CHAOS AND PUBLIC KEY INFRASTRUCTURE (PKI)

By

CHEW JUN YEE

DISSERTATION

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Supervised by:
Dr Varun Jeoti
Dr Abdul Huq b. M. Abdul Wahab

JUNE 2004

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

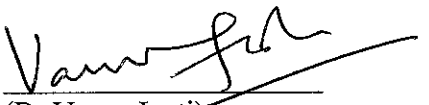
Chaos & Public Key Infrastructure

by

Chew Jun Yee

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,



(Dr Varun Jeoti)

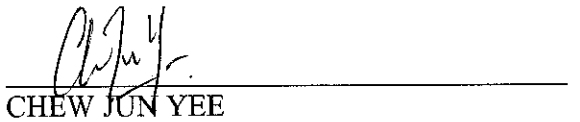
UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

June 2004

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



CHEW JUN YEE

ABSTRACT

The emergence of chaos theory promised a new era in the field of cryptography as the properties of a chaotic system are exploited. Many studies have been done in this area, in which various schemes employing chaotic systems have been proposed. Schemes ranging from different aspects of chaotic systems to the both symmetric and asymmetric encryption are published. However, according to [1], the author suggested a more comprehensive insight into both chaotic systems and cryptography algorithms is needed before doing any design to avoid having a "both weak and slow ciphers". The author of [1] has a published work entitled "Public-key Encryption Based on Chebyshev Maps" [2] and this is utilized as the core of a new public key encryption scheme. The new scheme proposed here is "Public-Key Encryption based on Logistic Map" which employs many similar concepts as [2].

A thorough study on various polynomials has been conducted and implementation on MATLAB has been done which includes conventional public key encryption scheme such as RSA algorithm. It is continued with the implementation of [2] to test for its workability in MATLAB platform. A major problem faced in this implementation has been solved while implementing the new logistic map scheme. Consequently, the new scheme is able to provide higher precision, thus higher security level, although at the price of the performance. Most importantly, however, is the proof of the workability of the whole new scheme. The project thus concludes with comparison of the new public key scheme based on logistic map with RSA algorithm on MATLAB platform.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank my supervisor, Dr Varun Jeoti for his great supervision and selfless help. He has been providing me with useful advices and guidance to lead this project into completion. Thank you for being understanding and patient with me.

I would also like to express my deepest gratitude to Ms Easwari Sivanandan for her amazing help, discussion and moral support throughout the whole project. She is my partner doing on symmetric chaotic encryption, and has been providing me with technical help in the same area. Thank you for being supportive and understanding.

Thank you to Dr Abdul Huq b. M. Abdul Wahab for his help in programming; Mr Ong Boon Leong for his references in starting in cryptography; any other lecturers or people for any of their direct or indirect help to me throughout the project.

Last but not least, thank you to both of my parents for their moral support without which I would have great difficulties in finishing this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background of Study

Many efforts have been put into developing chaos-based cryptographic systems in recent years as emerging chaotic systems promised a new direction for innovation in cryptography since the development of public-key cryptography. Many of these works are published, which shows a great deal of improvements and advancement over the years. However, a strong relationship between cryptographic systems and chaos-based systems is yet to be established, according to Kocarev in [1]. Cryptography and chaotic systems have many similarities as well as many differences. It has been pointed the properties of both for a clearer comparison (refer to Table 1):

Table 1 Comparison between Chaotic Systems and Cryptographic Systems (Source: "Chaos-Based Cryptography: A Brief Overview"[1])

CHAOTIC SYSTEMS	CRYPTOGRAPHIC ALGORITHMS
Phase space: (sub)set of real numbers	Phase space: finite set of integers
Iterations	Rounds
Parameters	Key
Sensitivity to a change in initial conditions and parameters	Diffusion
???? (No known counterpart)	Security & performance

Based on Table 1, designers of any new scheme must be able to overcome many the differences in order to integrate chaotic systems into cryptography. Many of the recently published works on this area have outlined the algorithm for software implementation. The focus of this project was such implementation, whereby a thorough study has being done on the following paper: "Public-Key Encryption Based on Chebyshev Maps" by Kocarev and Tasev [2].

