = \frac{dx}{dt}$. Therefore,

$$y(t) = \frac{dx}{dt} = \frac{d(e^{\alpha t}[A\cos \beta t + B \sin \beta t])}{dt} \qquad \text{--(3.24)}$$

Solving for y in equation 3.24, produces

$$y(t) = e^{\alpha t}[(A\alpha + B\beta)\cos \beta t + (B\alpha - A\beta)\sin \beta t] \qquad \text{---(3.25)}$$

Then, the initial conditions at t=0 are x(0)=Xo, and y(0)=Yo; that's,

$$x(0) = 1[A + 0] = Xo$$
$$y(0) = 1[(A\alpha + B\beta) + 0] = Yo \qquad \text{---(3.26)}$$

Solving the system of equations given in (3.26), we obtain A=Xo, and $B = \frac{Yo - \alpha Xo}{\beta}$. The complete solutions of system (3.12) are therefore given below:

$$x(t) = e^{\alpha t}[X_o \cos \beta t + (\frac{Y_o - \alpha X_o}{\beta}) \sin \beta t]$$

$$y(t) = e^{\alpha t}[Y_o \cos \beta t + \frac{\alpha}{\beta}(Y_o - \alpha X_o - \frac{\beta^2}{\alpha} X_0) \sin \beta t] \quad \text{---(3.27)}$$

3.4.3 Calculation of Initial Conditions

In this work, the initial conditions of system (3.12) are calculated using the values of the keyFinal array, which is the output of the S-Boxes as follows.

$$X_o = \text{var} + keyFinalX$$
$$Y_o = \text{var} + keyFinalY \quad \text{---(3.28)}$$

Where ***var*** is a user defined value, kept secret as part of the key, keyFinalX and keyFinalY are two different keys generated using different seed values. A single key can be used to calculate both initial conditions, but using two keys, one for each initial condition, is by far better from security point of view. For this case, two keys (*keyFinalX and keyFinalY*) are used. They are generated using different seed values. The user defined variable, *var*, is introduced in order to make the exhaustive key search attack more difficult. Usually keys are generated to be positive whole numbers or characters. But, in my case, a real number comprising both whole and decimal values is added to the substituted key to obtain the initial conditions. This significantly increases the effort made to obtain the key by brute force analysis or other types of cipher attacks. What's more, it is used to finely vary the chaos so as to obtain a balanced proportion of zeros and ones without significantly changing the key.

Figure 3.5 portrays 5-scroll chaotic attractors generated using the solutions of system (3.12) given in equation number (3.27). The chaos is generated using a key of length 136 bits. The extracted keys are substituted using the S-boxes as described in section 3.3.2. The substituted keys, which are the outputs of the S-boxes, and the *var* value used to generate the chaos in figure 3.5 are given below.

keyFinalX= ***11881113816021014263531863716510 2***
keyFinalY= ***11515410245191208781102411495549102***
var=5.98732516340

α=0.0049

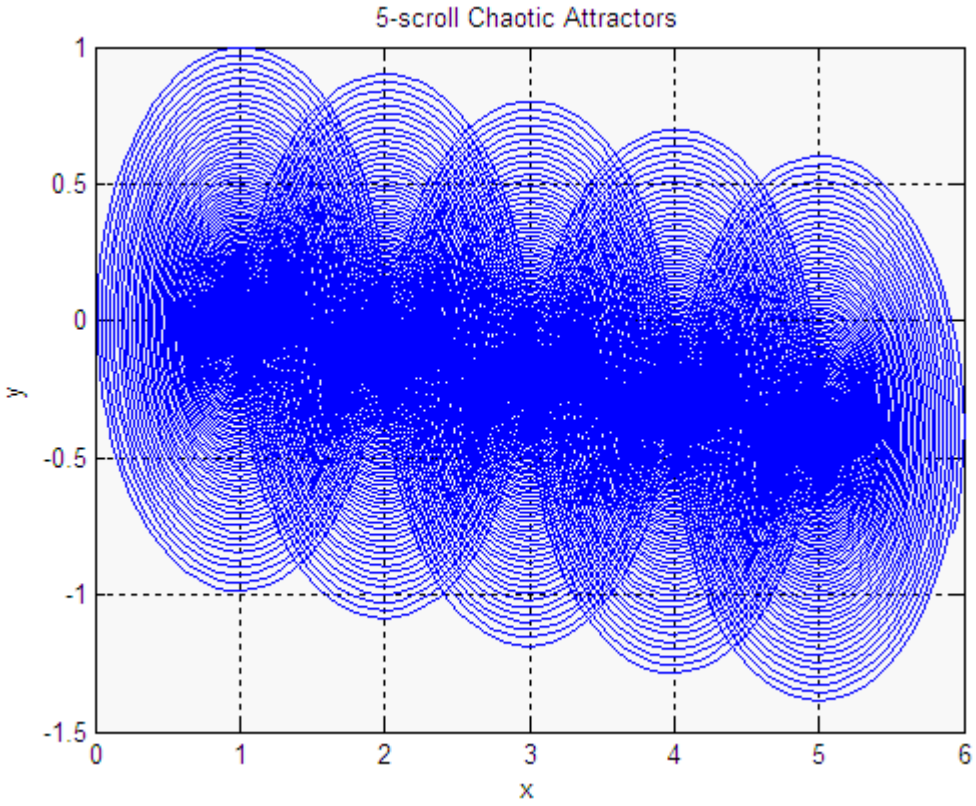


Figure 3.5: 5-scroll Chaotic Attractors.

Therefore, the corresponding initial conditions and constants are calculated as follows:

Xo=var+ ***keyFinalX=11881113816021014263531863716510***7. ***98732516340***
Yo=var+ ***keyFinalY=11515410245191208781102411495549105. 98732516340***

$\beta = \sqrt{1-\alpha^2}$ =0.999976.

The chaos generated by system (3.12) is very sensitive to initial conditions and the range within which the values of the positive constant, α, lies. In fact, as my simulation results show system (3.12) may not generate chaos for any value of α. Some papers surveyed during the literature review phase of this thesis work indicate that chaos can be generated for small initial conditions and some α values. However, in this thesis work those α values don't apply at all because I have used a long, secure key space to compute the values of the state variables x and y. The range of α values within which multi-scroll chaotic attractors can be generated is affected by the key space used. Therfore, I have devised an algorithm that determines the lower and upper limits of the α values within which chaos can be produced considering the minimum key length of the multi-

scroll chaotic encryption to be 128 bits. The monobit (or frequency) test, the number one test of randomness, is the most integral part of this algorithm. It works in such a way that it continuosly checks the proportion of zeroes and ones, and keeps on changing the value of α until the probablisitic value of the monobit test is greater or equal to the decision line value, 0.01. The monobit test is explained in detail in chapter four, section 4.3.1 of this thesis report. The algorithm devised to obatain the suitable range of α is stated in table 3.6.

Table 3.6: Algorithm for determining the range of α values.

```
Var
        p, α, Sn, Xo, Yo, Sobs, z, var,chaos: double;
        balance, alpha, probability:double;
        E: long; //E is bit sequence of chaos
        N, count:int;
  Begin
        p:=0.0;
        α :=0.0;
        var:=0;
         count:=0;
 while(p<0.01) Do
        chaos:=generateChaos(Xo, Yo, α);
         E:=convertToBinary(chaos);
         n:=E.length;
        var:=var+increament1;
        Xo:=var+KeyX; //KeyX is obtained from key generation section
        Yo :=var+KeyY; //KeyY is obtained from key generation section
         α := α + increament2;
         alpha(count):= α;
         for j:=0 to n-1 Do
            begin
                Sn:=sum(2*E(j)-1); ;
             end;
```

```
                balance(count):= Sn;
                Sobs:=|Sn|/sqrt(n);
                 z:=Sobs/sqrt(2);
                 p:=erfc(z); //erfc=complementary error function
                 probability(count):=p;
                 count++;
        end;
                display("alpha"  "balance"  "probability");
end;
```

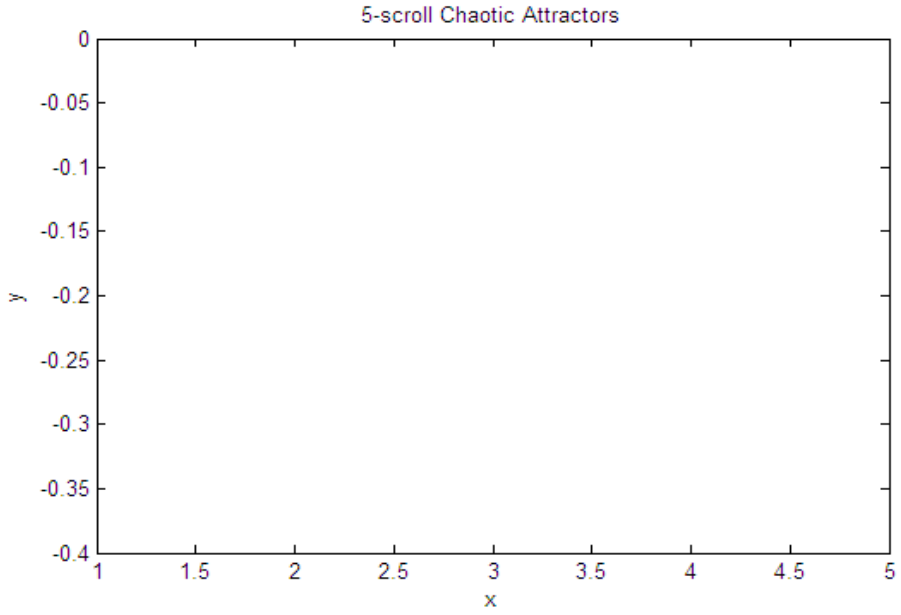
Simulation results show that for a minimum key length of 128 bits, it is possible to generate chaos for α values in the range (0.064,0). For α values out of the defined range, no chaos can be generated. For α values in the range [0.0049, 0.0045], balanced chaotic sequence can be generated; for α values very close to 0, imbalanced chaotic sequence, where the number of 1's is greater than that of 0's, is generated; and for α values in the range (0.0045,0.0009), imbalanced chaos is generated, where number of 0's is greater than that of 1's. Besides, in this thesis work, the minimum value of α is a number very close to 0, and the maximum value is 0.064. In fact, the maximum value decrease as the key space increases significantly. Some of the simulation results are depicted in table 3.7.

Table 3.7: Range of α values.

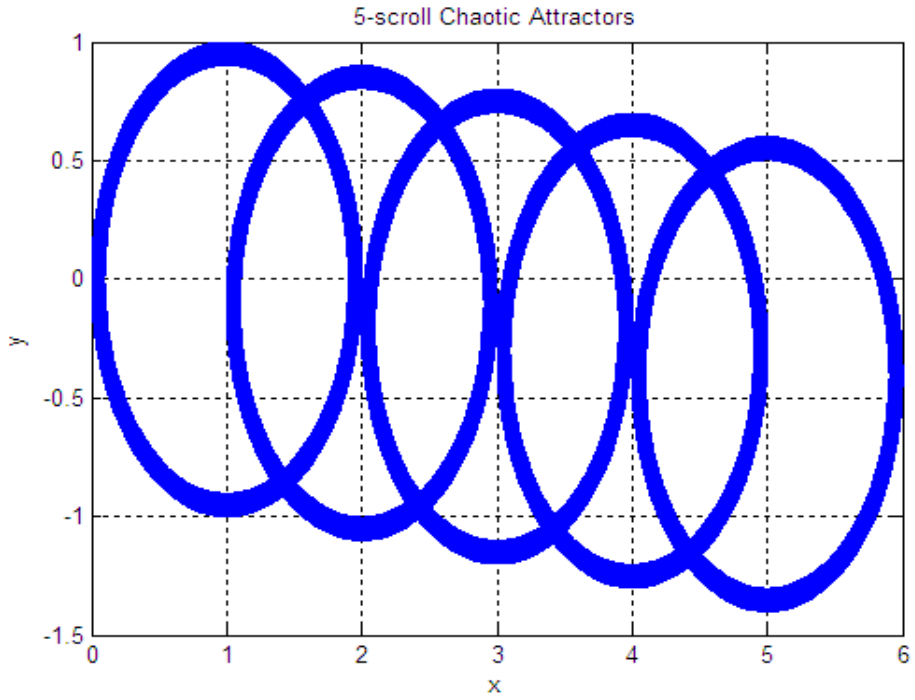
α	Balance (+→1's>0's, -→1's<0's, ±→balanced)	Probability, p(z)=erfc(z)	Decision Rule
0.00000	+	0	***P>=0.01***
0.00001	+	0	
0.00090	-	0	
0.00490	±	***0.9491***	
0.07000	+	0	

The table shows the values of α at different simulation runs, and the corresponding probability of balance of the given chaotic sequence using the algorithm in table 3.6. A probability value greater than or equal to the decision rule, ***p=0.01***, under the column header probability indicates

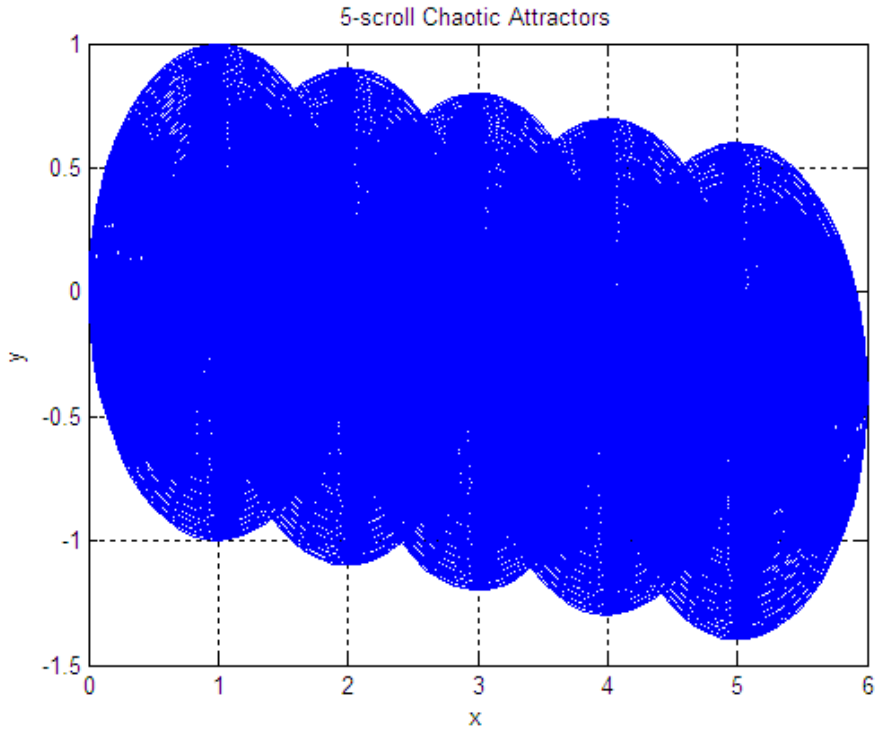
that the number of 0's and 1's in the given chaotic sequence is balanced, and conversely. Under the column balance, '+' indicates greater number of ones, '-' indicates greater number of zeros, and '±' indicates balanced number of 1's and 0's.



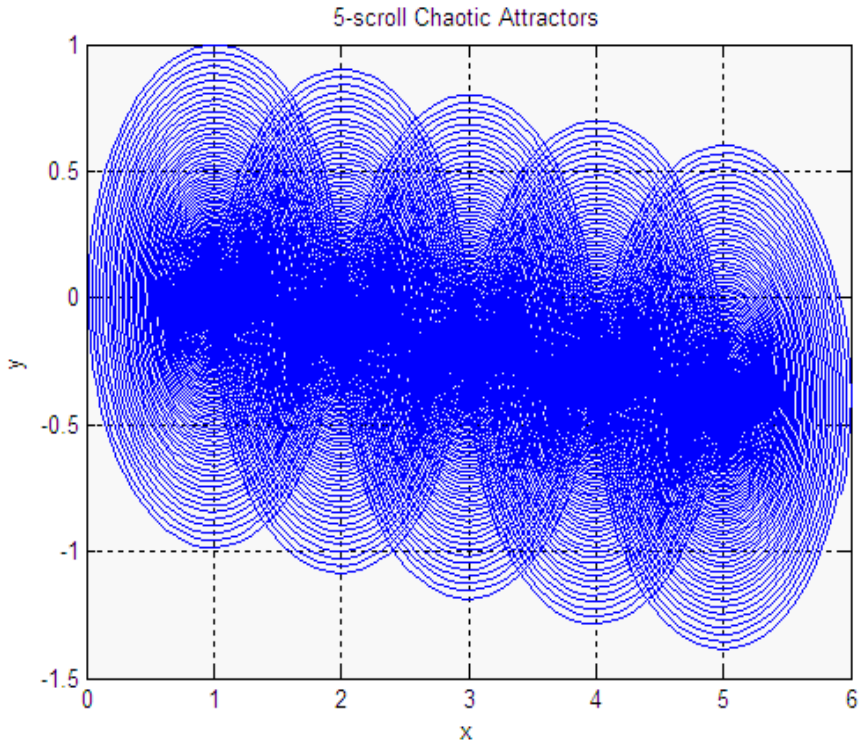
(a) 5-scroll Chaotic Attractor for *α=0.07 or α=0*



(b) 5-scroll Chaotic Attractor for ***α=0.00001***



(c) 5-scroll Chaotic Attractor for ***α=0.0009***



(d) 5-scroll Chaotic Attractor for ***α=0.0049***

Figure 3.6: 5-scroll Chaotic Attractor for ***different α values***

Corresponding chaotic figures for the α values illustrated in table 3.7 are portrayed in figures 3.6 (a), (b), (c), and (d).

3.5 Enciphering Process

Conventional enciphering is a scheme where an intelligible text (plaintext in crypto terms) is made unintelligible (ciphertext in crypto terms) using a secure key [1]. In this thesis work, the key is used to calculate the initial condition required for the generation of chaos using a system described in section 3.4, and the enciphering process is done as delineated in figure 3.7.

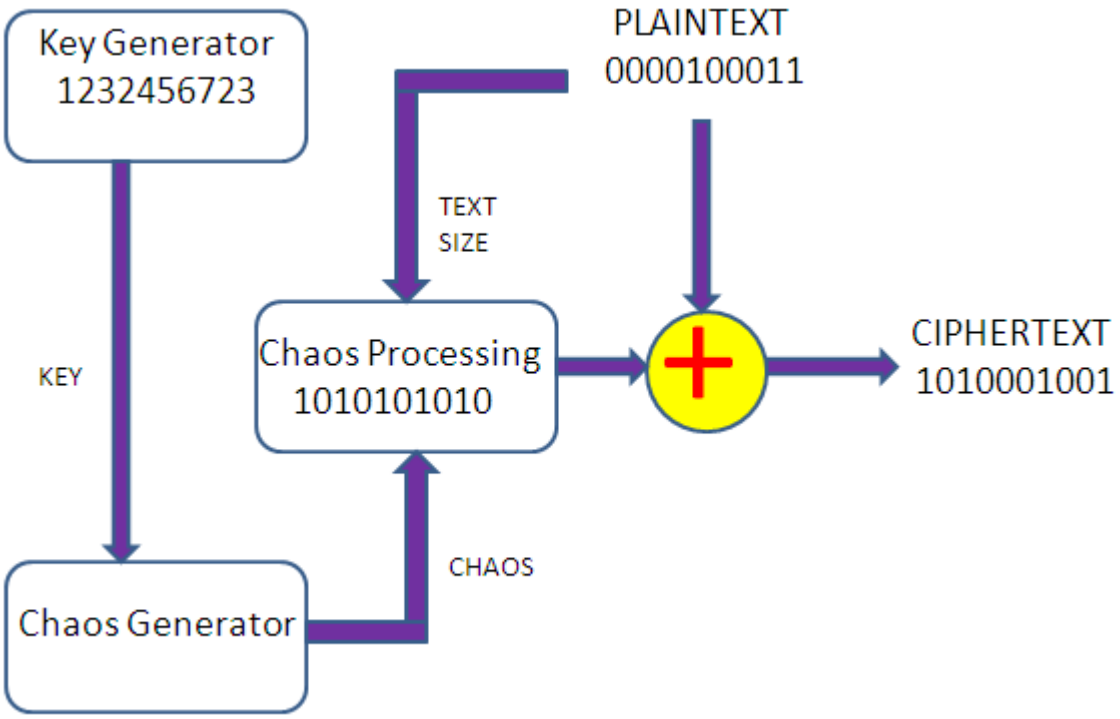


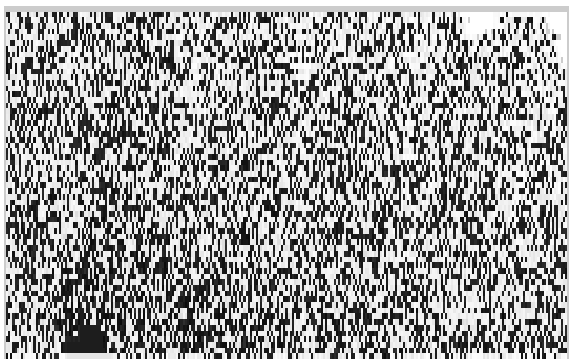
Figure 3.7: Enciphering Process

During chaos processing, three equal sections namely upper, middle and lower are cropped from the overall balanced chaos shown in figure 3.5, and converted to grayscales as depicted in figures 3.8 (a), (b), and (c), and mixed together (just after conversion to binary sequence) to further increase the probability of the balance of 1's and 0's by avoiding localized imbalances. Then, the combination of the three crops is resized as per the size of the input plaintext as portrayed in figure 3.9. That's, the mixed chaos is rescaled in such a way as to have dimensions equal to the

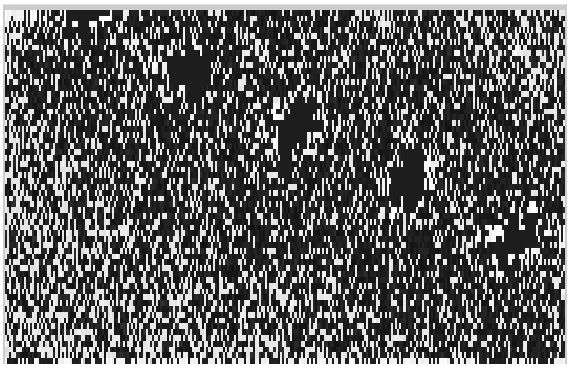
square root of the size of the binary sequence of the input plaintext. For the security of the algorithm to be strong, the minimum chaotic sequence size has to be 128 bits. If text size is less than 128 bits, it is padded with zeros to meet the minimum size. The pseudo code for the algorithm used for cropping, mixing, padding and resizing is given in table 3.8.

Table 3.8: Algorithm for chaos processing

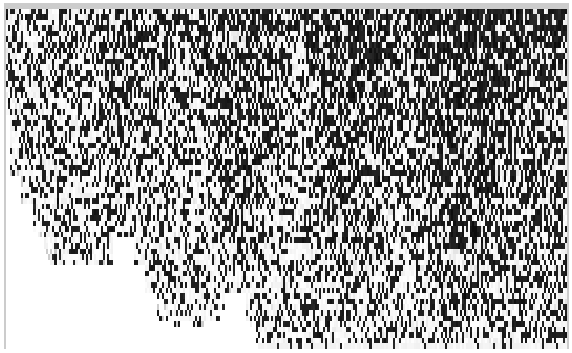
```
Var
        sizedChaos, crop-1, crop-2,crop-3,mixedCrop:image;
       paddedText: long;
      textSize, paddedSize, scale :int;
Begin
    textSize:= plaintextBinary.length;
   mixedCrop := crop-1 XOR crop-2 XOR crop-3;
   If (textSize<128) Do
     Begin
       For j: =1 to 128 Do
            Begin
              If (j<=textSize) Do
                Begin
                    paddedText(j):= plaintextBinary(j);
               Else
                    paddedText(j):=0;
               End;
            End;
      End;
      paddedSize:= paddedText.length;
      scale:=sqrt(paddedSize);
     sizedChaos:=imresize(mixedCrop [scale scale])
  End;
```



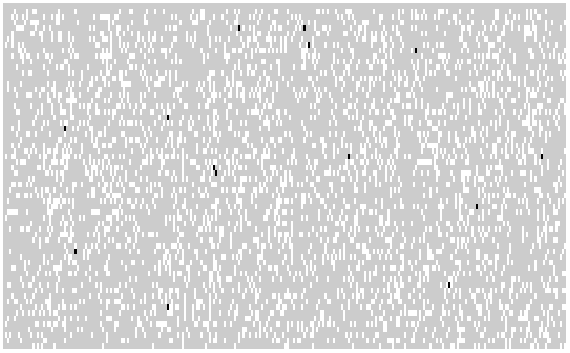
(a) Upper Crop



(b) Middle Crop



(b) Lower Crop



(d) Mixed=a XOR b XOR c

Figure 3.8: chaos cropping and mixing.

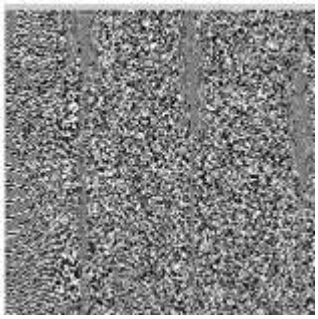


Figure 3.9: resized balanced chaos, $\sqrt{\boldsymbol{textSize}}\,\boldsymbol{x}\sqrt{\boldsymbol{textSize}}$

Eventually, figure 3.9 is converted to a sequence of 0's and 1's, and the enciphering process is done by mixing the binary sequence of plaintext with it using the logical bitwise XOR operator. The pseudo code of algorithm devised for the enciphering process of the multi-scroll chaotic encryption system is given in table 3.7.

Table 3.9: Algorithm for enciphering process

Input: sizedChaosBinary, paddedTextBinary;

```
Var
   ciphertext: long;
   count:=1,textSize:int;
Begin
   textSize:= paddedTextBinary.length;
  While (count<= textSize) Do
     Begin
      For i:=1 to w  Do
        begin
          For j:=1 to h  Do
           Begin
             ciphertext[count]:= paddedTextBinary[count] XOR sizedChaosBinary[count];
             count :=count+1;
           End;
        End;
     End;
End;
```

3.6 Deciphering Process

The process of deciphering in this chaotic algorithm is essentially the same as the enciphering process. The rule is as follows: the same key used during the enciphering process is used in the deciphering process, but the cipher text, in lieu of the plaintext, is used as input to the chaotic algorithm. The ciphertext, which is the output of the enciphering process, is mixed using the XOR operator with the appropriately sized chaos generated using the same key as depicted in figure 3.10.

plaintext [textSize]= ciphertext[textSize] XOR sizedChaos[textSize]

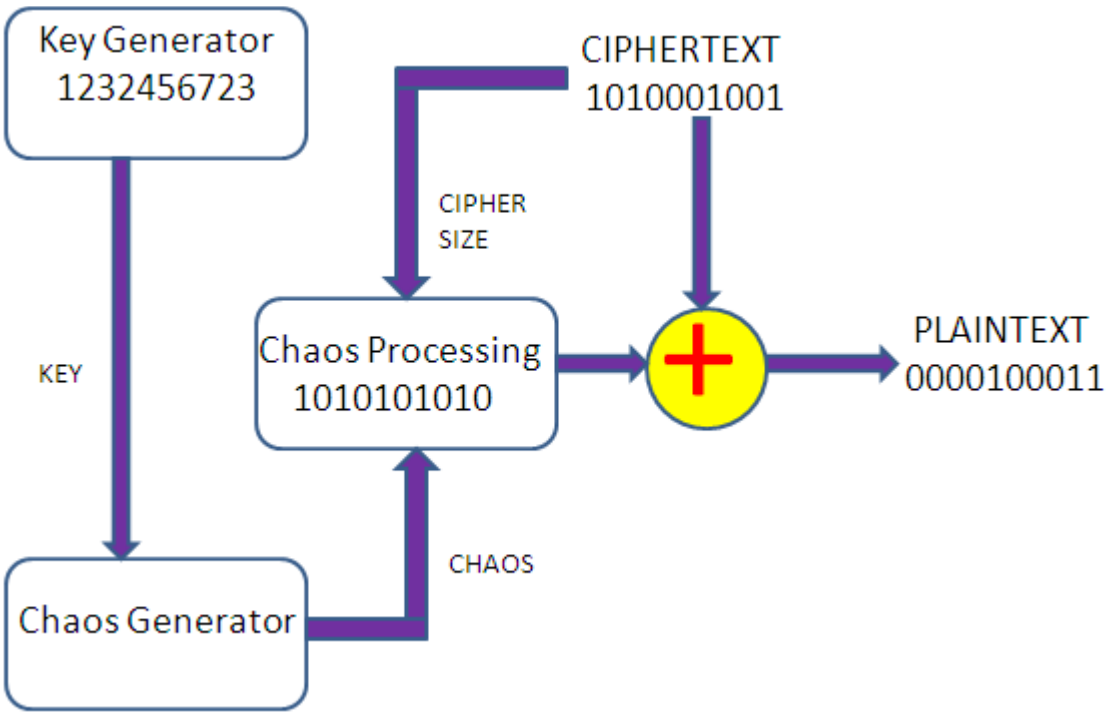


Figure 3.10: Deciphering Process.

3.7 Integrity of Shared Image and Key Exchange Method

In this section, the mechanism used to check the integrity of the shared image from which both enciphering and deciphering keys are extracted and the key exchange scheme are discussed.

3.7.1 Integrity of Shared Image

In modern cryptography, there are a number of data integrity mechanisms that provide protection against unauthorized, malicious modifications of data. One way is to use hash cryptography. There are many hashing techniques that provide data integrity and origin authentication [6, 7, 8]. In this thesis work, I have used a keyed hash function called hashed message authentication code (or HMAC) to check the integrity of the shared image. HMAC is generated in a way depicted in figure 3.11.

The publicly shared image from which the encryption and decryption key is extracted might be tampered by malicious parties. As a result, the receiving party may not be able to successfully recover the clear text from the unintelligible ciphertext due to the fact that the extracted decryption key is different from the encryption key. Then, to check whether the publicly shared

image has been maliciously manipulated or not, a fixed length manipulation detection code called HMAC is computed from the combination of the shared image and a secret key at the sending end and sent to the receiving side together with the message as portrayed in figure 3.11. The secret hash key is secretly shared between the sending and receiving parties. At the receiving end, on the receipt of the message and HMAC, the receiving party generates HMAC from the shared image using the same hashing key, and compare it with the received HMAC. If they are found to be equal, he/she proceeds with the decryption key generation being assured that the shared image's integrity is intact; he/she, otherwise, request change of shared image and retransmission of message.

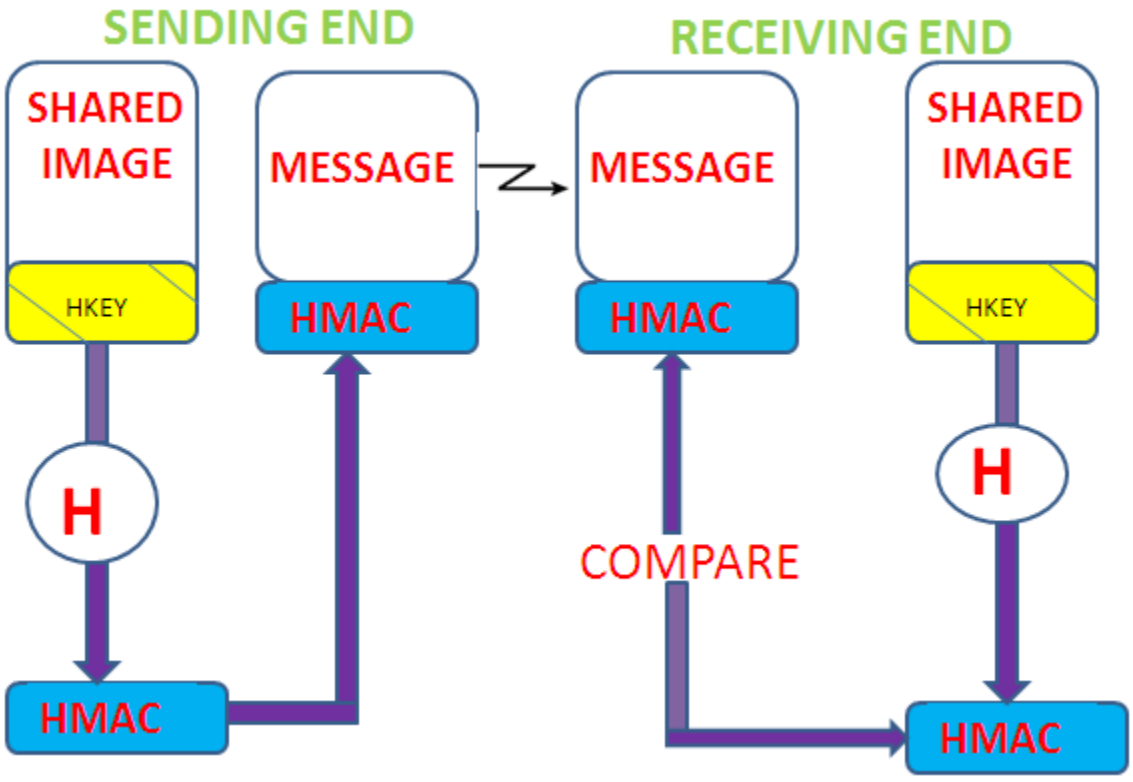


Figure 3.11: Generation and comparison of HMAC.

In this work, I have used secure hash algorithm (SHA-1). That's, the shared image is combined with a secret key at both sending and receiving ends and hashed using SHA-1 to produce a fixed length string of numbers. The output of SHA-1 is 160-bit long comprising 40 hexadecimal numbers. The hashing key, ***HKey=696212147483646***, is the concatenation of the multiplier and modulus (or HKey=a+m) of the PRNG used to extract the key from the shared image. The

encircled H in figure 3.11 stands for the hashing function, SHA-1, and it hashes the shared image and secret key into a 160-bit, 40 hexadecimal numbers. The HMAC of the shared image depicted in figure 3.2 is given below. It serves two purposes namely shared image integrity check, and origin authentication.

HMAC=H (HKey, Shared Image) =1DF5FAF45ABC4CA81DA967D70F3EB664EB5DFFF2

3.7.2 Key Exchange Method

In general, in a secret key encryption, as the number of communicating parties increases the exchange of the secret key becomes insecure. That's, n users who want to communicate in pairs need **n * (n - 1)/2** keys. The number of keys needed increases at a rate proportional to the square of the number of users! So a property of symmetric encryption systems is that they require a means of **key distribution**. Public key systems excel at key management due to the nature of the public key approach; we can send a public key in an e-mail message or post it in a public directory [2]. In this thesis work, in order to make the secret key exchange secure a publicly shared image from which the key extract is used, and the information required to extract the key is communicated using public cryptography, RSA.

Suppose you need to send a protected message to someone you do not know and who does not know you. The situation depends on being able to exchange an encryption key in such a way that nobody else can intercept it. The problem of two previously unknown parties exchanging cryptographic keys is both hard and important. Indeed, the problem is almost circular: To establish an encrypted session, you need an encrypted means to exchange keys. Public key cryptography can help because asymmetric keys come in pairs, one half of the pair can be exposed without compromising the other half. [2].

Suppose S and R want to derive a shared symmetric key. Suppose also that S and R both have public keys for a common encryption algorithm; call these k_{PRIV-S}, k_{PUB-S}, k_{PRIV-R}, and k_{PUB-R}, for the private and public keys for S and R, respectively. The simplest solution is for S to choose any symmetric key K, and send E (k_{PRIV-S},K) to R. Then, R takes S's public key, removes the

encryption, and obtains K. Alas, any eavesdropper who can get S's public key can also obtain K. Instead, let S send E (k_{PUB-R}, K) to R. Then, only R can decrypt K. Unfortunately, R has no assurance that K came from S [2]. But there is a useful alternative. The solution is for S to send to R:

$$E(k_{PUB-R}, E(k_{PRIV-S}, K))$$

We can think of this exchange in terms of lockboxes and keys. If S wants to send something protected to R (key information in this case), then the exchange works something like this. S puts the protected information in a lockbox that can be opened only with S's public key. Then, that lockbox is put inside a second lockbox that can be opened only with R's private key. R can then use his private key to open the outer box (something only he can do) and use S's public key to open the inner box (proving that the package came from S). In other words, the protocol wraps the protected information in two packages: the first unwrapped only with S's public key, and the second unwrapped only with R's private key. This approach is illustrated in Figure 3.8.

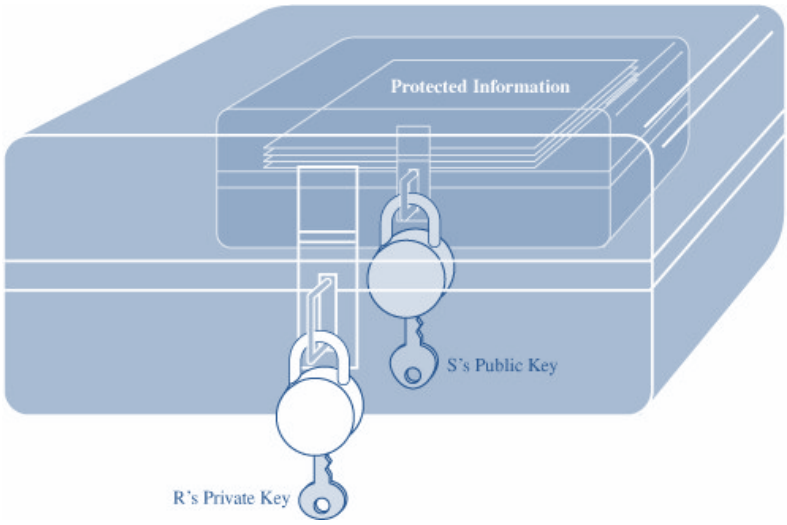


Figure 3.12: Secure Secret Key Exchange [2]

Eventually, all the information required for the extraction of the key from the publicly shared image and other constants in this chaotic algorithm is exchanged being encrypted using a public-key RSA encryption. Let ***INFO=Seeds*** + ***n*** + ***var*** + ***α*** + ***HKey*** + ***LenMul*** +…; then, it is sent as follows:

$$E\,(K_{PUB\text{-}R},\ E\,(K_{PRIV\text{-}S},\ INFO))$$

Where E stands for public encryption RSA, Kpub=public key, Kpriv=private key, R stands for Receiver, S stands for Sender, n is number of scrolls, and LenMul is the number of digits contained in the multiplier of the PRNG used for key extraction.

Chapter Four

TEST AND SIMULATION OF THE CHAOTIC ENCRYPTION

4.1 Overview

It is a customary practice to perform various types of pertinent tests on a system or algorithm after the completion of its design phase so as to validate or reject it. In this thesis work, three types of tests namely functional test, statistical randomness test, and Monte Carlo Simulation test are carried out to check the functionality of the design, and security of the cipher produced, respectively.

4.2 Functional Test

To verify if the designed multi-scroll chaotic algorithm can function as required, a sample plaintext was enciphered and then deciphered. A plaintext is browsed using the GUI depicted in figure 4.1, and is enciphered using an encryption key to convert it into a form which is unintelligible. Eventually, the ciphertext is deciphered using the same key to check if the chaotic deciphering process can fully recover the clear text (or plaintext) back from it.

As copied from the text areas of the GUI in figure 4.1, the plaintext, ciphertext, and decrypted text are respectively displayed below verifying that the chaotic algorithm works as designed and required. That's, the original plaintext and the recovered (or decrypted text) are the same.

Plaintext=

How do we protect our most valuable assets? One option is to place them in a safe place, like a bank. We seldom hear of a bank robbery these days, even though it was once a fairly lucrative undertaking. In the American Wild West, banks kept large amounts of cash on hand, as well as gold and silver, which could not be traced easily. In those days, cash was much more commonly used than checks. Communications and transportation were primitive enough that it might have been hours before the legal authorities were informed of a robbery and days before they could actually arrive at the scene of the crime, by which time the robbers were long gone. To control the situation, a single guard for the night was only marginally effective. Should you have wanted to commit a robbery, you might have needed only a little common sense and perhaps several days to analyze the

situation; you certainly did not require much sophisticated training. Indeed, you usually learned on the job, assisting other robbers in a form of apprenticeship. On balance, all these factors tipped very much in the favor of the criminal, so bank robbery was, for a time, considered to be a profitable business. Protecting assets was difficult and not always effective. Today, however, asset protection is easier, with many factors working against the potential criminal. Very sophisticated alarm and camera systems silently protect secure places like banks whether people are around or not. The techniques of criminal investigation have become so effective that a person can be identified by genetic material (DNA), fingerprints, retinal patterns, voice, a composite sketch, ballistics evidence, or other hard-to-mask characteristics. The assets are stored in a safer form. For instance, many bank branches now contain less cash than some large retail stores because much of a bank's business is conducted with checks, electronic transfers, credit cards, or debit cards. Sites that must store large amounts of cash or currency are protected with many levels of security: several layers of physical systems, complex locks, multiple-party systems requiring the agreement of several people to allow access, and other schemes. Significant improvements in transportation and communication mean that police can be at the scene of a crime in minutes; dispatchers can alert other officers in seconds about the suspects to watch for. From the criminal's point of view, the risk and required sophistication are so high that there are usually easier ways than bank robbery to make money. Cryptography, secret writing, is the strongest tool for controlling against many kinds of security threats. Well-disguised data cannot be read, modified, or fabricated easily. Cryptography is rooted in higher mathematics: group and field theory, computational complexity, and even real analysis, not to mention probability and statistics. Fortunately, it is not necessary to understand the underlying mathematics to be able to use cryptography. [2]

Ciphertext=

:"u1:u"0u%':!06!u:'u8:&!u#494790u4&&0!&ju;0u:%!<:;u<&u!:u%9460u!=08u<;u4u&430u%9460yu9<>0u4u74;>{u0u&091:8u=04'u:3u4u74;>u':770',u!=0&0u14,&yu0#0;u!=:2=u<!u"4&u:;60u4u34<'9,u96'4!<#0u;10'!4><;2{u;u!=0u80'<64;u<91u0&!yu74;>&u>0%!u94'20u48:;!&u:3u64&=u:;u=4;1yu4&u"099u4&u2:91u4;1u&<9#0'yu"=<6=u6:91u;:!u70u!'4601u04&<9,{u;u!=:&0u14,&yu64&=u"4&u86=u8:'0u6:88:;9,u&01u!=4;u6=06>&{u:88;<64!<:;&u4;1u!'4;&%:'!4!<:;u"0'0u%'<8<!<#0u0;:2=u!=4!u<!u8<2=!u=4#0u700;u=:'&u703:'0u!=0u90249u4!=:'<!<0&u"0'0u<;3:'801u:3u4u':770',u4;1u14,&u703:'0u!=0,u6:91u46!499,u4"<#0u4!u!=0u&60;0u:3u!=0u6'<80yu7,u"=<6=u!<80u!