1.2 Problem Statement

The introduction of chaotic systems into cryptography marks a very interesting area for research into innovating novel encryption schemes. In both symmetric (private key) and asymmetric (public key) encryption, chaos-based elements have been integrated to varying degree of success in terms of implementation, security and performance aspects. The focus of the project will be on encryption that uses chaotic maps, especially on Chebyshev maps. The work done is based on [2]. A new chaos-based public key scheme is first needed to be created and later on compare for its advantages and disadvantages with other similar schemes.

The reason of using chaos is that it has strong dynamical properties which give strong cryptographical properties.

1.3 Objective and Scope of Study

The objectives of the project for the first semester are:

1. To learn and study existing Public Key Infrastructure (PKI) schemes based security
2. To implement RSA algorithm in JAVA and MATLAB

The objectives of the project are for the second semester:

1. To study and understand Chebyshev maps-based public-key encryption
2. To create a new public key encryption scheme using another polynomial as a map substituting Chebyshev polynomial in [2]
3. To compare this new scheme with RSA algorithm on MATLAB platform

The coverage will be based on the implementation of public-key encryption scheme based on Chebyshev maps [2].

1.4 Justification

A new public key encryption scheme based on chaos has been created and defined in this project. It is based on logistic map. The whole scheme is modeled after work done by Kocarev and Tasev in "Public-Key Encryption Based on Chebyshev Maps" with modifications done to adapt logistic map. In the conclusion of the aforementioned paper, the authors say [2]:

> The algorithm described here works with Chebyshev polynomials, but can be generalized to work with any chaotic map $x_{n+1}=F_p(x_n)$ for which F can be written as $F_p(x)=f(pf^{-1}(x))$, so that $F_p(F_s)=F_{ps}$. Is there any other (chaotic) map with the semi-group property $F_p(F_s)=F_{ps}$?

They also cited work by Kohda and Fujisaki [10] on *Jacobian elliptic Chebyshev rational maps* which exhibits semi-group property, as their future research topic.

In this groundwork laid out, it is organic to proceed to look for a chaotic map with a semi-group property and replaces Chebyshev maps with this new map into the scheme with little or no modification, since it is general for any chaotic map with semi-group property as claimed by the authors. Thus came with the realization that logistic map can used due its property $F_p(F_s)=F_{p+s}$ (which is slightly different than the semi-group property mentioned). From here onwards, there is just some modification done to the scheme in [2] to adapt logistic map.

The usage of chaos provides security if implemented properly in a scheme; and with the current progress in chaos-based cryptography listed in [1], it can be seen that chaos-based schemes are having advantage in simplicity as well. This is true for scheme in [2] and the scheme proposed in this project.

The scheme proposed here is feasible for implementation for practical usage, provided with proper optimization for better performance without compromising security. It offers a new exciting topic for further research for wider implementation of chaos-based encryption.

CHAPTER 2
THEORY

2.1 Cryptography Fundamentals

In this section, the will be review on basic concepts of cryptography.

2.1.1 Public Key Encryption (Asymmetric Encryption)

Public key encryption (also known as asymmetric encryption) is a unique encryption scheme. Unlike symmetric encryption where a single secret key is used to encrypt and decrypt a message, public key encryption is asymmetric encryption. This means there are two different keys involved – a private key and a public key. These key pairs are mathematically linked. Encryption is done with one key and decryption can only be done with the other key. This solves the problem of secret key transmission of the symmetric encryption scheme. In this system, it is the recipient who generates a public and private key pair and post only the public key in the public domain. The sender looks up the recipient's public key in a public directory and encrypts his or her message using that key. The encrypted message is sent over the network to the recipient. Due to the uniqueness of asymmetric encryption, the encrypted message can be only decrypted with the other key – in this case is the recipient's private key. The private key is kept secret by the recipient. Thus, a hacker cannot decrypt the message even though he or she has intercepted the message and get hold of the recipient's public key. The encrypted message can be securely retrieved and decrypted by the recipient with his or her private key.

However, the scheme does not stop here. Asymmetric encryption has disadvantages as compared with symmetric encryption – it produces a large encrypted message and it takes more computation, thus slower. This is not suitable for transmission over the Internet. However, there is a novel way to get the best of symmetric and asymmetric encryption schemes. It is done by encrypting a message with a secret key (of symmetric encryption) and then encrypting only the secret key using the recipient's public key (of asymmetric encryption). The encrypted private key is not large in size

because a key is usually small. This poses no problem for transmission over the Internet. Both encrypted secret key and the encrypted message are sent over the network. This solves the problem of transfer while maintaining the security of an asymmetric system. This method - also referred as hybrid system - gets the best of both encryption schemes.