=0u':770'&u"0'0u9:;2u2:;0{u:u6:;!':9u!=0u&<!4!<:;yu4u&<;290u24'1u3:'u!=0u;<2=!u"4&u:;9,u84'2<;499,u03306!<#0{u=:91u,:u=4#0u"4;!01u!:u6:88<!u4u':770',yu,:u8<2=!u=4#0u;00101u:;9,u4u9<!!90u6:88:;u&0;&0u4;1u%0'=4%&u&0#0'49u14,&u!:u4;49,/0u!=0u&<!4!<:;nu,:u60'!4<;9,u1<1u;:!u'0$<'0u86=u&:%=<&!<64!01u!'4<;<;2{u;1001yu,:u&499,u904';01u:;u!=0u?:7yu4&&<&!<;2u:!=0'u':770'&u<;u4u3:'8u:3u4%%'0;!<60&=<%{u;u7494;60yu499u!=0&0u346!:'&u!<%%01u#0',u86=u<;u!=0u34#:'u:3u!=0u6'<8<;49yu&:u74;>u':770',u"4&yu3:'u4u!<80yu6:;&<10'01u!:u70u4u%':3<!4790u7&<;0&&{u':!06!<;2u4&&0!&u"4&u1<33<69!u4;1u;:!u49"4,&u03306!<#0{:14,yu=:"0#0'yu4&&0!u%':!06!<:;u<&u04&<0'yu"<!=u84;,u346!:'&u":'><;2u424<;&!u!=0u%:!0;!<49u6'<8<;49{u0',u&:%=<&!<64!01u494'8u4;1u6480'4u&,&!08&u&<90;!9,u%':!06!u&06'0u%9460&u9<>0u74;>&u"=0!=0'u%0:%90u4'0u4':;1u:'u;:!{u=0u!06=;<$0&u:3u6'<8<;49u<;#0&!<24!<:;u=4#0u706:80u&:u03306!<#0u!=4!u4u%0'&:;u64;u70u<10;!<3<01u7,u20;0!<6u84!0'<49u}|yu3<;20'%'<;!&yu'0!<;49u%4!!0';&yu#:<60yu4u6:8%:&<!0u&>0!6=yu7499<&!<6&u0#<10;60yu:'u:!=0'u=4'1x!:x84&>u6=4'46!0'<&!<6&{u=0u4&&0!&u4'0u&!:'01u<;u4u&430'u3:'8{u:'u<;&!4;60yu84;,u74;>u7'4;6=0&u;:"u6:;!4<;u90&&u64&=u!=4;u&:80u94'20u'0!4<9u&!:'0&u7064&0u86=u:3u4u74;>r&u7&<;0&&u<&u6:;16!01u"<!=u6=06>&yu0906!':;<6u!'4;&30'&yu6'01<!u64'1&yu:'u107<!u64'1&{u<!0&u!=4!u8&!u&!:'0u94'20u48:;!&u:3u64&=u:'u6"0;6,u4'0u%':!06!01u"<!=u84;,u90#09&u:3u&06'<!,ou&0#0'49u94,0'&u:3u%=,&<649u&,&!08&yu6:8%90u9:6>&yu89!<%90x%4'!,u&,&!08&u'0$<'<;2u!=0u42'0080;!u:3u&0#0'49u%0:%90u!:u499:"u4660&&yu4;1u:!=0'u&6=080&{u<2;<3<64;!u<8%':#080;!&u<;u!'4;&%:'!4!<:;u4;1u6:88;<64!<:;u804;u!=4!u%:9<60u64;u70u4!u!=0u&60;0u:3u4u6'<80u<;u8<;!0&nu1<&%4!6=0'&u64;u490'!u:!=0'u:33<60'&u<;u&06:;1&u47:!u!=0u&&%06!&u!:u"4!6=u3:'{u':8u!=0u6'<8<;49r&u%:<;!u:3u#<0"yu!=0u'<&>u4;1u'0$<'01u&:%=<&!<64!<:;u4'0u&:u=<2=u!=4!u!=0'0u4'0u&499,u04&<0'u"4,&u!=4;u74;>u':770',u!:u84>0u8:;0,{',%!:2'4%=,yu&06'0!u"'<!<;2yu<&u!=0u&!':;20&!u!::9u3:'u6:;!':99<;2u424<;&!u84;,u><;1&u:3u&06'<!,u!='04!&{u099x1<&2<&01u14!4u64;;:!u70u'041yu8:1<3<01yu:'u347'<64!01u04&<9,{u',%!:2'4%=,u<&u'::!01u<;u=<2=0'u84!=084!<6&ou2':%u4;1u3<091u!=0:',yu6:8%!4!<:;49u6:8%90<!,yu4;1u0#0;u'049u4;49,&<&yu;:!u!:u80;!<:;u%':747<9<!,u4;1u&!4!<&!<6&{u:'!;4!09,yu<!u<&u;:!u;060&&4',u!:u;10'&!4;1u!=0u;10'9,<;2u84!=084!<6&u!:u70u4790u!:u&0u6',%!:2'4%=,{_

Deciphered text=

How do we protect our most valuable assets? One option is to place them in a safe place, like a bank. We seldom hear of a bank robbery these days, even though it was once a fairly lucrative

undertaking. In the American Wild West, banks kept large amounts of cash on hand, as well as gold and silver, which could not be traced easily. In those days, cash was much more commonly used than checks. Communications and transportation were primitive enough that it might have been hours before the legal authorities were informed of a robbery and days before they could actually arrive at the scene of the crime, by which time the robbers were long gone. To control the situation, a single guard for the night was only marginally effective. Should you have wanted to commit a robbery, you might have needed only a little common sense and perhaps several days to analyze the situation; you certainly did not require much sophisticated training. Indeed, you usually learned on the job, assisting other robbers in a form of apprenticeship. On balance, all these factors tipped very much in the favor of the criminal, so bank robbery was, for a time, considered to be a profitable business. Protecting assets was difficult and not always effective. Today, however, asset protection is easier, with many factors working against the potential criminal. Very sophisticated alarm and camera systems silently protect secure places like banks whether people are around or not. The techniques of criminal investigation have become so effective that a person can be identified by genetic material (DNA), fingerprints, retinal patterns, voice, a composite sketch, ballistics evidence, or other hard-to-mask characteristics. The assets are stored in a safer form. For instance, many bank branches now contain less cash than some large retail stores because much of a bank's business is conducted with checks, electronic transfers, credit cards, or debit cards. Sites that must store large amounts of cash or currency are protected with many levels of security: several layers of physical systems, complex locks, multiple-party systems requiring the agreement of several people to allow access, and other schemes. Significant improvements in transportation and communication mean that police can be at the scene of a crime in minutes; dispatchers can alert other officers in seconds about the suspects to watch for. From the criminal's point of view, the risk and required sophistication are so high that there are usually easier ways than bank robbery to make money. Cryptography, secret writing, is the strongest tool for controlling against many kinds of security threats. Well-disguised data cannot be read, modified, or fabricated easily. Cryptography is rooted in higher mathematics: group and field theory, computational complexity, and even real analysis, not to mention probability and statistics. Fortunately, it is not necessary to understand the underlying mathematics to be able to use cryptography.

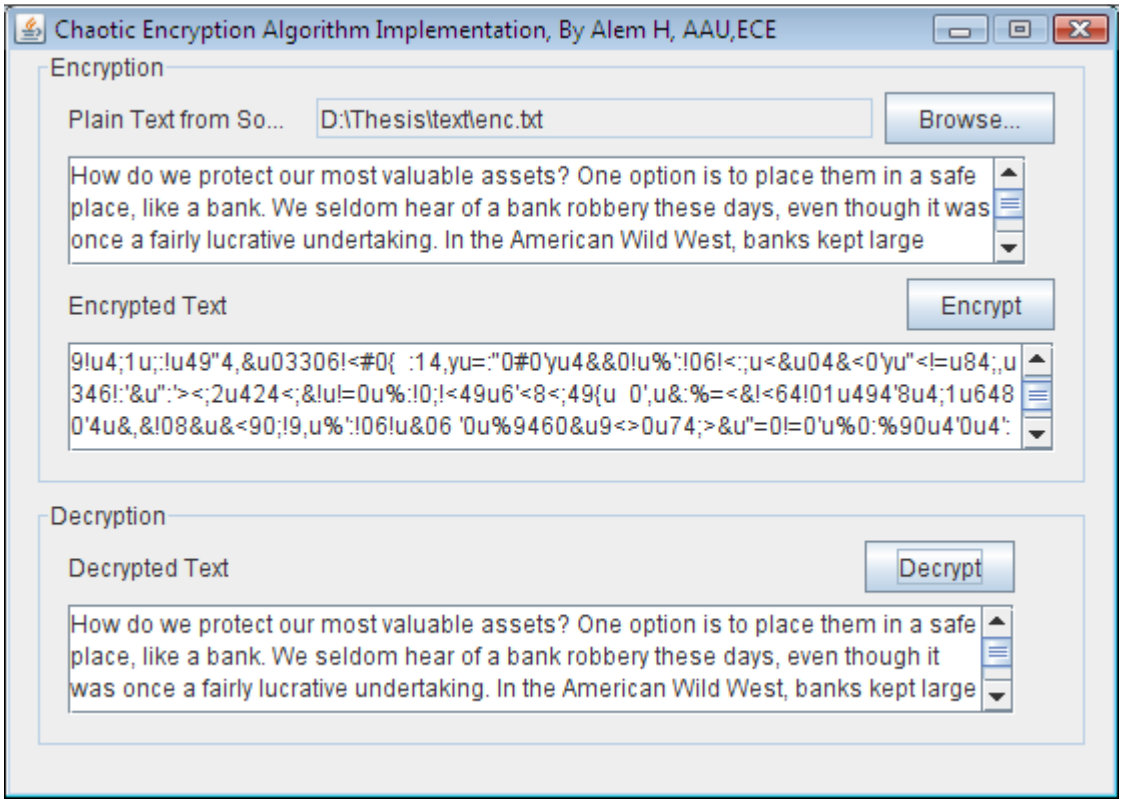


Figure 4.1: Chaotic Crypto System

4.3 Randomness Test

A number of different types of statistical tests can be applied to a given sequence to attempt to compare and evaluate the sequence to a truly random sequence. Randomness is a probabilistic property. The likely outcome of statistical tests, when applied to a truly random sequence, is known a priori and can be described in probabilistic terms. There are an infinite number of possible statistical tests, each assessing the presence or absence of a pattern which, if detected, would indicate that the sequence is nonrandom [31, 32].

In this thesis work, I have used some statistical standard randomness tests suitable for stream cipher defined in the NIST (National Institute of Standards and Technology of USA) and BSI (German Federal Office for Information Security) Test Suites to test the randomness of the binary sequences produced by chaotic oscillator or generator. The security of any cryptographic algorithm entirely depends on the security of the cipher and key stream. Here, we say that the cipher is secure and less prone to statistical attacks only when it is proved that the chaotic sequence with which it is mixed

during the enciphering process is random and unpredictable. As a result, the tests conducted in this work focus on a variety of different types of non-randomness that could exist in the chaotic sequence. The tests performed in this thesis include: Monobit test, block frequency test, run test, spectral test, and linear complexity test.

The tests, defined in the NIST and BSI test suites and conducted in this thesis work, use either the ***standard normal*** (z) or the ***chi-square*** (χ^2) as reference distribution. The standard normal distribution is used to compare the value of the test statistic obtained from the chaotic generator output sequence with the expected value of the statistic under the assumption of randomness. The test statistic for the standard normal distribution is of the form $\mathbf{z = (x - \mu)/\sigma^2}$, where x is the sample test statistic value, and μ and σ^2 are the expected value and the variance of the test statistic respectively. The χ^2 distribution is used to compare the goodness-of-fit of the observed frequencies of a sample measure to the corresponding expected frequencies of the hypothesized distribution. The test statistic is of the form $\boldsymbol{\chi^2 = \sum((o_i - e_i)^2/e_i)}$, where o_i and e_i are the observed and expected frequencies of occurrence of the measure, respectively. As clearly specified by aforementioned test suites, the ***Decision Rule*** for all of the tests conducted in this thesis work is at the ***1 % Level***. That's, if the computed P-value (probabilistic value) is < 0.01, then we conclude that the chaotic generator is non-random. Otherwise, we conclude that the chaotic generator is random.

4.3.1 Monobit Test

The central point of the test is the proportion of zeroes and ones in the entire sequence generated by the chaotic generator. The noble goal of this test is to determine whether the number of ones and zeros in the sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones in the sequence to ½ which in turn reflects the fraction of zeroes. That is, the number of ones and zeroes in the chaotic output bit sequence should be about the same. All other randomness tests depend on the passing of this test.

The test is denoted by *frequency (n),* where n is the length of the sequence bit string, and the sequence of the bits are represented by $\epsilon = \epsilon_1\, \epsilon_2, \ldots, \epsilon_n$. Then, the statistical value, $S_{obs,}$ is the absolute value of the sum of the X_i (where, $X_i = 2\, \epsilon_i - 1 = \pm 1$) in the sequence divided by the square root of

the length of the sequence. The reference distribution for the test statistic is half normal. If z (where $z= S_{obs} /\sqrt{2}$) is distributed as normal, then |z| is distributed as half normal. Plus, if the sequence is random, then the plus and minus ones will tend to cancel one another out so that the test statistic will be about 0. If there are too many ones or too many zeroes, then the test statistic will tend to be larger than zero. The steps or test procedures used in conducting this test are described as follows:

- ✓ ***Conversion to ±1***: The zeros and ones of the chaotic sequence (ϵ) are converted to values of –1 and +1 and are added together to produce $S_n = X_1 + X_2 + \ldots + X_n$, where $X_i = 2\,\epsilon_i - 1$.

  Variable: S_n: =0: long;

  For i=1 to n Do //n=sequence.length;

  $S_n = S_n + 2 * \epsilon\,[i] - 1;$

  End;

  In this work, sample chaotic attractors are generated as explained and illustrated in chapter three. Then, they are converted into a sequence of bits for the test convenience. The length of the sequence bit string is n=sequence.length=24568, and S_n=-10.

- ✓ ***Computation of the test statistic, S_{obs}***: it is the ration of the sum of 1's and -1's to the square root of the sequence bit length; i.e., $\boldsymbol{S_{obs}} = \frac{|Sn|}{\sqrt{n}}$. In this experiment, n= 24568, and S_n=-10. Hence, $\boldsymbol{S_{obs}} = \frac{|-10|}{\sqrt{24568}} = 0.0638.$, $z = \frac{S_{obs}}{\sqrt{2}} = \frac{0.0638.}{\sqrt{2}} = 0.0451.$
- ✓ ***Computation of the P-value***: the P-value in this test is computed using the complementary error function as follows: $P - value = erfc(z)$, where erfc is given by

$$\boldsymbol{erfc(z)} = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-u^2}\, du \quad \ldots(4.1)$$

The probability value is therefore obtained by, $P - value = \boldsymbol{erfc(z)} = 0.9491$, where the error function is stated in equation 4.1.

Since the P-value obtained above is >0.01; that's, P-value = 0.9491 > 0.01.We can therefore conclude that the chaotic generator is random. Besides, if the P-value were smaller (< 0.01), then this would be caused by $|\boldsymbol{S_n}|$ or $|\boldsymbol{S_{obs}}|$ being large. Large positive values of S_n are indicative of too many ones, and large negative values of S_n are indicative of too many zeros. But in this case,

the P-value is greater than the decision rule or level (1%) indicating that the number of 1's and 0's in the generated sequence is almost balanced, and hence the chaotic generator is random.

4.3.2 Frequency Test within a Block

The focus of the test is the proportion of one's within N-bit blocks. The cardinal objective of this test is to determine whether the frequency of ones in an N-bit block is approximately N/2, as would be expected under an assumption of true randomness.

Frequency (n, N), where n is the length of bit string in the generated sequence $\epsilon = \epsilon_1 \epsilon_2, \ldots, \epsilon_N$, and N is the length of each M-block, is the block frequency. The reference distribution for the test statistic is the ***chi-square*** (χ^2) distribution. χ^2 (*obs)* is a measure of how well the observed proportion of 1's within a given N-bit block matches the expected proportion, which is 1/2. The required statistic is described and computed as follows where n=sequence.length. N & M are selected as per the NIST input recommendations:

- ✓ ***Partitioning***: the generated sequence of bits is partitioned into M non-overlapping blocks having a bit length of N. N and M are chosen according to the NIST input recommendation for this particular test. That's, it is recommended that each sequence to be tested consists of a minimum of 100 bits (i.e., $n \geq 100$), and $n \geq MN$. The block size M should be selected such that $M \geq 20$, $M > .01n$ and $N < 100$. In this thesis work N=98 which is less than 100 as recommended. Therefore, the number of blocks, M, is obtained by integer division of n by N as depicted in equation 4.2.

$$M = (int)\frac{n}{N} \qquad \ldots(4.2)$$

M= (int) n/N=250, the unused bits of length ***n-M*N*** are discarded. No padding scheme is used. The generated sequence is then partitioned into 250 non-overlapping blocks and the corresponding π's are calculated as follows.

```
Varibale: c:=0, k:=0, cc:=0: int;
while(c<M) Do
        Begin
                for j=k+1 to k+N Do
                Begin
                        M1 (cc)=E(j);
                        cc=cc+1;
```

```
        end
        cc=0;
        k=j;
        π(c)=sum(M1)/N;
        M1=0;
        c=c+1;
    end
```

π_i, the proportion of ones in each M-bit block, is computed using the equation stated in 4.3:

$$\pi_i = \frac{\sum_{j=1}^{N} \epsilon_j}{N}, \text{ for } 1 \le i \le M \quad \ldots\ldots\ldots(4.3)$$

In this work, 250 $\pi_i's$, one for each block, were computed. Each π_i gives the ratio of ones in each block of bit length 98. That's, sum of ones in each block divided by the block size, 98. For instance, π_1 = 0.4490, ..., π_{250} = 0.5000 are some of the values of $\pi_i's$ obtained.

- ✓ ***Computation of the test statistic,*** χ^2 *(obs)*: it is computed by summing the square of the differences of the expected and observed values, where the observed values are the $\pi_i's$ and the expected value is ½.

$$\chi^2\ (obs) = 4 * N * \sum_{i=1}^{M} (\pi i - 1/2)^2 = \mathbf{2.6031} \text{ and } N/2 = 98/2 = \mathbf{49}$$

- ✓ ***Computation of the P-value***: the P-value is computed using the incomplete gamma function as follows: $P - value = \boldsymbol{gammainc}(N/2, \boldsymbol{\chi 2}\ (obs)/2)$, where $\boldsymbol{gammainc}$ is given by

$$\boldsymbol{gammainc}(N/2, \boldsymbol{\chi 2}\ (obs)/2) = \frac{\int_{\chi 2\,(obs)/2}^{\infty} e^{-u} u^{\frac{N}{2}-1} du}{\Gamma(N/2)} \quad \ldots(4.4)$$

Where the gamma function Γ (N/2)=$\int_0^{\infty} e^{-t} t^{N/2-1}\, du$

The probability value for this test is therefore obtained by, $P - value = \boldsymbol{gammainc}(N/2, \boldsymbol{\chi 2}\ (obs)/2) = 0.9998$, where the incomplete gamma function is stated in equation 4.4.

This test also validates the randomness of the chaotic random number generator since the P-value =0.9998 obtained above is greater than the decision rule or level (1%). That's, we can conclude that the sequence is random as the statistics of this test clearly indicate. Under this test, smaller P-values (<0.01) would have indicated a large deviation from the equal proportion of ones and zeros in at least one of the blocks obtained by partitioning the bit sequence output of the chaotic generator.

4.3.3 The Run Test

The main focus of this randomness test is the total number of runs in the chaotic sequence, where a run is an uninterrupted sequence of identical bits. A run of length k consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The objective of the run test is to determine whether the number of runs of ones and zeros of various lengths is as expected (n/2) for a random sequence. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

Given n, the length of the sequence bit string, and the sequence of the bits represented by $\epsilon = \epsilon_1\ \epsilon_2, \ldots, \epsilon_n$, the statistical value, V(obs), is the total number of runs across all n bits. The reference distribution for the test statistic is a $\chi 2$ distribution. The test statistic and concomitant quantities are computed as follows:

- ✓ ***Computation of π:*** π is the ratio of the sum of ones in the given sequence to the bit length of the sequence. $\pi = \frac{sum(1's)}{n} = \frac{12279}{24568} = 0.4998$. As a prerequisite, the value of $|\pi - 1/2| = 0.0002$ must be less than or equal to $|S_n|/\sqrt{n} = 0.0451$ (this value is calculated in the Monobit test). This criterion is satisfied and hence we can carry on with the runs test.
- ✓ ***Computation of test statistic*** $V_n(obs)$: it the sum of runs made from 0 to 1 or vice versa as described in equation 4.5 below.

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1 \qquad \ldots(4.5)$$

Where $r(k)=0$ if $\epsilon_k = \epsilon_{k+1}$, else $r(k)=1$.

The number of switches from 0 to 1 or conversely is found to be $\mathbf{V_n(obs) = 12280}$.

- ✓ ***Computation of P-value***: the probabilistic value is then calculated using the complementary error function as follows:

$$\boldsymbol{P - value = erfc\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right) = erfc\left(\frac{|12280 - 2*24568*0.4998*(1-0.4998)|}{2\sqrt{2*n}*0.4998*(1-0.4998)}\right)}$$

$$\boldsymbol{P - value = erfc(0.0361) = 0.9542}$$

The P-value of this test is found to be $\mathbf{0.9542}$ which is greater than the decision level 0.01. The sequence is therefore random. In this test a large value for V_n ***(obs)*** would have indicated an oscillation in the string which is too fast; a small value would have indicated that the oscillation is

too slow. The oscillation here refers to the switch/change from a one to a zero or vice versa. A fast oscillation occurs when there are a lot of changes between unlike bits (0 and 1). A stream with a slow oscillation has fewer runs than would be.

4.3.4 The Spectral Test

Under this test, the points of focus are the peak heights in the Discrete Fourier Transform of the chaotic sequence. The purpose of this test is to reveal periodicities in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different than 5 %.