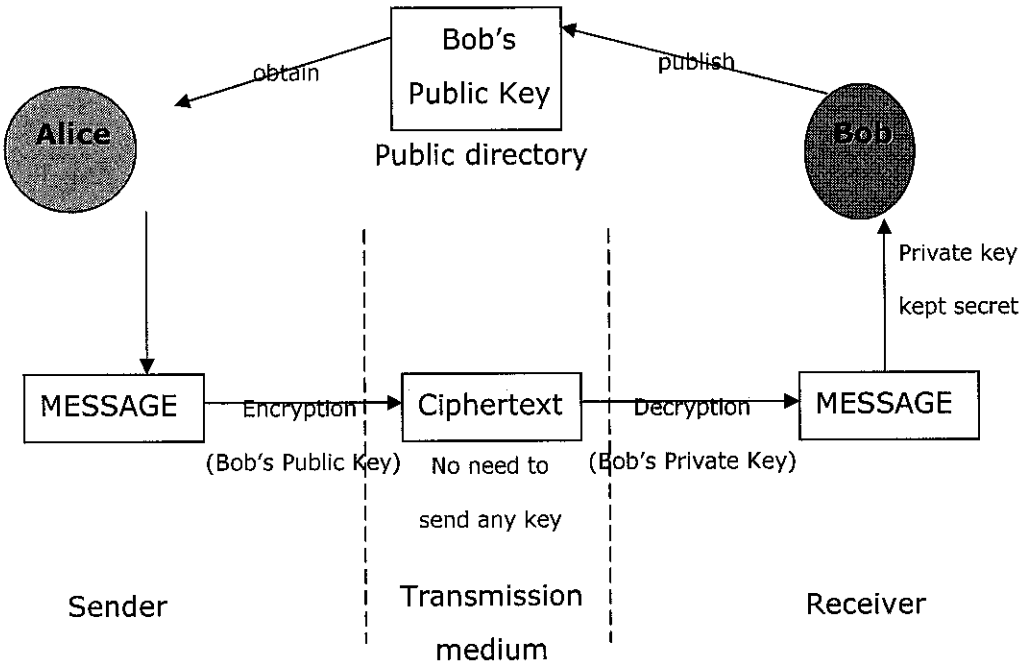


Figure 1 Public Key Encryption

2.1.2 *Public Key Infrastructure (PKI)*

Public Key Infrastructure is a multi-defined term. Generally it refers to a system of protocols, services and standards that support public key encryption. In extension to that definition, it usually includes the public key certificates, encryption schemes, digital signatures, digital certificates and non-repudiation as well. All these components are necessary to allow registration authorities to authenticate and verify the validity of parties that are involved in an electronic transaction.

However, there is no any particular standard or any strong emerging model of PKI. Much work is being currently done in this field and PKI is still subjected to evolution and modification. Below are the major components of PKI system.

2.1.2.1 Cryptographic Hash

Cryptographic hash is a way creating *digest* of an original data or file. Hash algorithm computes equation across a data or file and create a hash value. The unique features of cryptographic hash are:

- there is no possible way that the original data can be reconstructed from the hash value
- if there is a change, for example just *1 bit*, the hash value output will be entirely different

Cryptographic hash is therefore used to check a data's or file's integrity.

2.1.2.2 Digital Signature

The purpose of digital signature is to ensure that a data comes from a specific user, not from a substituted data from a third party. It uses cryptographic hash to create a digest of a file. The digest itself will be then encrypted with the sender's private key. The encrypted digest is attached with the encrypted file (which is encrypted with the recipient's public key). At the recipient's side, the person will first decrypt the file using his or her private key and then create a digest of the file using the same hash algorithm as the sender's. Then, the person will decrypt the attached encrypted digest with the sender's public key. The decrypted digest will be compared with the digest that the digest the recipient has just created. If both digests matches, the recipient can be assured that the file has not been deliberately changed by a third party.