ϵ is the sequence of chaotic bits generated by the chaotic generator and n is the bit length of the sequence. The test statistic, d, is the normalized difference between the observed and the expected number of frequency components that are beyond the 95 % threshold; and the reference distribution for the test statistic is the normal distribution. The pertinent statistical quantities and others are computed as follows:

- ✓ ***Conversion to ±1***: The zeros and ones of the chaotic sequence (ϵ) are converted to values of –1 and +1 to produce the sequence $X = X_1, X_2, \ldots, X_n$, where $X_i = 2\,\epsilon_i - 1$.
- ✓ ***Computation of Discrete Fourier transform (DFT) of X***: By computing the DFT of X expressed as ***S = DFT(X),*** a sequence of complex variables is produced which represents periodic components of the sequence of bits at different frequencies as depicted in figure 4.1. Next, the modulus (or magnitude) of S is calculated as ***M = modulus (S´)*** ≡ ***|S'|≡abs(S')***, where S´ is the substring consisting of the first n/2 elements in S, and the modulus function produces a sequence of peak heights.
- ✓ ***Computation of the threshold value, T***: for this test the threshold value, T, is calculated as $T = \sqrt{n * log\frac{1}{0.05}} = \sqrt{\mathbf{24568} * log\frac{1}{0.05}} = 271.2916$, which is equal to the 95 % peak height threshold value. Under an assumption of randomness, 95 % of the values obtained from the test should not exceed T. T is defined in the NIST test Suite.

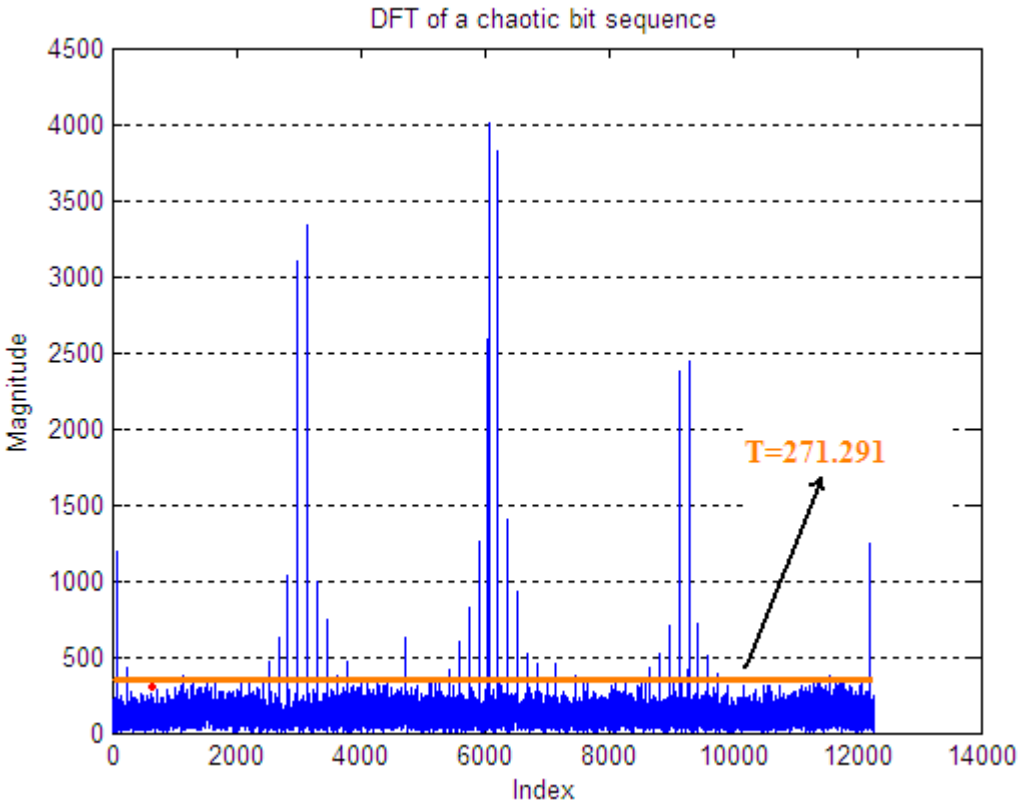


Figure 4.2: DFT of a chaotic sequence

- ✓ ***Computation of N_1 and N_o***: N_1 and N_o are the actual observed and the expected theoretical numbers of peaks, respectively. N_o is the expected theoretical (95 %) number of peaks that are less than T under the assumption of randomness. N_1,the actual observed number of peaks in M that are less than T, is computed using the following algorithm:

```
Variable: count=0: int;
   for j=1:n/2-2 Do
     if(M(1)>M(2)  && j==1)
           peak(count)=M(1);
           count=count+1;
     else if(M(j)<M(j+1)&&M(j+1)>M(j+2))
           peak(count)=M(j+1);
           count=count+1;
     else if(M(n/2)>M(n/2-1)  && j+2==n/2)
           peak(count)=M(n/2);
           count=count+1;
     end;
   end
  N1=peak.length-Na; where Na is number of peaks above T
```

Hence, for the sample chaotic sequence generated, N_o=0.95*0.5*n=11670, and the observed value is N_1= 11639.

- ✓ ***Computation of d***: it is computed as $d = \frac{N1-No}{\sqrt{n*0.95*0.05*0.25}} = \frac{-31}{\sqrt{n*0.95*0.05*0.25}} = -\mathbf{1.8149}$, then $z = \frac{|d|}{\sqrt{2}} = \frac{1.8149}{\sqrt{2}} = \mathbf{1.2833}$
- ✓ **Computation of the P-value**: $P - value = \boldsymbol{erfc(z)} = \boldsymbol{erfc(1.2833)} = \mathbf{0.0695.}$

The P-value= $\mathbf{0.0695}$ obtained in this test verifies that the sequence generated by the chaotic source or generator is random. This is clearly illustrated in figure 4.1. The figure depicts the DFT magnitudes of 24568 bits generated from a chaotic generator which is hereby proved to be satisfactory. The horizontal line, T=271.2916, in the figure is the 95 % confidence boundary. The P −value is found to be $\mathbf{0.0695}$ showing that more than 95% of the peak values are below the confidence boundary proving the hypothesis that the chaotic generator is random

4.3.5 The Linear Complexity Test

The focus of this test is the length of a linear feedback shift register (LFSR). The objective is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by longer LFSRs. An LFSR that is too short implies non-randomness.

Linear-Complexity (M, n, N), where M is the length of bit in a block, n is the length of bit string in the generated sequence $\epsilon = \epsilon_1\ \epsilon_2, \ldots, \epsilon_N$, and N is the number of blocks, is the measure of the linear complexity. The reference distribution for the test statistic is the ***chi-square*** (χ^2) distribution. χ^2 (*obs)* is a measure of how well the observed number of occurrences of fixed length LFSRs matches the expected number of occurrences under an assumption of randomness. The required statistic is described and computed as follows where n=sequence.length=10^6. N & M are selected as per the NIST input recommendations: that's, choose $n \geq 10^6$. The value of M must be in the range $500 \leq M \leq 5000$, and $N \geq 200$ for the χ^2 result to be valid. K, the number of degrees of freedom has been hard coded into the test to be K = 6 by NIST.

- ✓ ***Partitioning***: Partition the n-bit sequence into N independent blocks of M bits, where n = MN. Taking N=1,000, M=n/N=1,000.

✓ ***Determination of the linear complexity, L_i***: the linear complexity L_i of each of the N blocks (i = 1,…,N) in this test is determined using the Berlekamp-Massey algorithm [11, 33] . L_i is the length of the shortest linear feedback shift register sequence that generates all bits in the block i. Within any L_i-bit sequence, some combination of the bits, using the feedback function (XOR/ Sum modulo 2/Odd parity), produces the next bit in the sequence, bit $L_i + 1$. For a chaotic sequence, L_1 is found to be ***502***. 1000 L_i's, one for each block, are computed.

✓ ***Calculation of the Theoretical Mean, μ***: $\mu = \frac{M}{2} + \frac{(9+(-1)^{M+1})}{36} - \frac{(\frac{M}{3}+\frac{2}{9})}{2^M}$=500.2

✓ ***Calculation and Recording of T_i values***: $\boldsymbol{Ti} = (-1)^M * (Li - \mu) + \frac{2}{9}$,

For instance, T_1= $(-1)^{1000}$*(502-500.2)+2/9=2, in such a way 1000 T_i's are computed and recorded in V_0, …, V_6 as follows:

If: $T_i \leq -2.5$, Increment V_o by 1

$-2.5 < T_i \leq -1.5$, increment V_1 by 1

$-1.5 < T_i \leq -0.5$, increment V_2 by 1

$-0.5 < T_i \leq 0.5$, increment V_3 by 1

$0.5 < T_i \leq 1.5$, increment V_4 by 1

$1.5 < T_i \leq 2.5$, increment V_5 by 1

$T_i > 2.5$, increment V_6 by 1

T_1=2, hence it is stored in V_5; that's, V_5 is increased by one. The vector V is therefore found to be V[7] = {13 23 119 493 254 67 18}.

✓ ***Computation of the test statistic, χ^2 (obs)***: $\boldsymbol{\chi 2\ (obs)} = \sum_{i=0}^{K} \frac{(Vi - N\pi i)^2}{N\pi i}$

Where π_0 = 0.010417, π_1 = 0.03125, π_2 = 0.125, π_3 = 0.5, π_4 = 0.25, π_5 = 0.0625, π_6 = 0.020833 (or π[7]= {0.010417 0.03125 0.125 0.5 0.25 0.0625 0.020833}) are probabilities pre-computed by NIST for each of the seven classes described above.

```
Var
    summ, temp: double;
Begin
    Summ:=0;
    for j:=1 to 7 Do
```

```
        Begin
            temp:=power((V(j)- π (j)*N),2);
            summ:=summ+temp/( π (j)*N);
        end
    χ2 (obs): = summ;
End;
```

$$\chi 2\ (obs) = 3.9777$$

✓ ***Computation of the P-value***: $P - value = gammainc(\frac{K}{2}, \frac{\chi 2\ (obs)}{2}) = 0.8030$

In conclusion, the fact that the P-value= 0.8030 is found to be **> 0.01** indicate that the observed frequency counts of T_i stored in the V_i bins are proportional to the computed π_i's as expected. Hence, the chaotic sequence is random.

4.4 Monte Carlo Simulation Test

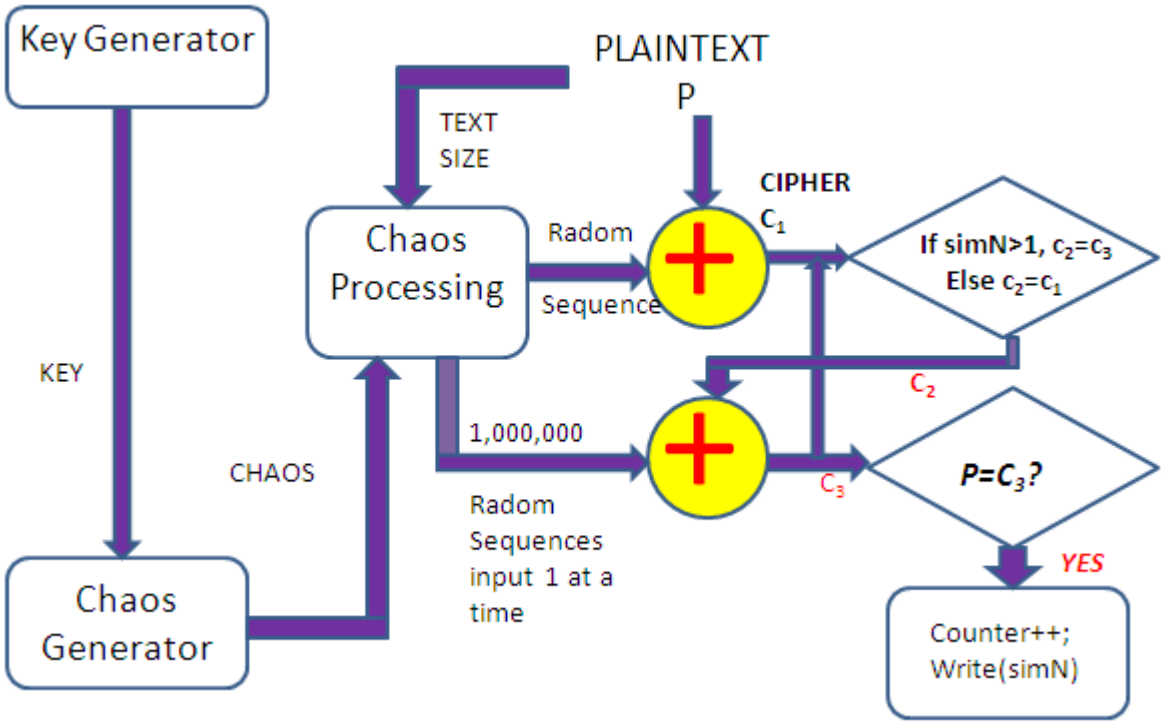


Figure 4.3: Monte Carlo Simulation for simN=1,000,000 runs.

It is a type of simulation test that makes use of randomly generated sequences of bits (or numbers) [31, 32]. The noble purpose of this test is to determine whether the repeated simulation (or re-enciphering) of the ciphertext can end up in the original plaintext or not thereby validating the security of the cipher.

In this thesis work, the Monte Carlo Simulation Test was conducted on an Apple Desktop Computer having a processor of Intel (R) Core (TM) 2 Duo CPU, E8235 @2.86 GHz 2.86 GHz and a RAM of 4.00 GB. The test was performed repeatedly on the cipher to validate its security. It was conducted in such a way that the ciphertext obtained from the chaotic encryption was re-enciphered a number of times by mixing it with a randomly generated chaotic sequence of bits. It was done in two ways: first the ciphertext was simulated (or enciphered) about a million times with a single randomly generated chaotic sequence of bits generated using a 136-bit long key. The output of every simulation was compared with the plaintext to check whether the repeated enciphering of the cipher with a single sequence of chaos can turn out the original plaintext. Secondly, the cipher was re-enciphered a million times with a random sequence generated at every simulation run as depicted in figure 4.3. That's, for one million simulations, a one million different random chaotic sequences were used to re-encipher the ciphertext. Figure 4.3 gives the abstract view of how the Monte Carlo test is simulated in this work where variables C_1, C_2 and C_3 stand for ciphers 1, 2 and 3 respectively, P stands for plaintext, the encircled + symbol represents logical bitwise xor operation, and simN is the number of simulations. In this case, 1,000,000 keys out of the 2^{136} possible combination of keys were used. That's, one combination of the 136-bit key at a time was used for every simulation run, and a total of 1,000,000 keys of different combinations were used for the entire simulation (1,000,000 runs).

The pseudo code or algorithm I devised to perform the Monte Carlo test (of the second type described above) on the chaotic cipher is given in table 4.1.

Table 4.1: Algorithm for Monte Carlo simulation

```
Input
    textBinary, key(136 bits);
Var
    keyTemp, textBinaryTemp,ciphertext:long;
    count,counter,matches,n:int;
    p, pint: double;// probabilities of matchings
Begin
```

```
    count:=0;
    counter:=0;
    textBinaryTemp:=textBinary;
    n:= textBinary.length;
    while(count~=1000000) Do
        Begin
          keyTemp=key.rand(136);
          chaos=generateChaos(keyTemp);
          chaosBinary=processChaos(chaos);
          for i:=0 to n-1 Do
             Begin
                 ciphertext(i):=xor(textBinaryTemp(i),chaosBinary(i));
          end
          textBinaryTemp:=ciphertext;
          if(textBinary==cipherText)
            matches(count):=count;//at which simN does matching occur?
            Counter++;
          end;
         pint(count):=counter/count;//intermediate probabilities of matching
         count=count+1;
    end;
    p:=(counter)/(count-1;// overall probability of matching
    display("count" "pint" "matches" "p");
end;
```

Table 4.2 illustrates the partial simulation results of the Monte Carlo Simulation Test performed in this thesis work for a simulation number, simN=1, 000, 000. In table 4.2,'-' means doesn't occur.

Table 4.2: Results of Monte Carlo simulation Test

Count	Intermediate probabilities, p_{int}	Matching occurs at simN	Overall probability, p
1	0	-	0
100,000	0	-	
200, 000	0	-	
300, 000	0	-	
…	…	…	
1, 000, 000	0	-	

As indicated in table 4.2, the intermediate probabilities for the 1, 000, 000 simulations and the overall probability were found to be zero. The interpretation of the results is that the chance of obtaining the plaintext from a corresponding ciphertext by performing the Monte Carlo Simulation using both the same and different chaotic sequences on the designed chaotic encryption algorithm is

zero. It is found to be difficult to recover the plaintext by simply repeatedly re-enciphering the cipher for a simulation number of 1, 000, 000. To successfully recover the plaintext from the ciphertext by doing so, the Monte Carlo Simulation Test has to be increased to an exhaustive chaotic sequence search analysis. However, exhaustive analysis is not feasible because the minimum chaotic sequence allowed in this work is 128 bits. Consequently, it is very much infeasible and impractical to conduct and exhaustive search for an input of length 128 bits or greater with the existing computing power.

Chapter Five

PERFORMANCE ANALYSIS OF THE CHAOTIC ENCRYPTION

5.1 Overview

Performance evaluation is an art. That's, just like a work of art, successful evaluation cannot be produced mechanically. Every evaluation requires an intimate knowledge of the system being modeled and a careful selection of the methodology, workload, and tools. Defining the real problem and converting it to a form in which established tools and techniques can be used and where time and other constraints can be met is a major part of the analyst's art. Hence, like an artist, each analyst has a unique style. Given the same problem, two analysts may choose different performance metrics and evaluation methodologies. In fact, given the same data, two analysts may interpret them differently [31]. Putting it in simple terms, performance refers to how well a given system does a piece of work or an activity as measured in terms of predefined touchstones or metrics. In this work, relevant metrics are identified, performance of the chaotic algorithm is measured, and eventually, the chaotic encryption algorithm is compared with existing one public key, RSA and one secret key, AES, encryption algorithms.

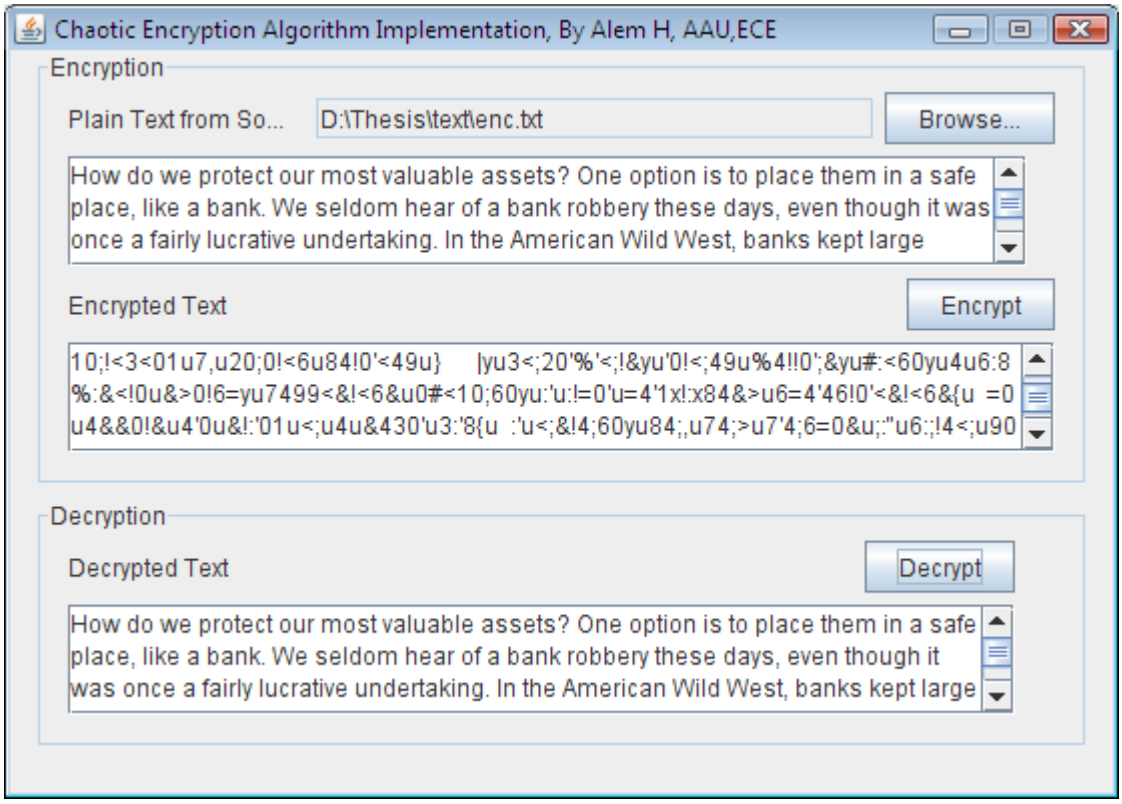
5.2 Metrics and Performance Evaluation

To study the performance of the designed algorithm, a set of pertinent performance metrics must have been identified first. One approach to prepare the set is to list the services offered by the designed system [31]. For each service request made to the designed system, there are several possible outcomes. Generally, these outcomes may be put into a number of categories. For instance, the designed crypto-system may perform the enciphering and deciphering processes correctly and fast, incorrectly, refuse to perform the service, or produce a secure cipher, etc. These are related to time-rate-resource metrics for successful performance, which are also called responsiveness, productivity, and utilization metrics, respectively. In this work, five pertinent metrics are used to evaluate the performance of the chaotic encryption. The metrics include encryption time, power consumption, encryption throughput, CPU time and cipher size. Then the

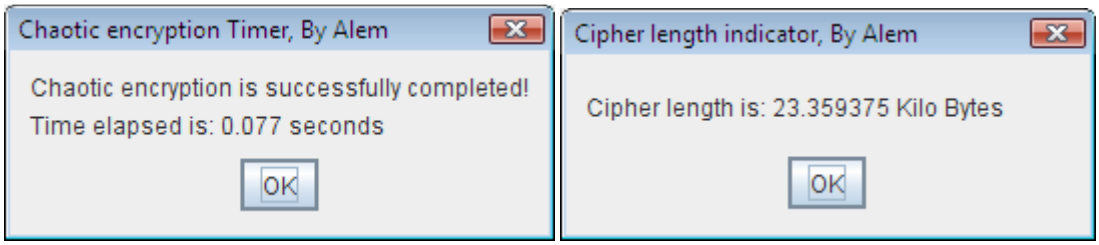
experiment was conducted and performance results were collected using a laptop having processor of Intel (R) Pentium (R) Dual CPU T2370 @ 1.73GHz, 1.73GHz, and a RAM of 1GB.

5.2.1 Encryption Time

It is the time interval between arrival of a plaintext and its successful enciphering. In most encryption algorithms, the encryption time is dependent on the computational complexity of the algorithm, key length, and the size of the plaintext to be encrypted



(a) Multi-Scroll Chaotic Crypto System



(b) Encryption timer (c) Measure of cipher length

Figure 5.1: Measurement of Encryption Time and text size

Here, in this thesis work, a number of encryption times for various plaintext sizes and key length are collected and analyzed as follows. Figure 5.1 illustrates how the encryption time and text size are measured.

✓ **The Effect of Changing the Key Length on the Encryption Time**

Unlike to other encryption algorithms, the key length doesn't affect significantly the computation time of the chaotic algorithm designed in this work. This is due to the fact that the key is used only to calculate the initial conditions of the system used to generate chaos. The initial conditions are calculated in way described in equations 3.28.

✓ **The Effect of Changing the plaintext size on the Encryption Time**

Various data sizes ranging from 7.84 kb to 500 Kb are enciphered, and their respective encryption times are collected in the same way demonstrated in figure 5.1. Table 5.1 delineates the various data enciphered and their corresponding time consumptions.

Table 5.1: Data sizes and their encryption times

Data size in Kb	7.84	15.87	23.36	31.20	39.30	55.16	73.87	79.16	158.30	457.16
Enc Time in sec	0.031	0.047	0.078	0.094	0.156	0.250	0.437	0.515	1.232	3.823

To show how the encryption time varies with change in the data size, the corresponding values in table 5.1 are graphed into a graphical representation portrayed in figure 5.2 depicts that the enciphering time increases as the data size to be enciphered is increased. There is a direct relationship between data size and the enciphering time.

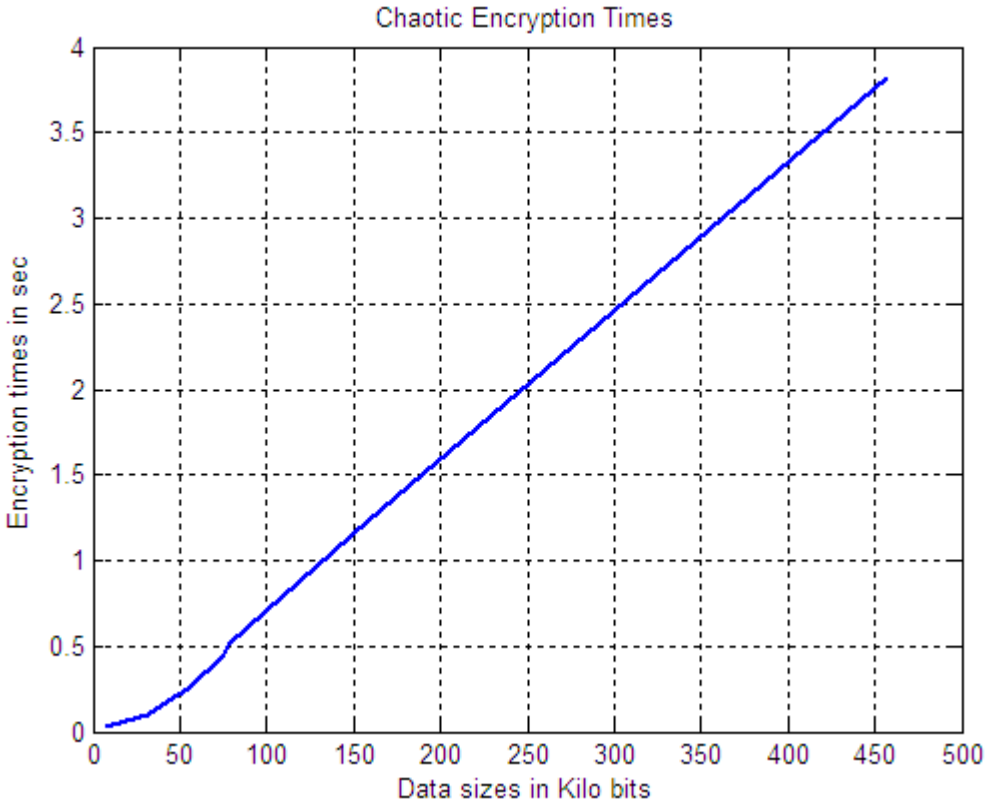


Figure 5.2: Data size versus Encryption time

5.2.2 Encryption Throughput

It is the measure of the number of bytes of ciphertext completed (enciphered) during an observation period (enciphering time). Mathematically, it is depicted as follows:

$$Encryption\ Throughput, Xe = \frac{\text{number of completions in bytes}}{Encryption\ Time\ (s)} \qquad \ldots 5.1$$

✓ **The Effect of Changing the plaintext size on the Encryption Throughput**

Table 5.2 illustrates how the encryption throughput is affected as the data size increases.

Table 5.2: Data sizes and their throughputs

Data size in bytes	1004	2031	2990	3994	5030	7060	9455	10132	20262	58516
Thruput	32372	43220	38334	42485	32246	28242	21637	19675	16446	15306

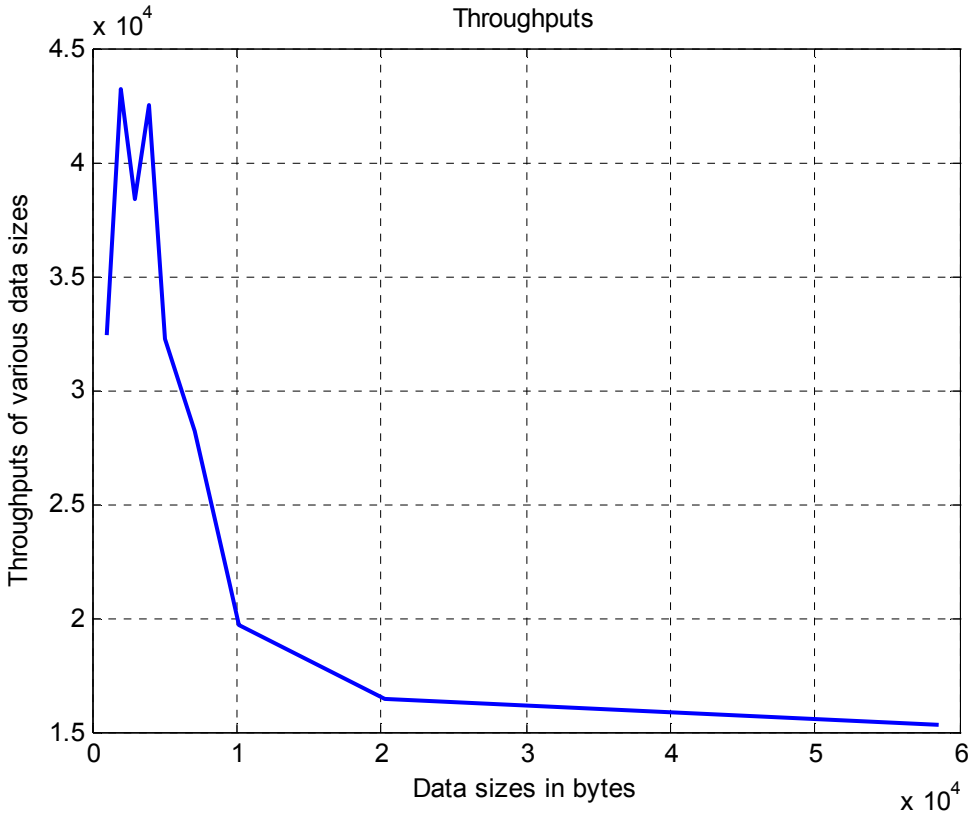


Figure 5.3: Encryption throughput Vs Data sizes

The tabular values are then portrayed into a graphical representation, as depicted in figure 5.3. The graph shows that for initially small size data the throughput is not affected; rather it increases with increase in data size. However, once the data size gets large enough, further increase in the data size keeps on diminishing the encryption throughput as depicted in the graph of figure 5.3.

5.2.3 Power Consumption

Now-a-days, there is a limitation in battery power sources. Such technologies as CPU and memory are growing faster, and so is their need for power. However, battery technology is increasing at a much slower rate, forming a battery gap. Because of this, battery capacity plays a major role in the usability of devices and algorithms [3]. Hence, it is worthwhile to analyze the power consumption of the designed chaotic algorithm.

For computation of the energy cost of encryption, I use the same techniques described in [3]. I present a basic cost of encryption represented by the product of the total number of clock cycles taken by the encryption and the average current drawn by each CPU clock cycle. The basic encryption cost is in unit of ampere-cycle. To calculate the total energy cost, we divide the ampere-cycles by the clock frequency in cycles/second of a processor; we obtain the energy cost of encryption in ampere-seconds. Then, we multiply the ampere-seconds with the processor's operating voltage, and we obtain the energy cost in Joule. That's, by using the cycles, the operating voltage of the CPU, and the average current drawn for each cycle, we can calculate the energy consumption of the cryptographic algorithm. Then, the amount of energy consumed by the chaotic algorithm C to achieve its goal (encryption or decryption) is given by:

$$\boldsymbol{E = V cc * I * T} \text{ joules} \quad \text{---(5.2)}$$

Where for a given hardware Vcc is fixed. The encryption time, T, is considered the time that an encryption algorithm takes to produce a cipher text from a plaintext, and I is the average current consumed per CPU cycle.

In this work, the experiment was conducted and performance results were collected using a laptop with Pentium Dual1.73GHz CPU. Therefore, the approximate average current consumed is taken to be 100mA, and the CPU voltage is Vcc=1.25 volts as obtained from Inter processor manuals. The power consumption performance analysis results for various data sizes are collected based on these current and voltage ratings.

- ✓ **The Effect of changing the plaintext size on the Power Consumption.**

Table 4.3 illustrates the effect data size increase on the power consumed during encryption process. The table depicts that the energy consumed increases as the data size to be enciphered increases for energy consumed is dependent on the duration of time a process runs.

Table 5.3: Data sizes and their throughputs

Data size In Kb	7.84	15.87	23.36	31.20	39.30	55.16	73.87	79.16	158.30	457.16
E in Joule	0.004	0.006	0.010	0.012	0.020	0.031	0.055	0.064	0.154	0.478

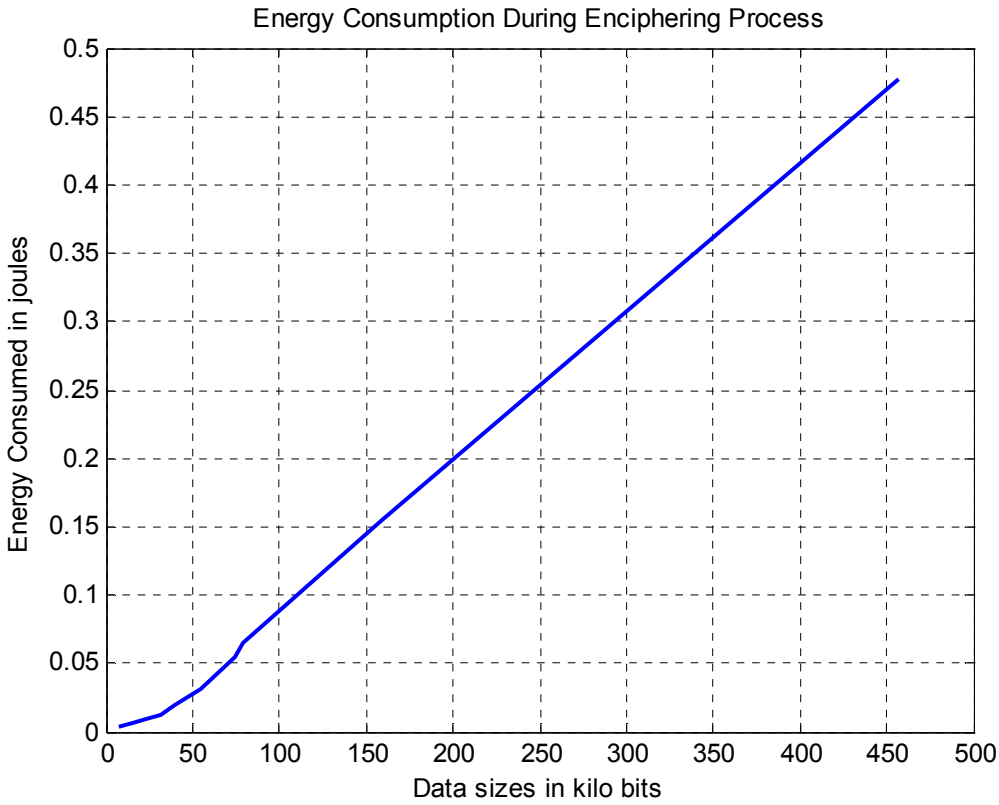


Figure 5.4: Data Size Vs Energy consumption for Enciphering Process

The data in table 5.3 are graphically depicted in figure 5.4 clearly showing the variation of energy consumption with different data sizes. It can also be inferred from the graph that it is similar to the data size versus encryption time graph in figure 5.2 which implies that the energy consumed during an enciphering process is directly proportional to the encryption time taken.

5.2.4 CPU time

The CPU time is the time during which the CPU is committed only to the particular process execution. It reflects the load of the CPU. The more CPU time is used in the encryption process, the higher is the load of the CPU. In this work, the CPU time is calculated using the technique described in [31] as follows:

$$CPU\ busy\ time, Tcpu = \frac{\text{CPU utilization}}{Observation\ Period} \quad \ldots (5.3)$$

Where the observation period is equal to the enciphering time and CPU utilization for the encryption process is collected using process explorer.

It is found out that the longer the encryption time is the busier the CPU becomes. That's, if the time required to encipher a certain text is longer, the CPU load (or busy time) is proportionally higher.

5.2.5 Cipher Size

One of the most integral Shannon's Characteristics of "Good" Ciphers is that the size of the enciphered text should be no larger than the plaintext of the original message [2].

As it is the case with other secret key encryption algorithms, in this work the size of the plaintext and the ciphertext are found to be the same fulfilling the Shannon's size Characteristics of "Good" Ciphers. The two merged pop-up message boxes portrayed in figure 5.5 verify that the plaintext and ciphertext, in this algorithm, are of the same size.

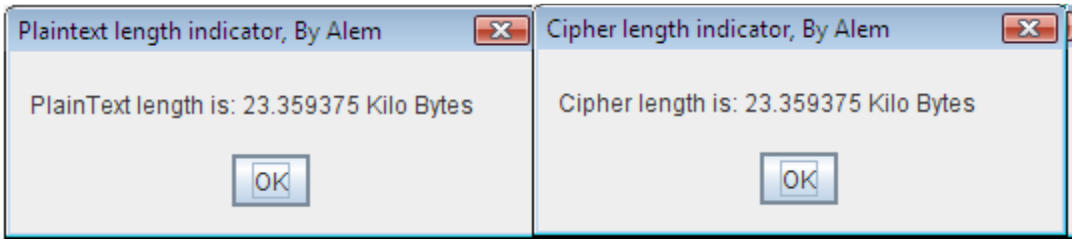


Figure 5.5: Length measure of a clear and cipher texts.

5.3 Comparison with AES and RSA

The performance of the designed algorithm is compared with the popular current-in-use secret key encryption algorithm, AES, and with a public key encryption, RSA. The data sets, used in the chaotic encryption, are encrypted using both AES and RSA, and their performance is evaluated for the same metrics used above. Here, while analyzing the performance of AES and RSA, only secure key lengths are used, 128 bits for AES and 1024 bits for RSA.

✓ Encryption times

Encryption times for same set of various data sizes, used to analyze the performance of the chaotic enciphering, are collected as depicted in tables 5.4 and 5.5.

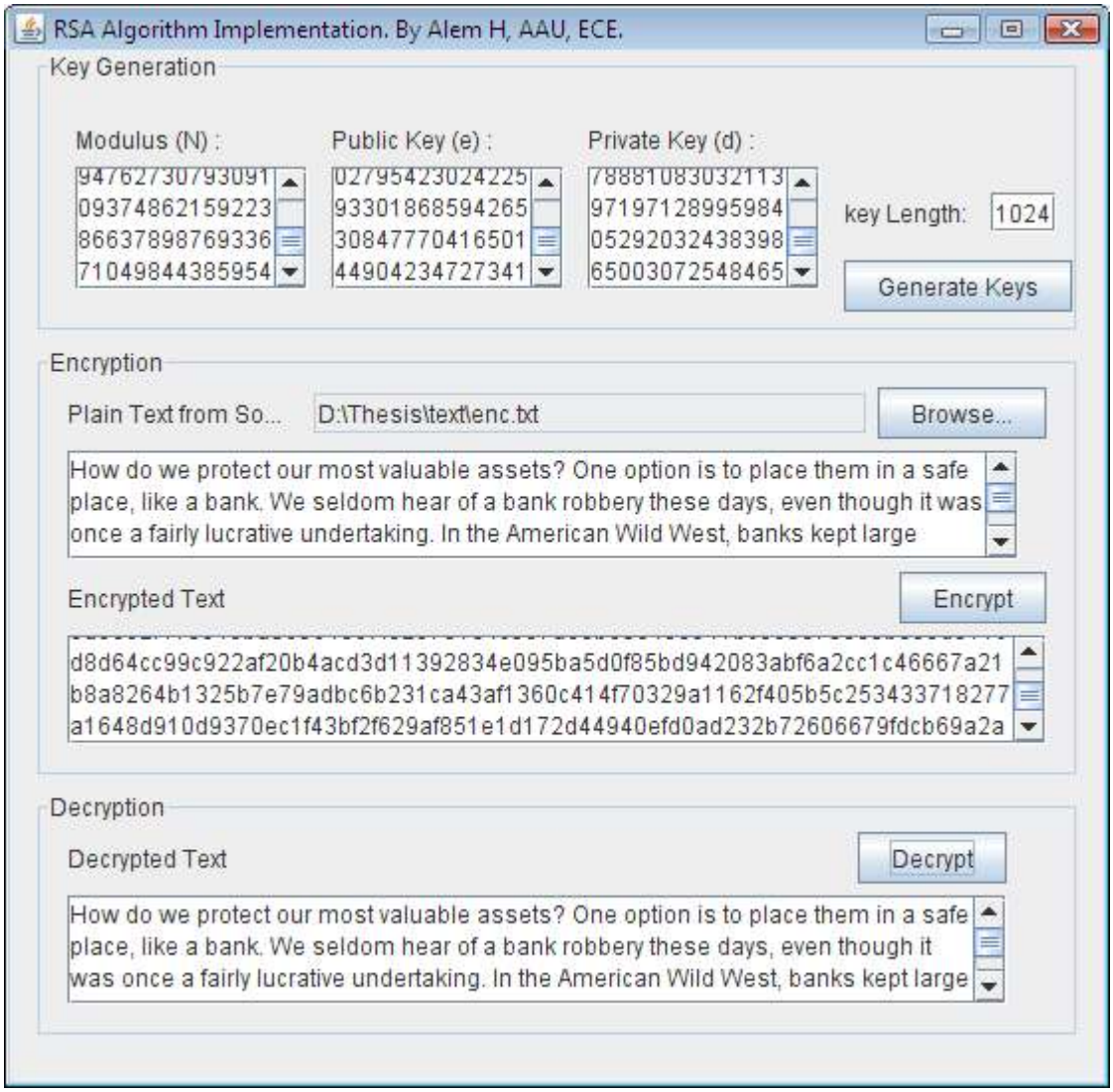


Figure 5.6: RSA Crypto System

Figure 5.7: RSA Encryption and Decryption Timers

The encryption and decryption times in table 5.4 for RSA are collected using the crypto scheme and timers depicted in figures 5.6 and 5.7 respectively.

Table 5.4: Encryption &Decryption times of RSA

Data size in Kb	7.84	15.87	23.36	31.20	39.30	55.16	73.87	79.16	158.30	457.16
Enc Time in sec	0.506	0.978	1.438	1.852	2.326	3.243	4.248	4.596	9.355	32.554
Dec Time in sec	1.793	3.514	5.124	6.818	8.494	11.90	15.72	16.99	33.78	97.29

As depicted in table 5.4, the decryption time of RSA algorithm is higher than the encryption time. This is due the fact that the enciphering comprises modular computation of $(plaintext)^{PubKey}$ ***mode n***, whereas the deciphering process involves the modular computation given as $(plaintext)^{pubkeyprivkey}$ ***mode n.*** PrivateKey *PubKey is much larger than PubKey, and hence requires higher computational time.