2.1.2.3 Digital Certificate

However, a third party can change someone's public key in the directory, which makes the entire process failed right at the beginning. One way to ensure that public key belongs to the right person is through digital certificates. A digital certificate is a document that guarantees a public is associated with a particular user and. The digital certificate is issued by a trusted authority. To check for validity of a digital certificate, the trusted authority's public key is used. The digital certificate contains information on:

- Name, address, organization
- Owner's public key

- Certificate validity dates
- Certifying authority's digital signature

2.1.2.4 *Non-repudiation*

Non-repudiation is a way to prevent someone from cheating by breaking their promise. It is an extension to digital signatures whereby the time (before the digital signature is created) is encoded to the document. A trusted time source is used to ensure a reliable system.

2.2 Chaos Theory

Chaos theory describes unpredictable behaviour of natural, dynamic systems that are susceptible to slight changes in initial conditions. Although chaotic systems are complex, they are mathematically deterministic [4]. They obey mathematical laws, but their behaviour appears random.

From [4],

> Thus, while chaotic systems share many of the properties of stochastic processes, they also possess a deterministic structure which makes it possible to generate "noiselike" chaotic signals in a theoretical reproducible manner.

Using this property, cryptologists have adapted chaos into cryptography due to its random nature. The adaptation is in form chaotic mapping, which are thoroughly discussed in [5].

2.2.1 *Chaotic Maps*

Chaotic map is a function that transforms initial point into new location over and over again sequentially. Each round of such transformation is called *iteration*. It is chaotic in nature because a slight change in the initial condition will produce a totally

different outcome at a final fixed iteration. There are mainly three types of chaotic maps:

- One dimensional maps
- Multi-dimensional maps – coupled and uncoupled maps
- Phase space maps

The usage of chaotic maps in encryption schemes is obvious: to encrypt (as an algorithm) through the properties of diffusion and confusion of a chaotic map. Different schemes may use different maps according to the designer's justification. Chaotic maps in discussion in this paper will be only deterministic chaotic discrete time dynamical system [3] which in form of:

$$x(n+1) = f_k(x(n)) \tag{1}$$

where $f_k : S \in R^N \rightarrow R^N$ is a nonlinear function, and k denotes its parameters.

2.3 Chaos-based Cryptography

As discussed earlier in Section 1.1, many differences of the both chaotic systems and cryptography needed to be fully understood, in which will lead to gaining a better view on the relationship between the two. Referring to Table 2, it can be seen that the chaotic systems operates on set or subset of real numbers only while cryptographic systems on a finite set of integers [1]. This causes difficulty in having a practical implementation in form of circuitry. Another setback is that there is no known equivalent of security and performance in chaotic systems. However, these does not hinder from developing any chaos-based encryption schemes because there are three crucial similarities between cryptography and chaos. The equivalent of *rounds, key* and *diffusion* in cryptographic algorithms in chaotic systems are *iterations, parameters* and *initial changes sensitivity*, respectively.

In [1], it is suggested two general guiding principles for designing a practical algorithm:

- *diffusion*: spreading out the influence of a single plaintext digit over many ciphertext digits so as to hide the statistical structure of the plaintext
- *confusion*: use of transformations which complicate dependence of the statistics of ciphertext on the statistics of plaintext

2.3.1 Chebyshev Polynomial of the First Kind

Chebyshev map presented in [2] is based on Chebyshev Polynomial of the First Kind, which is defined as [2]:

$$T_{p+1}(x) = 2xT_p(x) - T_{p-1}(x) \tag{2}$$

where degree $p=1,2,...$, $T_0=1$, and $T_1=x$. It has semi-group properties:

$$T_r(T_s(x)) = T_{rs}(x) \tag{3}$$

from where the following equation is derived:

$$x_n = T_p(T_p(...T_p(x_0))) = T_{p^n}(x_0) \tag{4}$$

2.3.2 Other Polynomials and Maps

Moving from Chebyshev Polynomial of the First Kind, there many polynomials that are potential candidates for a new chaotic map scheme similar to [2]. In this section, the objective is to identify which polynomial will be suitable for a new chaos-based public key encryption.

2.3.3 Chebyshev Polynomial of the Second Kind

Chebyshev Polynomials of the Second Kind has the same recursive formula as the First Kind, except it has different initial values. Chebyshev Polynomial of the Second Kind is defined as follows:

$$U_n(x) = 2xU_{n+1}(x) - U_{n+2}(x) \tag{5}$$

where $U_0(x)=1$, $U_1(x)=2x$ etc.