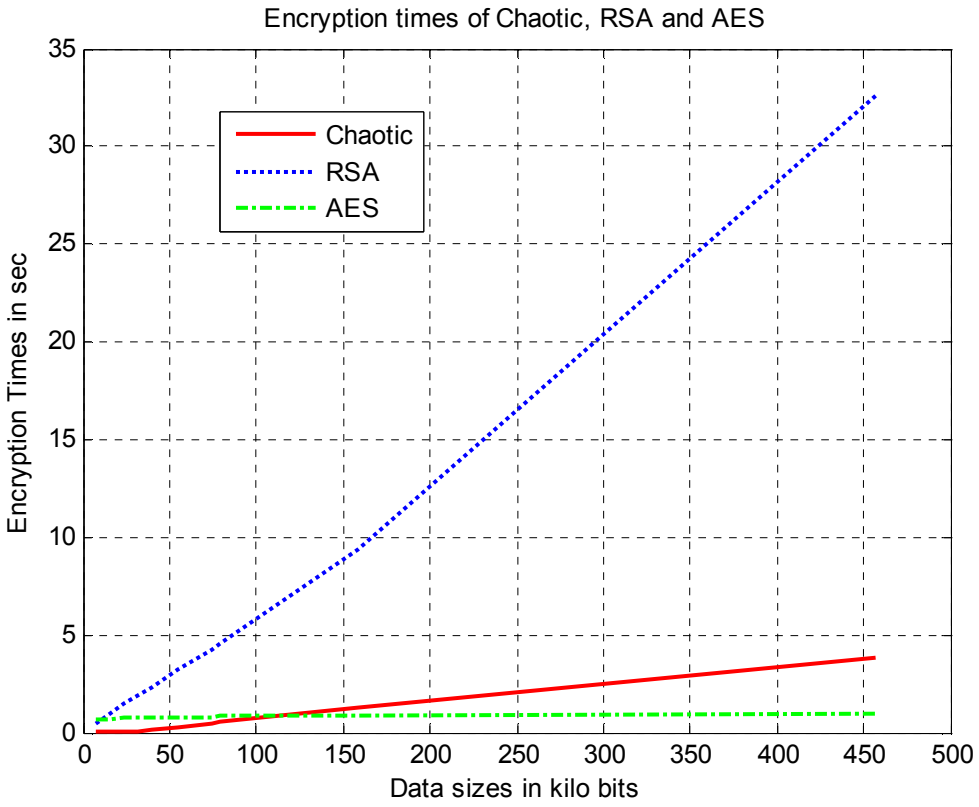


Figure 5.8: Enciphering times of Chaotic, RSA, and AES

Likewise, encryption times for AES, for the various data sizes enciphered, are collected as illustrated in table 5.5. The deciphering times for AES are more or less close to the enciphering times.

Table 5.5: Encryption times of AES

Data size in Kb	7.84	15.87	23.36	31.20	39.30	55.16	73.87	79.16	158.30	457.16
Enc Time in sec	0.669	0.692	0.719	0.731	0.735	0.807	0.813	0.821	0.829	0.987

Figure 5.8 shows that the chaotic encryption is much faster than RSA algorithm for any data size. It has better time performance than AES, too, but for data sizes less than or equal to 125Kb.

✓ Encryption Throughput

Table 5.6 illustrates how the RSA encryption throughput is affected as the data size increases.

Table 5.6: Data sizes and RSA throughputs

Data size In bytes	1004	2031	2990	3994	5030	7060	9455	10132	20262	58516
Thruput	1984	2077	2079	2157	2163	2177	2226	2205	2166	`798

Table 5.7 illustrates how the AES encryption throughput is affected as the data size increases.

Table 5.7: Data sizes and AES throughputs

Data size In bytes	1004	2031	2990	3994	5030	7060	9455	10132	20262	58516
Thruput	1501	2935	4159	5464	6844	8748	11630	12341	24441	59287

Figure 5.9 illustrates that the throughput performance of the chaotic encryption is very much high for smaller size of data, and it keeps on decreasing as the data size increases. It has higher performance than AES for smaller data sizes, but it is very much superior to RSA for any data size.

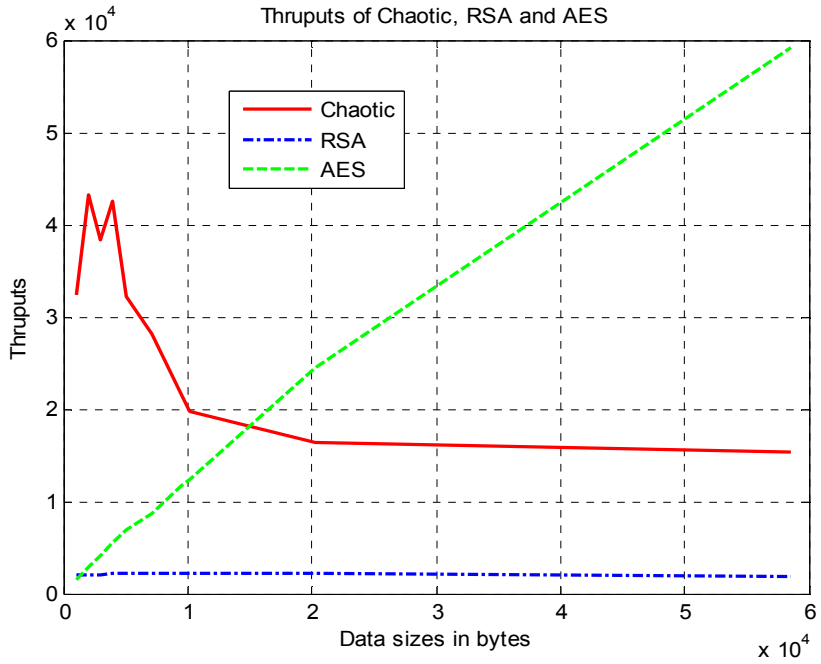


Figure 5.9: Throughputs of Chaotic, RSA, and AES

✓ Encryption power consumption

The power consumptions of the Multi-Scroll Chaotic, RSA and AES are depicted in figure 5.10.

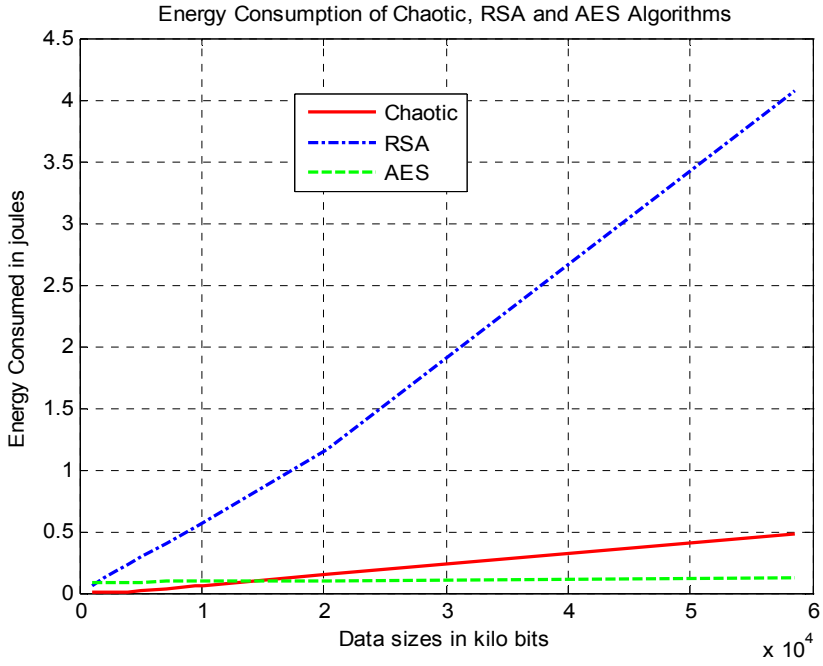


Figure 5.10: Power Consumption of Chaotic, RSA, and AES Algorithms.

The figure demonstrates that the designed algorithm has less power consumption than AES for relatively small data sizes, and much better performance than the RSA for any data size. The graphs in this figure are similar to the ones on figure 5.8 proving that the energy consumed by an enciphering process is proportional to the time consumed by that process.

✓ CPU time

CPU time metrics also produces similar results because it is, in one way or the other affected by the enciphering time. Its graph has more or less the same shape as that of encryption time.

✓ Memory Cost

In secret key encryption, the plaintext and cipher text are of the same size. Similarly, the plaintext and ciphertext of the designed algorithm are of the same size. But, considering the RSA, the cipher size is greater than the size of the input plaintext. As a result, the cipher of an RSA algorithm occupies more memory as compared to that of chaotic and AES.

Table 5.8: memory cost of algorithms

Algorithm	Chaotic	AES	RSA
Plaintext and ciphertext sizes	Same (=)	Same (=)	Cipher size> text size

✓ Security of Multi-Scroll Chaotic, AES and RSA algorithms

The security of the chaotic encryption scheme lies on the difficulty of obtaining the exact key combination from amongst the very large set of possible combinations of the key due to limitations in the computational power of today's computers. The complexity class of the input key of this algorithm is NP-complete in that all the NP (Non-deterministic Polynomial Time) problems have polynomial solutions. That's, the key length can be of any value greater than 128 bits; then, to perform exhaustive key search analysis, one has to try all combinations of every bit sequence length starting from 128 up to the length of the key used. It is described below mathematically. Let the key length be L, where L>128 bits, and k=L-128 bits, then the number of operations required to obtain the deciphering key via exhaustive key search analysis is

$$\text{Number of operations } = 2^{128} + 2^{129} + 2^{130} + \cdots + 2^{L} \quad \text{---(5.1)}$$

$$\text{Number of operations } = 2^{128} * (2^{0} + 2^{1} + 2^{2} \ldots + 2^{k}) \quad \text{--- (5.2)}$$

The power series, whose base is 2, in equation 5.2 can be summed as

$$\left(2^0 + 2^1 + 2^2 \ldots + 2^k\right) = 2^{k+1} - 1 \quad \text{--- (5.3)}$$

Therefore, to successfully search the deciphering key using brute force analysis, one needs to make $\boldsymbol{2^{128}(2^{k+1}-1)}$ operations which are impractical and infeasible with the current computing power of computers. Table 4.9 shows various key sizes and the time required to search the correct deciphering key using as many computers as the number of atoms contained by the planet Earth. Number of atoms of Earth, $N_{ae}=3.6x10^{51}$atoms.

Table 5.9: Time required to break some chaotic keys.

Key size in bits	Number of keys	Time required at $3.6x10^{51}$ decryption/second
128	$2^{128}=10^{39}$	0.28 ps (Pico seconds)
136	$2^{136}*(2^9-1)=10^{44}$	28 ns (nano seconds)
256	$2^{256}*(2^{128}-1)=10^{116}$	$2.8x10^{57}$ years
512	$2^{521}*(2^{384}-1)=10^{270}$	$2.8x10^{218}$ years

As illustrated in table 5.9, trying to obtain the deciphering key via brute force analysis is really impractical and infeasible, so the algorithm is very secure. Besides, the cipher is less prone to statistical attacks that can be perpetrated by malicious parties because the chaotic sequence with which it is mixed is random and unpredictable. Considering the key security, this algorithm is much superior to both AES and RSA in that the key can be of any length greater than 128 bits without significantly affecting the computational time.

Considering AES, the AES in use today has a fixed key length of 128 bits, and 10 rounds for key scheduling and mixing. The number of operations required for exhaustive key search analysis is 2^{128}. The AES key can be increased to such longer values as 192, 256, etc which are strictly multiples of 64, and thereby increase the key security. However, for every 64 bits increase in key size the number of rounds increases by 2 which significantly increases the computational time. What's more, when Monte Carlo Simulation was performed on the first four rounds of the AES, it produces matches.

The security of the RSA algorithm rests on the difficulty of factoring the modulus, n, of the scheme. If it is factored using the best known, and most efficient factoring algorithm known as Brent-Pollard, it takes $\frac{e^{\sqrt{2*lnp*\ln(lnp)}}}{lnp}$ operations on number *n* whose largest prime factor is p [6]. The number of operations required to factor the current in use 1024 bits modulus is e^{63} (or 2^{90}). In terms of operations required, it is equivalent to an 80-bit key symmetric encryption algorithm. The security of RSA can be enhanced by increasing the key size, but it becomes computationally extortionate. Table 5.10 summarizes the security comparison of the Multi-Scroll chaotic, AES and RSA encryption algorithms.

Table 5.10: summary of security comparison.

Algorithm	Number of operations required	Examples
Chaotic	$2^{128}(2^{k+1}-1)$	Key=136 bits→2^{145} operations
AES	2^n	Key=128 bits→2^{128} operations
RSA	$\frac{e^{\sqrt{2*lnp*\ln(lnp)}}}{lnp}$	N=1024 bits →2^{90} operations

In summary, considering all the metrics set to measure the performance of the designed algorithm, the designed chaotic enciphering algorithm has better performance for small size data than the AES. But it is by far better than RSA for any data size. What's more, the chaotic algorithm's computational time is not significantly affected by an increase in the key length unlike to both AES and RSA. The key length can be of any value greater than 128 bits unlike to AES where the key length must be multiple of 64. Considering security, the chaotic encryption algorithm is superior to both AES and RSA.

Chapter Six

CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusion

At present, Internet and network applications are growing very fast, so the needs to protect such applications are increased. Encryption algorithms play an immense role in information security systems. My thesis work has presented the optimization of multi-scroll chaotic attractors for text encryption and its performance analysis. All essential and collateral parameters are systematically determined as to satisfy the specific cryptographic security requirements. The algorithm solves at least some of the drawbacks suffered by existing cryptographic algorithms such as AES and RSA.

In this work, a multi-scroll chaotic enciphering algorithm is fully described, and validated via functional and randomness tests. Then, appropriate metrics for performance measurements are identified; the performance of the algorithm was measured and compared with such existing cryptographic algorithms as RSA and AES. The test results show that the designed algorithm works as required; that's, the data enciphered by the enciphering process is fully recovered by the deciphering process. The tests and security analyses prove that the cipher is not prone to selected cipher or statistical attacks and the key is secure.

What's more, another point to conclude about is that performance of the chaotic encryption algorithm is by far better than the RSA. That's, it has less encryption time, less power consumption, and higher throughput than the RSA. The size of the plaintext and ciphertext is also the same in the chaotic algorithm which is not the case in RSA. RSA cipher is longer than the plaintext causing more resource consumptions and congestion in memory and bandwidth. Comparing it with AES, the present day most popular encryption algorithm, it has better performance only for relatively smaller data sizes. The chaotic encryption has other advantages over RSA and AES in that it is key length independent; that's, the key length can be made longer

without significantly affecting the computational time. Besides, this algorithm is safer than both RSA and AES in that there is no known method of attack devised so far.

6.2 Recommendation

6.2.1 Application and Contribution.

Now-a-days, there is a great tendency towards the use of network, particularly the Internet, for commercial, medical, personal, etc, information transaction or exchange. At the same time, network or Internet users always want to have the assurance that every communication or information exchange they make is secure and private. Therefore, the work done in this thesis can play a good role in providing security and privacy to both computer and network users in our country.

In our country efforts have already been made to create a secure network by INSA, the Information Network Agency. Then, this thesis work can have a great contribution in that it can be applied in such areas as Banking systems to make online payment and transactions via e-cards possible, and in the ETC, Ethiopian Telecommunication Corporation, for secure and private communication amongst customers. It can also be used for government and military applications where there is a need to protect classified information.

6.2.2 Further works

Usually cryptographic algorithms are standardized as information processing standards only after passing a series of cryptanalytical tests conducted by high caliber and competent cryptographers and cryptanalysts for a long time in highly sophisticated crypto labs. I therefore recommend people who have full access to well setup crypto labs to perform standard cryptanalytical tests on this cryptographic algorithm.

APPENDIX

Appendix A

Matlab Code for the entire Multi-Scroll Chaotic Crypto System

```
%-------------------------------------------------------------------------
%@uthor: Alem Haddush Fitwi, M.Sc Student of Computer Engineering
%Chaotic Encryption using a shared image as a key
%AAU, FOT, ECE, May, 2010
%-------------------------------------------------------------------------
%------------Constants----------
a=0.0049;
var=5.98732516340;
keyL=136;
k=keyL;
sn=5; %scroll number
mult=69621;
m=2147483647;
Xo=1;%seed-1
X1=1000;%seed-2
X2=7;%seed-3
X3=1100;%seed-4
%-------------------------------------------------------------------------
%Step-1: Accessing the shared image
%_____________________________
keyImage=imread('key.jpg');
%-------------------------------------------------------------------------
%Step-2: Image processing
%_________________________
%keyImage size=261x326
keyGray=rgb2gray(keyImage);%convert RGB image to grayscale
keyPixel=keyGray;%grabbing pixel values gray image
%-------------------------------------------------------------------------
%Step-3: Key Generation
%_____________________
%step-3.1: Key Extraction

%----------PRNG LCG---------
for i=1:17
temp1=mod(mult*Xo,m);
indexX1(i)=mod(temp1,261);
temp2=mod(mult*X1,m);
indexX2(i)=mod(temp2,326);
Xo=indexX1(i);
X1=indexX2(i);
  %--------
temp3=mod(mult*X2,m);
indexY1(i)=mod(temp3,261);
temp4=mod(mult*X3,m);
indexY2(i)=mod(temp3,326);
X2=indexY1(i);
X3=indexY2(i);
end
%---------------------------
```

```
countX=1;
countY=1;
n=1;
while(n<=17)

        keyExtX(countX)=keyPixel(indexX1(n),indexX2(n));
        countX=countX+1;
        keyExtY(countY)=keyPixel(indexY1(n),indexY2(n));
        countY=countY+1;
        n=n+1;
end
%----------------------------
fid = fopen('keyExt.text','wb');
fwrite(fid,keyExtX(1,countX-1),'integer*1');
fclose(fid);
%----------------------------
%step-3.2: Convert the extracted key to binary values
countX=countX-1;
countY=countY-1;
%Xbinary
bitCountX=1;
for ii=1:countX
    cc1=1;
    nnx=keyExtX(ii);
    while(cc1<9)
        Qx=nnx/2;
        Rx=mod(nnx,2);
        nnx=Qx-Rx/2;
        Px(cc1)=Rx;
        cc1=cc1+1;
    end

    for k=1:8
        keyBinaryX(bitCountX)=Px(cc1-k);
        bitCountX=bitCountX+1;
    end
end
bitCountX=bitCountX-1;
%Ybinary
bitCountY=1;
for jj=1:countY
    cc2=1;
    nny=keyExtY(jj);
    while(cc2<9)
        Qy=nny/2;
        Ry=mod(nny,2);
        nny=Qy-Ry/2;
        Py(cc2)=Ry;
        cc2=cc2+1;
    end

    for kk=1:8
        keyBinaryY(bitCountY)=Py(cc2-kk);
        bitCountY=bitCountY+1;
    end
end
```

```
bitCountY=bitCountY-1;
%step-3.2: Key substitution using S-boxes of order GF(2^5)
S1=[14 21 7 28 18 16 27 1 3 17 8 13 25 4 5 22 30 15 19 24 31 6 20 11 2 0 29
10 12 9 23 26;
    24 16 18 0 13 11 19 27 5 12 20 1 7 26 4 8 30 15 21 28 23 31 9 17 29 10 2
22 6 25 3 14;
    28 19 23 1 25 24 27 21 30 18 15 26 20 13 2 14 29 0 7 5 8 16 17 9 31 11 4
6 10 12 3 22;
    6 15 18 3 0 27 23 11 12 25 20 4 31 5 9 7 14 1 24 17 21 30 16 10 8 2 13 29
19 26 28 22];

S2=[26 2 31 8 22 15 25 9 18 12 29 24 21 3 20 11 14 0 13 27 30 19 16 4 17 6 1
10 28 7 5 23;
    1 26 3 0 13 4 25 6 14 18 15 24 21 9 12 19 30 5 11 27 28 20 23 22 2 7 29
16 10 31 8 17;
    14 22 2 10 4 16 8 21 18 27 19 17 28 15 11 9 20 24 29 0 6 23 12 1 3 13 7
26 31 5 30 25;
    15 1 23 11 21 25 30 12 8 18 4 3 9 29 22 14 0 5 24 19 26 13 16 28 10 31 20
7 6 27 2 17];

S3=[13 11 27 10 16 12 30 5 29 17 31 8 24 3 6 4 0 7 20 2 26 22 15 23 19 21 25
28 18 14 1 9;
    10 4 21 20 31 1 18 3 29 28 12 19 5 15 22 14 17 7 30 24 13 11 2 0 26 9 23
25 8 16 27 6;
    10 20 21 23 24 26 1 31 27 17 2 0 16 25 22 29 9 14 11 3 19 5 28 6 7 30 15
13 4 18 8 12;
    2 16 21 17 27 7 15 22 28 8 18 23 24 25 12 10 26 14 0 9 31 19 3 29 20 6 5
4 1 11 13 30];

S4=[8 24 6 23 26 28 1 0 5 4 11 15 18 19 31 7 29 17 2 21 14 10 30 20 16 27 13
25 12 22 3 9;
    28 25 2 29 1 12 27 17 3 7 11 6 5 26 13 31 14 21 0 23 9 24 19 10 16 20 18
15 22 4 8 30;
    5 10 31 28 26 25 13 0 18 19 23 22 3 30 4 16 7 8 15 6 2 27 24 1 21 11 29
20 9 12 17 14;
    22 31 2 21 27 23 26 0 24 3 25 28 4 7 18 29 14 9 11 5 17 13 8 12 19 15 1
30 16 20 6 10];

S5=[18 16 9 23 10 19 8 28 14 6 17 13 2 1 12 25 30 29 24 31 7 5 27 3 15 11 0
21 22 26 20 4;
    17 23 28 15 0 11 21 25 19 30 7 13 22 27 18 24 16 31 8 12 10 6 3 5 1 29 9
2 4 20 14 26;
    11 24 20 9 28 6 7 23 19 21 1 18 27 0 30 26 2 22 10 31 8 13 29 3 25 12 14
17 15 5 16 4;
    10 28 0 21 23 18 7 27 12 14 8 11 29 26 22 4 31 1 3 6 2 19 9 17 20 25 15 5
30 24 16 13];

S6=[20 2 11 29 30 18 7 22 1 13 9 23 25 6 5 14 15 28 27 17 8 4 31 24 26 12 21
0 10 3 19 16;
    21 15 14 0 8 13 4 9 1 23 12 25 20 22 17 16 28 10 30 29 24 19 6 7 26 27 5
2 31 3 11 18;
    5 28 10 6 15 24 30 7 3 12 0 29 19 25 13 1 22 8 26 27 14 23 2 18 4 31 9 11
17 16 20 21;
    25 4 8 15 23 14 7 5 24 18 1 17 27 26 31 22 6 21 30 2 16 3 10 13 9 29 19
12 11 20 0 28];
```

```
S7=[13 28 7 4 24 19 3 2 31 14 29 0 23 1 9 11 20 17 15 16 22 6 8 21 25 18 26
12 10 5 27 30;
    25 24 1 2 30 8 20 27 11 23 6 12 0 28 19 26 15 3 5 21 9 17 31 16 18 7 29
13 10 14 4 22;
    11 10 4 24 20 5 18 30 17 8 23 15 25 27 19 31 2 7 3 14 28 12 9 16 1 0 21 6
22 13 26 29;
    21 28 5 24 12 31 2 3 11 17 30 7 0 10 14 1 4 23 16 26 29 27 19 22 18 13 20
6 9 8 15 25];

%-------------------------------------------------------------------------
%Padding
mm=mod(8*keyL,49);
for im=1:49-mm
    keyBinaryX(keyL*8+im)=0;
    keyBinaryY(keyL*8+im)=0;
end
%-------------------------------------------------------------------------
ctrl=1;
cr=1;
cc4=1;
count1=max(size(keyBinaryX))/49;
while(ctrl~=(count1+1))
            %while begins here
            for i2=1:7
                for j2=1:7
                    sboxX(i2,j2)=keyBinaryX(cr);
                    sboxY(i2,j2)=keyBinaryY(cr);
                    cr=cr+1;
                end
            end

            for j3=1:7
                %---
                rowX=sboxX(j3,1)*2+sboxX(j3,7);
                if(rowX==0)
                    rowX=1;
                end;

columnX=sboxX(j3,2)*16+sboxX(j3,3)*8+sboxX(j3,4)*4+sboxX(j3,5)*2+sboxX(j3,6);
                if(columnX==0)
                    columnX=1;
                end;
                rowY=sboxY(j3,1)*2+sboxY(j3,7);
                if(rowY==0)
                    rowY=1;
                end;

columnY=sboxY(j3,2)*16+sboxY(j3,3)*8+sboxY(j3,4)*4+sboxY(j3,5)*2+sboxY(j3,6);
                if(columnY==0)
                    columnY=1;
                end;
                if(j3==1)
                    sboxoX=S1(rowX,columnX);
                    sboxoY=S1(rowY,columnY);
                elseif(j3==2)
```

```
                    sboxoX=S2(rowX,columnX);
                    sboxoY=S2(rowY,columnY);
                elseif(j3==3)
                    sboxoX=S3(rowX,columnX);
                    sboxoY=S3(rowY,columnY);
                elseif(j3==4)
                    sboxoX=S4(rowX,columnX);
                    sboxoY=S4(rowY,columnY);
                elseif(j3==5)
                    sboxoX=S5(rowX,columnX);
                    sboxoY=S5(rowY,columnY);
                elseif(j3==6)
                    sboxoX=S6(rowX,columnX);
                    sboxoY=S6(rowY,columnY);
                elseif(j3==7)
                    sboxoX=S7(rowX,columnX);
                    sboxoY=S7(rowY,columnY);
                end;
                %Binarization
                cc3=1;
                tempX=sboxoX;
                tempY=sboxoY;
                while(cc3<6)
                    Q1X=tempX/2;
                    Q1Y=tempY/2;
                    R1X=mod(tempX,2);
                    R1Y=mod(tempY,2);
                    tempX=Q1X-R1X/2;
                    tempY=Q1Y-R1Y/2;
                    BX(cc3)=R1X;
                    BY(cc3)=R1Y;
                    cc3=cc3+1;
                end
                for ll=1:5
                    SoX(cc4)=BX(cc3-ll);
                    SoY(cc4)=BY(cc3-ll);
                    cc4=cc4+1;
                end

            %---
            end
            ctrl=ctrl+1;
   %while ends here
end
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
ccb=1;
checkN=1;
ff=int16((cc4-1)/8);
 for j18=1:ff
     cc18=1;
     index20=7;
     SoXI=0;
     SoYI=0;
     while(cc18<9 && checkN~=cc4)
```

```
          SoXI=SoXI+SoX(ccb)*power(2,index20);
          SoYI=SoYI+SoY(ccb)*power(2,index20);
          ccb=ccb+1;
          cc18=cc18+1;
          checkN=checkN+1;
          index20=index20-1;
      end
      SoXInt(j18)=SoXI;
      SoYInt(j18)=SoYI;
 end
%-------------------------------------------------------------------------
ccb=1;
      index20=cc4-1;
      SoXIn=0;
      SoYIn=0;
      cc18=1;
      while(cc18<cc4)
          SoXIn=SoXIn+SoX(ccb)*power(2,index20);
          SoYIn=SoYIn+SoY(ccb)*power(2,index20);
          ccb=ccb+1;
          cc18=cc18+1;
          index20=index20-1;
      end
%-------------------------------------------------------------------------
%step-4: generation of chaotic figures
%-------------------------------------------------------------------------
%step-4.1: Calculation of Initial Conditions
%a=0.0225,0.00225
b=sqrt(1-a^2);
Xo=var+SoXIn;% balanced 11881113816021014263531863716 5
Yo=var+SoYIn;% balanced 11515410245191208781102411495549
%step-4.2: Solutions of System used for generation of chaos
t=0.0:.1:9999+var;
X1=exp(a*t).*(Xo*cos(b*t)+1/b*(Yo-a*Xo).*sin(b*t));
Y1=exp(a*t).*(Yo*cos(b*t)+a/b*(Yo-a*Xo-b^2/a*Xo)*sin(b*t));
x=X1/(max(X1));
y=Y1/(max(Y1));
dec=0;
for i=1:sn
    plot(x+i,y-dec);
    dec=dec+0.1;
    hold on
end
%step-5: Saving the chaotic figure as image.
%-------------------------------------------------------------------------
%h=gcf=get current figure handle
saveas(gcf,'chaos.jpg');
chaosImage=imread('chaos.jpg');
%-----------------------------------------------
%cropping
imsize=size(chaosImage);
Height=imsize(1);
width=imsize(2);
rect1=[201 181 800 180];
rect2=[201 361 800 180];
rect3=[201 521 800 180];
```

```
RGB1 = imcrop(chaosImage,rect1);
RGB2 = imcrop(chaosImage,rect2);
RGB3 = imcrop(chaosImage,rect3);
croppedimgray1=rgb2gray(RGB1);
croppedimgray2=rgb2gray(RGB2);
croppedimgray3=rgb2gray(RGB3);
cimbin1=croppedimgray1/255;
cimbin2=croppedimgray2/255;
cimbin3=croppedimgray3/255;
balancedOne=xor(cimbin1,cimbin2);
balancedFin=xor(balancedOne,cimbin3);
balancedFinG=balancedFin*255;
%-------------------------------------------

%step-6: Accessing and processing the plaintext message.
%-------------------------------------------------------------------------------
%step-6.1: accessing the plaintextfavou
fid=fopen('plaintext.m');
if(fid==-1)
    display('File does not exist!');
else
    plainText=fileread('plaintext.m');
    display(plainText);
    plainTextInt=double(plainText);
    %display(plainTextInt);
    textSize=size(plainTextInt);
end;
%---------------------------------------

%step-6.2: converting the plaintext to binary values
cc6=1;
for j6=1:max(textSize)
    temp6=plainTextInt(j6);
    cc66=1;
    while(cc66<9)
        Q6=temp6/2;
        R6=mod(temp6,2);
        temp6=Q6-R6/2;
        B6(cc66)=R6;
        cc66=cc66+1;
    end
    for i6=1:8
        textBinary(cc6)=B6(cc66-i6);
        cc6=cc6+1;
    end
end
%------------2D-DFT of plaintext---------------------
 c20=int16(sqrt(cc6-1))-1;
 c21=1;
     for i21=1:c20
         for j21=1:c20
             plainTextDFT(i21,j21)=textBinary(c21);
             c21=c21+1;
         end
     end
 plainDFT=fft2(double(plainTextDFT));
```

```
 %mesh(abs(plainDFT));
 %-----------------------------------------------
%step-7: pre-enciphering processes
%-------------------------------------------------------------------------------
scale=(sqrt(8*max(textSize)));
sizedChaos=imresize(balancedFinG,[scale scale],'bicubic');
sizedC=imresize(chaosImage,[scale scale],'bicubic');
sizedChaosBin=sizedChaos/255;
cipherchaosran=randerr(int16(scale),int16(scale));

%if((8*max(textSize))<(h5*h25)/16)
i7=1;
size12=size(sizedChaosBin);
for ii7=1:min(size12)
    for jj7=1:max(size12)
        chaosCipher(i7)=sizedChaosBin(ii7,jj7);
        chaosCipherr(i7)=cipherchaosran(ii7,jj7);
        i7=i7+1;
    end
end
for hh=1:(8*max(textSize))
    ChaosCipher(hh)=chaosCipher(hh);
    ChaosCipher2(hh)=chaosCipher(hh);
end
ind=(randperm(20));
kmm=1;
for km=1:max(randperm(20))
    if(ind(km)>10)
        index100(kmm)=ind(km);
        kmm=kmm+1;
    end;
end

ind1=index100(7);
% ind1=15;
XXo=33;
for hhh=1:ind1
    indexXx1=mod(3*XXo,137);
    if(ChaosCipher(indexXx1)==1)
        ChaosCipher(indexXx1)=0;
    end;
    XXo=indexXx1;
end

if(i7<(8*max(textSize)))
    for hh=1:(8*max(textSize))-i7
        chaosCipher(i7)=0;
        chaosCipher(i7+hh)=0;
    end
end;
display('-----------------------------------------------------------------');
%step-8: Enciphering process
%-------------------------------------------------------------------------------
 for i8=1:(8*max(textSize))
        cipherText(i8)=xor(textBinary(i8),ChaosCipher(i8));
 end
```

```
%------------2D DFT of cipher---------------------
c10=int16(sqrt(8*max(textSize)))-1;
cipherTextDFT1=int16(rand(c10));
c11=1;
    for i11=1:c10
        for j11=1:c10
            cipherTextDFT(i11,j11)=cipherText(c11);
            c11=c11+1;
        end
    end
cipherDFT=fft2(double(cipherTextDFT1));
%mesh(abs(cipherDFT));
%-----------------------------------------------
c8=1;
for j8=1:max(textSize)
    cc8=1;
    index=7;
    cipherIn=0;
    while(cc8<9)
        cipherIn=cipherIn+cipherText(c8)*power(2,index);
        c8=c8+1;
        cc8=cc8+1;
        index=index-1;
    end
    cipherInt(j8)=cipherIn;
end
%display(cipherInt);
cipherTextt=char(cipherInt);
display(cipherTextt);

%step-9: Deciphering process
%-------------------------------------------------------------------------
display('-------------------------------------------------------------------------');
for i9=1:(8*max(textSize))
       DecryptedText(i9)=xor(cipherText(i9),chaosCipher(i9));
end
c9=1;
for j9=1:max(textSize)
    cc9=1;
    index9=7;
    decryptedIn=0;
    while(cc9<9)
        decryptedIn=decryptedIn+DecryptedText(c9)*power(2,index9);
        c9=c9+1;
        cc9=cc9+1;
        index9=index9-1;
    end
    decryptedInt(j9)=decryptedIn;
end
%display(decryptedInt);
decryptedText=char(decryptedInt);
display(decryptedText);
%-------------------------------------------------------------------------
```

Appendix B
Matlab Code for Performance Analysis

```
%-------------------------------------------------------------------------
%@uthor: Alem Haddush Fitwi, M.Sc in Computer Engineering
%Performance Analysis
%-------------------------------------------------------------------------
%Encryption Times
EncTCE=[0.031   0.047   0.078   0.094   0.156   0.250   0.437   0.515   1.232
3.823 ];
EncTRSA=[0.506  0.978   1.438   1.852   2.326   3.243   4.248   4.596   9.355
32.554];
EncTAES=[0.669  0.692   0.719   0.731   0.735   0.807   0.813   0.821   0.829
0.987];
%-------------------------------------------------------------------------
%Data in kilo bits and in bytes
data=[7.84  15.87   23.36   31.20   39.30   55.16   73.87   79.16   158.30
457.16];
datab=[1004 2031    2990    3994    5030    7060    9455    10132   20262
58516];

for i=1:10
    %Thruputs
    CETH(i)=datab(i)/EncTCE(i);
    RSATH(i)=datab(i)/EncTRSA(i);
    AESTH(i)=datab(i)/EncTAES(i);
    %Power consumption
    PCE(i)=.1*1.25*EncTCE(i);
    PRSA(i)=.1*1.25*EncTRSA(i);
    PAES(i)=.1*1.25*EncTAES(i);
end
%----CEET
plot(data,EncTCE)
title('Chaotic Encryption Times')
xlabel('Data sizes in Kilo bits')
ylabel('Encryption times in sec')
%----CETH
plot(datab,CETH)
title('Throughputs')
xlabel('Data sizes in bytes')
ylabel('Throughputs of various data sizes')
%----PCE
plot(data,PCE)
title('Energy Consumption During Enciphering Process')
xlabel('Data sizes in kilo bits')
ylabel('Energy Consumed in joules')

% CE, AES, RSA Encryption Times
plot(data,EncTCE)
hold on
plot(data,EncTRSA);
hold on
plot(data,EncTAES);
title('Encryption times of Chaotic, RSA and AES')
xlabel('Data sizes in kilo bits')
```

```
ylabel('Encryption Times in sec')

% CE, AES, RSA Thruputs
plot(datab,CETH)
hold on
plot(datab,RSATH);
hold on
plot(datab,AESTH);
title('Thruputs of Chaotic, RSA and AES')
xlabel('Data sizes in bytes')
ylabel('Thruputs')

% CE, AES, RSA Power consumptions
plot(datab,PCE)
hold on
plot(datab,PRSA);
hold on
plot(datab,PAES);
title('Energy Consumption of Chaotic, RSA and AES Algorithms')
xlabel('Data sizes in kilo bits')
ylabel('Energy Consumed in joules')

%----------------End of Program---------------------------------
```

Appendix C

Java Code for HMAC generation

```
//-------------------------------------------------------------------------
//@uthor: Alem Haddush Fitwi, M.Sc student of Computer Engineering,
 // AAU, FOT, ECE, June 2010 G.C
//--------------------------------Headers/packages-----------------------------
package alem.ChaoticCryptosystem;
import java.security.SignatureException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.lang.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.imageio.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.image.BufferedImage;
//----------------HMAC Class-----------------------------------------------------
public class HMAC {
        //---------The main method--------------------------
        public static void main(String[] AAUECE){
                int a=69621;
                int m=(int)Math.pow(2,31)-1;
                String key1=Integer.toString(a)+Integer.toString(m);
                System.out.println("key="+key1);
                System.out.println("Please wait for some minutes, HMAC is being
computed.....");
                String path="D:\\key.jpg";;
                String path1="D:\\keyGray.jpg";;
                BufferedImage image=null;
                BufferedImage new_img=null;
                int w=0,h=0;
                int[][] ImagePixel = new int[1000][1000];

                try{
                        image=ImageIO.read(new File(path));
                }catch (Exception e){
                    e.printStackTrace();
                  }
        //-----------------
          /* JFrame frame1 = new JFrame("A shared RGB Image used as a key!");
                JLabel label1 = new JLabel(new ImageIcon(image));
                frame1.getContentPane().add(label1, BorderLayout.CENTER);
```

```
                frame1.pack();
                frame1.setVisible(true);*/
        //---------------
           new_img = new BufferedImage( image.getWidth(),image.getHeight(),
           BufferedImage.TYPE_BYTE_GRAY);
        Graphics gr = new_img.getGraphics();
        gr.drawImage(image, 0, 0, null);
        gr.dispose();
//----------------
   /*JFrame frame2 = new JFrame("A shared gray Image used as a key!");
                JLabel label2 = new JLabel(new ImageIcon(new_img));
                frame2.getContentPane().add(label2, BorderLayout.CENTER);
                frame2.pack();
                frame2.setVisible(true);*/
//------------------
   try{
                ImageIO.write(new_img, "jpg", new File(path1));
                }catch (Exception e){
                    e.printStackTrace();
                 }
//-------------------
   int temp=0;
   byte pix[]=new byte[1];
                w=new_img.getWidth();
                h=new_img.getHeight();
                int[][]matrix=new int[w][h];
      for (int row=0;row<w;++row){
    for (int col=0;col<h;++col){
     new_img.getRaster().getDataElements(row, col, pix);
     matrix[row][col]=pix[0];
     ImagePixel[row][col]=Math.abs(matrix[row][col]);

    }
    }
      //----------------------
                //System.out.println("h="+h+"w="+w);
                //System.out.println("hw="+h*w);
                String hexVal=" ";
                for(int i=0;i<w;i++){
                for(int j=0;j<h;j++){
                        hexVal+=Integer.toHexString(ImagePixel[i][j]);
                   }
            }
          //System.out.println(hexVal);
```

```
            String  data1=hexVal;
            calculateHMAC(data1,key1);
        }

        //------------Fields---------------------------------

        static final String HEXES ="0123456789ABCDEF";
        private static final String HMAC_SHA1_ALGORITHM ="HmacSHA1";

        //-----------Byte to Hex converter method-------------
        public static String getHex( byte [] raw ){
                if ( raw == null ){
                        return null;
                  }
      final StringBuilder hex = new StringBuilder(raw.length*2);
      for ( final byte b : raw ){
        hex.append(HEXES.charAt((b & 0xF0)>> 4)).append(HEXES.charAt((b & 0x0F)));
       }
       return hex.toString();
    }

  //----------The HMAC method-----------------------------

        public static String calculateHMAC(String data, String key){

                String result=null;

                try {
                        byte[] keyBytes = key.getBytes();                       ;
                        SecretKeySpec signingKey = new SecretKeySpec(keyBytes,
HMAC_SHA1_ALGORITHM);
                        Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
                        mac.init(signingKey);
                        byte[] rawHmac = mac.doFinal(data.getBytes());
                        String hexBytes = getHex(rawHmac);
                        result=hexBytes;
                        //System.out.println("Len=" + result.length());
                        System.out.println("HMAC="+ result);
                }
                catch (Exception e) {
                        System.out.println("Failed to generate HMAC cos of" + e.getMessage());
                        }
                return result;
                }
}
```

Appendix D

Java Code for AES

```
//@uthor: Alem Haddush Fitwi, M.Sc student of Computer Engineering, May 2010 G.C
//-------------------------------Headers/packages-----------------------------
package alem.ChaoticCryptosystem;
import java.io.*;
import java.security.*;
import javax.crypto.*;
import javax.swing.JOptionPane;

public class AESE{
  //-----------------------Main Method begins here---------------------------
  public static void main(String[] args)
   {
        //-------------------Key Generation-----------------------------------
        String outFile="D:\\Thesis\\text\\key.txt";
     try{
                KeyGenerator keygen = KeyGenerator.getInstance("AES");
        SecureRandom random = new SecureRandom();
        keygen.init(random);
        SecretKey key = keygen.generateKey();
        System.out.println("key="+key);
        ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(outFile));
         out.writeObject(key);
         out.close();

        }
        catch (IOException e)
        {
        e.printStackTrace();
        }
     catch (GeneralSecurityException e)
        {
        e.printStackTrace();
        }

        //-------------------AES Enciphering-----------------------------------

      String inFileK1="D:\\Thesis\\text\\key.txt";
      String inFile1="D:\\Thesis\\text\\ptxt10.txt";
      String outFile1="D:\\Thesis\\text\\cipher3.txt";


     try{
           int mode1= Cipher.ENCRYPT_MODE;
```

```
        ObjectInputStream keyIn = new ObjectInputStream(new FileInputStream(inFileK1));
      Key key = (Key) keyIn.readObject();
      keyIn.close();

      InputStream in1 = new FileInputStream(inFile1);
      OutputStream out1 = new FileOutputStream(outFile1);
      Cipher cipher1 = Cipher.getInstance("AES");
      cipher1.init(mode1, key);
      crypt(in1, out1, cipher1);
      in1.close();
      out1.close();

}
catch (IOException e)
    {
    e.printStackTrace();
    }
catch (GeneralSecurityException e)
    {
    e.printStackTrace();
    }
catch (ClassNotFoundException e)
    {
    e.printStackTrace();
    }

//--------------------AES Deciphering-----------------------------------
String inFileK2="D:\\Thesis\\text\\key.txt";
String inFile2="D:\\Thesis\\text\\cipher3.txt";
String outFile2="D:\\Thesis\\text\\decrypt3.txt";
double jj=0.32;

try{
        int mode2 = Cipher.DECRYPT_MODE;
        ObjectInputStream keyIn = new ObjectInputStream(new FileInputStream(inFileK2));
      Key key = (Key) keyIn.readObject();
      keyIn.close();

      InputStream in2 = new FileInputStream(inFile2);
      OutputStream out2 = new FileOutputStream(outFile2);
      Cipher cipher2 = Cipher.getInstance("AES");
      cipher2.init(mode2, key);
      crypt(in2, out2, cipher2);
      in2.close();
      out2.close();
```

```
      }
      catch (IOException e)
          {
          e.printStackTrace();
          }
      catch (GeneralSecurityException e)
          {
          e.printStackTrace();
          }
      catch (ClassNotFoundException e)
          {
          e.printStackTrace();
          }

   }
   //------------------End of Main Method-------------------------------------
   //-------------Member Method Begins here-----------------------------------
   public static void crypt(InputStream in, OutputStream out, Cipher cipher)
        throws IOException, GeneralSecurityException
   {
     int blockSize = cipher.getBlockSize();
     int outputSize = cipher.getOutputSize(blockSize);
     byte[] inBytes = new byte[blockSize];
     byte[] outBytes = new byte[outputSize];
     int inLength = 0;
     boolean more = true;
     while (more)
     {
       inLength = in.read(inBytes);
       if (inLength == blockSize)
       {
         int outLength = cipher.update(inBytes, 0, blockSize, outBytes);
         out.write(outBytes, 0, outLength);
       }
       else more = false;
     }
     if (inLength > 0) outBytes = cipher.doFinal(inBytes, 0, inLength);
     else outBytes = cipher.doFinal();
     out.write(outBytes);
   }
   //-------------Member Method Ends here-------------------------------------
 }
//End of Class AESE
```

Appendix E

Java code for RSA

```
//-------------------------------------------------------------------------
//----------@uthor: Alem Haddush Fitwi. GSR/0996/01-------------
//----------------------Headers/packages---------------------------------

package alem.ChaoticCryptosystem;
import java.security.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.UIManager;

//----------------------class Jframe-----------------------------

public class RSAFinal extends javax.swing.JFrame {


    public RSAFinal() {
        initComponents();
    }


    private void initComponents() {

        panelKey = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        textKeyLength = new javax.swing.JTextField();
        jScrollPane1 = new javax.swing.JScrollPane();
        modulus = new javax.swing.JTextArea();
        jScrollPane2 = new javax.swing.JScrollPane();
        pubKey = new javax.swing.JTextArea();
        jScrollPane3 = new javax.swing.JScrollPane();
        priKey = new javax.swing.JTextArea();
        generateKey = new javax.swing.JButton();
        encryptPanel = new javax.swing.JPanel();
        plainTextScrollPane = new javax.swing.JScrollPane();
        plainText = new javax.swing.JTextArea();
```

```
        jLabel6 = new javax.swing.JLabel();
        jScrollPane4 = new javax.swing.JScrollPane();
        encryptedText = new javax.swing.JTextArea();
        jLabel7 = new javax.swing.JLabel();
        encrypt = new javax.swing.JButton();
        textFieldFile = new javax.swing.JTextField();
        browse = new javax.swing.JButton();
        decryptPanel = new javax.swing.JPanel();
        jLabel8 = new javax.swing.JLabel();
        jScrollPane5 = new javax.swing.JScrollPane();
        decipheredText = new javax.swing.JTextArea();
        decrypt = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("RSA Algorithm Implementation. By Alem H, AAU, ECE.");
        setResizable(false);


panelKey.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.c
reateTitledBorder(null, "Key Generation", javax.swing.border.TitledBorder.LEFT,
javax.swing.border.TitledBorder.DEFAULT_POSITION)));

        jLabel1.setText("key Length:");

        jLabel2.setText("number of Bits");

        jLabel3.setText("Modulus (N) :");

        jLabel4.setText("Public Key (e) :");

        jLabel5.setText("Private Key (d) :");

        textKeyLength.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        textKeyLength.setText("32");

        modulus.setColumns(20);
        modulus.setEditable(false);
        modulus.setLineWrap(true);
        modulus.setRows(10);
        jScrollPane1.setViewportView(modulus);

        pubKey.setColumns(20);
        pubKey.setEditable(false);
        pubKey.setLineWrap(true);
        pubKey.setRows(10);
        jScrollPane2.setViewportView(pubKey);
```

```
        priKey.setColumns(20);
        priKey.setEditable(false);
        priKey.setLineWrap(true);
        priKey.setRows(10);
        jScrollPane3.setViewportView(priKey);

        generateKey.setText("Generate Keys");
        generateKey.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                generateKeyActionPerformed(evt);
            }
        });
    //-----------------------------------------------------------------------
        javax.swing.GroupLayout panelKeyLayout = new javax.swing.GroupLayout(panelKey);
        panelKey.setLayout(panelKeyLayout);
        panelKeyLayout.setHorizontalGroup(
            panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(panelKeyLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addComponent(jLabel3, javax.swing.GroupLayout.PREFERRED_SIZE, 89,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 117,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G, false)
                    .addComponent(jScrollPane2, 0, 0, Short.MAX_VALUE)
                    .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE, 115,
Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                    .addGroup(panelKeyLayout.createSequentialGroup()
                        .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE,
104, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                            .addGroup(panelKeyLayout.createSequentialGroup()
```

```
                        .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(textKeyLength,
javax.swing.GroupLayout.PREFERRED_SIZE, 33,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jLabel2))
                    .addComponent(generateKey)))
                .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 117,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        panelKeyLayout.linkSize(javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {jScrollPane1, jScrollPane2, jScrollPane3});

        panelKeyLayout.setVerticalGroup(
            panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(panelKeyLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                .addGroup(panelKeyLayout.createSequentialGroup()

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELI
NE)
                        .addComponent(jLabel3)
                        .addComponent(jLabel4)
                        .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                        .addGroup(panelKeyLayout.createSequentialGroup()

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILI
NG)
                            .addComponent(jScrollPane2,
javax.swing.GroupLayout.Alignment.LEADING, 0, 0, Short.MAX_VALUE)
                            .addComponent(jScrollPane1,
javax.swing.GroupLayout.Alignment.LEADING,
```

```
javax.swing.GroupLayout.PREFERRED_SIZE, 62,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
panelKeyLayout.createSequentialGroup()

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILI
NG)
                            .addGroup(panelKeyLayout.createSequentialGroup()

.addGroup(panelKeyLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELI
NE)
                                    .addComponent(jLabel1)
                                    .addComponent(jLabel2)
                                    .addComponent(textKeyLength,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                                .addGap(29, 29, 29))
                            .addComponent(jScrollPane3, 0, 0, Short.MAX_VALUE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)))
                    .addGap(23, 23, 23))
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
panelKeyLayout.createSequentialGroup()
                    .addComponent(generateKey)
                    .addContainerGap())))
        );

        panelKeyLayout.linkSize(javax.swing.SwingConstants.VERTICAL, new
java.awt.Component[] {jScrollPane1, jScrollPane2, jScrollPane3});

        encryptPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Encryption"));

        plainText.setColumns(20);
        plainText.setEditable(false);
        plainText.setLineWrap(true);
        plainText.setRows(5);
        plainText.setWrapStyleWord(true);
        plainTextScrollPane.setViewportView(plainText);

        jLabel6.setText("Plain Text from Source File : ");

        encryptedText.setColumns(20);
        encryptedText.setEditable(false);
        encryptedText.setLineWrap(true);
```

```
        encryptedText.setRows(5);
        encryptedText.setWrapStyleWord(true);
        jScrollPane4.setViewportView(encryptedText);

        jLabel7.setText("Encrypted Text");

        encrypt.setText("Encrypt");
        encrypt.setName("");
        encrypt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                encryptActionPerformed(evt);
            }
        });

        textFieldFile.setEditable(false);

        browse.setText("Browse...");
        browse.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                browseActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout encryptPanelLayout = new
javax.swing.GroupLayout(encryptPanel);
        encryptPanel.setLayout(encryptPanelLayout);
        encryptPanelLayout.setHorizontalGroup(

encryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(encryptPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(encryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAI
LING)
                    .addComponent(jScrollPane4, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 479, Short.MAX_VALUE)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
encryptPanelLayout.createSequentialGroup()
                        .addComponent(jLabel7)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
336, Short.MAX_VALUE)
                        .addComponent(encrypt))
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
encryptPanelLayout.createSequentialGroup()
                        .addComponent(jLabel6, javax.swing.GroupLayout.PREFERRED_SIZE, 112,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(textFieldFile, javax.swing.GroupLayout.DEFAULT_SIZE, 278,
Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(browse))
            .addComponent(plainTextScrollPane,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.PREFERRED_SIZE, 479,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addContainerGap())
    );
    encryptPanelLayout.setVerticalGroup(

encryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(encryptPanelLayout.createSequentialGroup()

.addGroup(encryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
            .addComponent(jLabel6)
            .addComponent(browse)
            .addComponent(textFieldFile, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(plainTextScrollPane, javax.swing.GroupLayout.PREFERRED_SIZE,
55, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(encryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
            .addComponent(jLabel7)
            .addComponent(encrypt))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jScrollPane4, javax.swing.GroupLayout.PREFERRED_SIZE, 53,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap())
    );

    encryptPanelLayout.linkSize(javax.swing.SwingConstants.VERTICAL, new
java.awt.Component[] {jScrollPane4, plainTextScrollPane});

    decryptPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Decryption"));

    jLabel8.setText("Decrypted Text");

    decipheredText.setColumns(20);
    decipheredText.setEditable(false);
```

```
        decipheredText.setLineWrap(true);
        decipheredText.setRows(5);
        decipheredText.setWrapStyleWord(true);
        jScrollPane5.setViewportView(decipheredText);

        decrypt.setText("Decrypt");
        decrypt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                decryptActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout decryptPanelLayout = new
javax.swing.GroupLayout(decryptPanel);
        decryptPanel.setLayout(decryptPanelLayout);
        decryptPanelLayout.setHorizontalGroup(

decryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(decryptPanelLayout.createSequentialGroup()
                .addContainerGap()

.addGroup(decryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
DING, false)
                    .addComponent(jScrollPane5, javax.swing.GroupLayout.PREFERRED_SIZE, 473,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(decryptPanelLayout.createSequentialGroup()
                        .addComponent(jLabel8)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(decrypt)))
                .addContainerGap(16, Short.MAX_VALUE))
        );
        decryptPanelLayout.setVerticalGroup(

decryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(decryptPanelLayout.createSequentialGroup()

.addGroup(decryptPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASE
LINE)
                    .addComponent(jLabel8)
                    .addComponent(decrypt))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane5, javax.swing.GroupLayout.PREFERRED_SIZE, 55,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
```

```
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                    .addComponent(panelKey, javax.swing.GroupLayout.Alignment.LEADING, 0, 515,
Short.MAX_VALUE)
                    .addComponent(decryptPanel, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                    .addComponent(encryptPanel, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                .addContainerGap(11, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(panelKey, javax.swing.GroupLayout.PREFERRED_SIZE, 143,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(encryptPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(decryptPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(22, 22, 22))
        );

        pack();
    }

private void browseActionPerformed(java.awt.event.ActionEvent evt) {
    readFile();
}

private void decryptActionPerformed(java.awt.event.ActionEvent evt) {

   String enc = encryptedText.getText();
   decipheredText.setText("");
   //Timer begins here
```

```
    long start=System.currentTimeMillis();
    if(enc.equals(""))
    {
       JOptionPane.showMessageDialog(this,String.format("No File to decrypt!"));
          return;
    }
   int beginIndex=0;
   int endIndex= enc.indexOf(" ");
   String decipheredMessage="";
   while(endIndex != -1)
   {
      String dec = enc.substring(beginIndex, endIndex);
      BigInteger decipheredMessageAsBigInteger = decrypt(new BigInteger(dec,16));
      decipheredMessage +=  new String(decipheredMessageAsBigInteger.toByteArray());
      beginIndex =endIndex+1;
      endIndex = enc.indexOf(" ", beginIndex);
   }
   decipheredText.setText(decipheredMessage);
   //Timer ends here
   long elapsed=System.currentTimeMillis()-start;
   JOptionPane.showMessageDialog(null, "RSA Decryption is successfully
completed!\n"+"Time elapsed is: "+((double)elapsed/1000)+" seconds","RSA decryption Timer,
By Alem",JOptionPane.PLAIN_MESSAGE );

}
public void readFile()
{
      fileChooser = new JFileChooser();
      int result = fileChooser.showOpenDialog(this);
      String inFile ="";
      String str;

      if(result == JFileChooser.APPROVE_OPTION)
      {
                  try
                  {
                        File file = fileChooser.getSelectedFile();
                        textFieldFile.setText(file.getPath());
                        str=textFieldFile.getText();
                        File in = new File(str);
                        Scanner read = new Scanner(in);

                        while(read.hasNext())
                        {
                     inFile += new String(read.nextLine())+ "\n";
                        }
```

```
                plainText.setText(inFile);
                        read.close();
            }
                catch(FileNotFoundException e)
                {
                        JOptionPane.showMessageDialog(this,String.format("File not found"));
                        return;
                }
        }
}
   private void encryptActionPerformed(java.awt.event.ActionEvent evt) {
   String plain = plainText.getText();
   encryptedText.setText("");
   //Timer begins here
   long start=System.currentTimeMillis();
   if(plain.equals("")|| modulus.getText().equals(""))
   {
      JOptionPane.showMessageDialog(this,String.format("ERROR!\nTry again!"));
         return;
   }
   int textLen= plain.length();
   int j=0;
   String encryptedMessage ="";
   while(j<textLen)
   {
      int endIndex = (j+bitLen/8) > textLen ? textLen : j+bitLen/8;
      String blockString = plain.substring(j,endIndex);
      byte[] plainByte = blockString.getBytes();
      BigInteger message = new BigInteger(plainByte);
      BigInteger encryptedMessageAsBigInteger = encrypt(message);
      encryptedMessage += encryptedMessageAsBigInteger.toString(16) + " ";
      j += blockString.length();
   }
   encryptedText.setText(encryptedMessage);
   //Timer ends here
   long elapsed=System.currentTimeMillis()-start;
         JOptionPane.showMessageDialog(null, "RSA encryption is successfully
completed!\n"+"Time elapsed is: "+((double)elapsed/1000)+" seconds","RSA encryption Timer,
By Alem",JOptionPane.PLAIN_MESSAGE );

}

private void generateKeyActionPerformed(java.awt.event.ActionEvent evt) {

   clearFields();
```

```
   String s=textKeyLength.getText();
   if(!isNumber(s))
   {
      JOptionPane.showMessageDialog(this,String.format("Please Enter Numeric Value!"));
      textKeyLength.requestFocus();
      textKeyLength.selectAll();
      return;
   }
   bitLen = Integer.parseInt(textKeyLength.getText());
   if(bitLen<8)
   {
      JOptionPane.showMessageDialog(this,String.format("Please Enter Bitlength >= 8 !"));
      textKeyLength.requestFocus();
      textKeyLength.selectAll();
      return;
   }
   // ---------------------------get two big primes----------------------------
   do
   {
      r = new SecureRandom();
      p = BigInteger.probablePrime(bitLen, r);
      do
      {
         q = BigInteger.probablePrime(bitLen, r);
      }while(q.compareTo(p)==0);
       N = p.multiply(q);
      phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
       e = BigInteger.probablePrime(bitLen/2, r);
       while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0 )
       {
          e.add(BigInteger.ONE);
       }
   }while(e.gcd(p.subtract(BigInteger.ONE)).compareTo(BigInteger.ONE)>0 ||
          e.gcd(q.subtract(BigInteger.ONE)).compareTo(BigInteger.ONE)>0);
   d = e.modInverse(phi);
   modulus.setText(N.toString());
   pubKey.setText(e.toString());
   priKey.setText(d.toString());
}

private BigInteger encrypt(BigInteger message)
{
   return message.modPow(e, N);
}
private BigInteger decrypt(BigInteger encrypted)
```

```
{
   return encrypted.modPow(d, N);
}
private boolean isNumber(String s)
{
   int i = s.length();
   for(int j =0;j<i;j++)
   {
      if(Character.isDigit(s.charAt(j))== false)
         return false;
   }
   return true;
}
private void clearFields()
{
   modulus.setText("");
   pubKey.setText("");
   priKey.setText("");
}

   public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
         public void run() {
            UIManager.put("swing.boldMetal",Boolean.FALSE);
            new RSAFinal().setVisible(true);
         }
      });
   }

   //--------------------------Declaration---------------------------------
   private javax.swing.JButton browse;
   private javax.swing.JTextArea decipheredText;
   private javax.swing.JButton decrypt;
   private javax.swing.JPanel decryptPanel;
   private javax.swing.JButton encrypt;
   private javax.swing.JPanel encryptPanel;
   private javax.swing.JTextArea encryptedText;
   private javax.swing.JButton generateKey;
   private javax.swing.JLabel jLabel1;
   private javax.swing.JLabel jLabel2;
   private javax.swing.JLabel jLabel3;
   private javax.swing.JLabel jLabel4;
   private javax.swing.JLabel jLabel5;
   private javax.swing.JLabel jLabel6;
   private javax.swing.JLabel jLabel7;
   private javax.swing.JLabel jLabel8;
```

```
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JScrollPane jScrollPane5;
    private javax.swing.JTextArea modulus;
    private javax.swing.JPanel panelKey;
    private javax.swing.JTextArea plainText;
    private javax.swing.JScrollPane plainTextScrollPane;
    private javax.swing.JTextArea priKey;
    private javax.swing.JTextArea pubKey;
    private javax.swing.JTextField textFieldFile;
    private javax.swing.JTextField textKeyLength;
    private SecureRandom r;
    private BigInteger p;
    private BigInteger q;
    private BigInteger N;
    private BigInteger phi;
    private BigInteger e;
    private BigInteger d;
    private int bitLen = 1024;
    private JFileChooser fileChooser;

}
```

REFERENCES

[1] By Nigel Smart, “Cryptography: An Introduction”, *CRC press,* pp.11-387, June 2010

[2] By Charles Pfleeger, “Security in Computing, Fourth Edition”, *Pfleeger Consulting Group, Shari Lawrence Pfleeger -RAND Corporation*, October, 2006

[3] K. Naik and D. S. L. Wei, “Software implementation strategies for power-conscious systems,” *Mobile Networks and Applications*, vol. 6, pp. 291-305, 2001.

[4] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz, “Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs”, *International Association for Cryptologic Research 2004*

[5] Federal Information Processing Standards Publication 197, November 26, 2001 Announcing the ADVANCED ENCRYPTION STANDARD (AES)

[6] William Stallings, “Lecture notes for use with Cryptography and Network Security”, May 2010.

[7] William Stallings, “Data and computer communications, 6th edition,” Prentice Hall, New Jersey, 1996

[8] Man Young Rhee, “Internet Security, Cryptographic Principles, Algorithms and Protocols”, John Wiley & Sons Ltd, 2003.

[9] By Wenbo Mao, “Modern Cryptography: Theory and Practice”, *Prentice Hall PTR,* pp. 44-700, July 25, 2003

[10] Ibrahim A. Al-Kadi, “The origins of cryptology: The Arab contributions”, April 1992

[11] Menezes, P. Van Oorschot, S. Vanstone, “Hand book of Applied Cryptography”, *CRC press*, pp. 1-588,1996.

[12] Walter Tuchman, "A brief history of the data encryption standard". *Intern besieged countering cyberspace scofflaws*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA. pp. 275–280, 1997.

[13] FIPS PUB 46-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Reaffirmed 1999 October 25

[14] FIPS PUB 46-2 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Reaffirmed 1993 December 30.

[15] FIPS PUB 46 FEDERAL INFORMATION PROCESSING STANDARDS

PUBLICATION affirmed 1977 January 15.

[16] FIPS PUB 46-1 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Reaffirmed 1988 January 22.

[17] Niels Ferguson, Richard Schroeppel, Doug Whiting, "A simple algebraic representation of Rijndael", *Proceedings of* Selected Areas in Cryptography, Lecture Notes in Computer Science. Springer-Verlag. pp. 103–111., 2001.

[18] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest; Clifford Stein, "Introduction to Algorithms (2e ed.)", MIT Press and McGraw-Hill. pp. 881–887., 2001

[19] Anoop MS , "Elliptic Curve Cryptography An Implementation Guide"

[20] D. Hankerson, A. Menezes, and S.A. Vanstone, "*Guide to Elliptic Curve Cryptography"*, Springer-Verlag, 2004.

[21] Stephen H. Kellert, "In the Wake of Chaos: Unpredictable Order in Dynamical Systems*"*, University of Chicago Press, 1993.

[22] Fengling Han, Jinhu Lu, Xinghuo Yu, Guanrong Chen and Yong FEng "Generating Multi-Scroll Chaotic Attractors Via a Linear Second Order Hysteresis System", *Watam press, 2005*

[23] Jinhu L' ua , Fengling Hanb, Xinghuo Yub, Guanrong Chenc, "Generating 3- D multi-scroll chaotic attractors: A hysteresis series switching method" , *IEEE, 2005*

[24] Fengling Han , Jiankun Hu a , Xinghuo Yu b , Yi Wang, "Fingerprint images encryption via multi-scroll chaotic attractors", Elsevier, 2007

[25] Jinhu Lü , K. Murali , Sudeshna Sinha , Henry Leung , M.A. Aziz-Alaoui ," Generating multi-scroll chaotic attractors by thresholding", 30 January 2008

[26] Mustak E. Yalcin, "Generating Multi-scroll and Hypercube Attractors: An overview", Lecture notes, *Istanbul Technical University.*

[27] Fengling Han, Xinghuo Yu, and Jiankun Hu, "A New Way of Generating Grid-Scroll Chaos and its Application to Biometric Authentication", *Melbourne VIC 3001*

[28] Guobo Xie , Simin Yu2, Yijun Liu,and Zhusong Liu, "Generation of Multi-Scroll Chaotic Attractors from Fifth-Order Chua System", *Guangdong University of Technology, Guangzhou, 510006, China*

[29] Weihua Deng , and Jinhu L¨ u, "Design of Multi-Directional Multi-Scroll Chaotic Attractors Based on Fractional Differential Systems", *Lanzhou University, china.*

[30] Xinghuo Yu, Fengling Han, Yong Feng, Guanrong Chen, "Domain of attraction of hysteresis-series based chaotic attractors.", vol-12, 2005

[31] By Raj Jain, "Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation nnd Modeling"

[32] Ben Laurie, "On Randomness", December 29, 2004

[33] Guang Gong , "C&O739x, Lecture Notes (draft)", Winter 1999.

,