2.3.4 Bessel Polynomial

Bessel polynomial has the following recursive formula:

$$B_n(x) = (2n-1)B_{n-1}(x) - x^2 B_{n-2}(x) \tag{6}$$

where $B_0(x)=1$, $B_1(x)=1+jx$ etc.

2.3.5 Legendre Polynomial

Legendre polynomial has the following recursive formula:

$$nP_n(t) = (2n-1)tP_{n-1}(t) - (n-1)P_{n-2}(t) \tag{7}$$

where $P_o(t)=1$, $P_1(t)=t$ etc.

2.3.6 Laguerre Polynomial

Laguerre polynomial has the following recursive formula:

$$L_n(t) = (2n-1-t)L_{n-1}(t) - (n-1)^2 L_{n-2}(t) \tag{8}$$

where $L_0(t)=1$, $L_1(t)=1-t$ etc.

2.3.7 Tent Map

Tent map is defined as:

$$\varphi = \mathrm{mod}\left[x - \left\lfloor \frac{x+a}{2} \right\rfloor \right] \tag{9}$$

where $-1 \le x \le 1$

All the aforementioned polynomials and maps do not have one particular characteristic to be a substitution polynomial for the scheme in [2], which is the semi-group property (see Equation (3)). In addition, Bessel function involves imaginary component which makes finding the terms even more difficult. These factors make a direct substitution of a polynomial to Chebyshev Polynomial of the First Kind is not possible. It requires a different scheme to make it work. This might a very time consuming process. A map that has the semi-group is highly desired.

2.4 Logistic Map

Logistic map is a non-linear dynamic equation, which demonstrates complex chaotic behaviour despite its simplicity.

Logistic map has the following recursive formula:

$$x(n+1) = rx(n)[1 - x(n)] \tag{10}$$

where $0<x(0)<1$ and r and is a constant.

The behaviour of the map is dependent on the constant r. The value of r must be chose in such a way that the logistic map exhibits chaotic behaviour. Referring to Table 2, the suitable value will be within $3.57 \leq r \leq 4$.

Semi-group property is exhibited here:

$$\varphi_r(\varphi_s(x)) = \varphi_{r+s}(x) \tag{11}$$

Table 2 Values of r and its effect on the behaviour of the logistic map (Source: "Logistic Map" Wikipedia)

r Range	Logistic Map Behaviour
$0<r<1$	• the population will eventually die, independent of the initial population
$1<r<2$	• the population will quickly stabilize on a single value • this value depends on r but does not depend on the initial population
$2<r<3$	• the population will also eventually stabilize on a single value, but first oscillates around that value for some time • the final value does not depend on the initial population
$3<r<3.45$	• the population will oscillate between two values forever • these two values are dependent on r but independent of the initial population
$3.45<r<3.54$	• the population will oscillate between four values forever • this behavior does not depend on the initial population
$3.54<r<3.57$	• the population will oscillate between 8 values, then 16, 32, etc • the lengths of the parameter intervals which yield the same number of oscillations decrease rapidly • the ratio between the lengths of two successive such bifurcation intervals approaches the Feigenbaum constant $\delta = 4.669$ • all of these behaviors do not depend on the initial population
$r=3.57$	• the onset of chaos • no any further oscillations observed • slight variations in the initial population yield dramatically different results over time, a prime characteristic of chaos
$3.57<r\leq4$	• exhibit chaotic behaviour, but there are still certain isolated of r that appear to show non-chaotic behavior; for instance around 3.82 there is a range of parameters r which show oscillation between three values, and for slightly higher values of r oscillation between 6 values, then 12 etc • there are other ranges which yield oscillation between 5 values etc • all oscillation periods do occur • these behaviours are independent of the initial value
$r>4$	• the values eventually leave the interval [0,1] and diverge for almost all initial values

Using MATLAB, the bifurcation diagram for logistic map is drawn. Bifurcation is where the period doubles, quadruples, etc that accompanies the onset of chaos. In following diagram (Figure 2) is generated using MATLAB. The range for r is from 0 to 4. The first bifurcation occurs at $r=3$. The chaotic behaviour shows when $r=3.57$. This bifurcation diagram is a fractal.

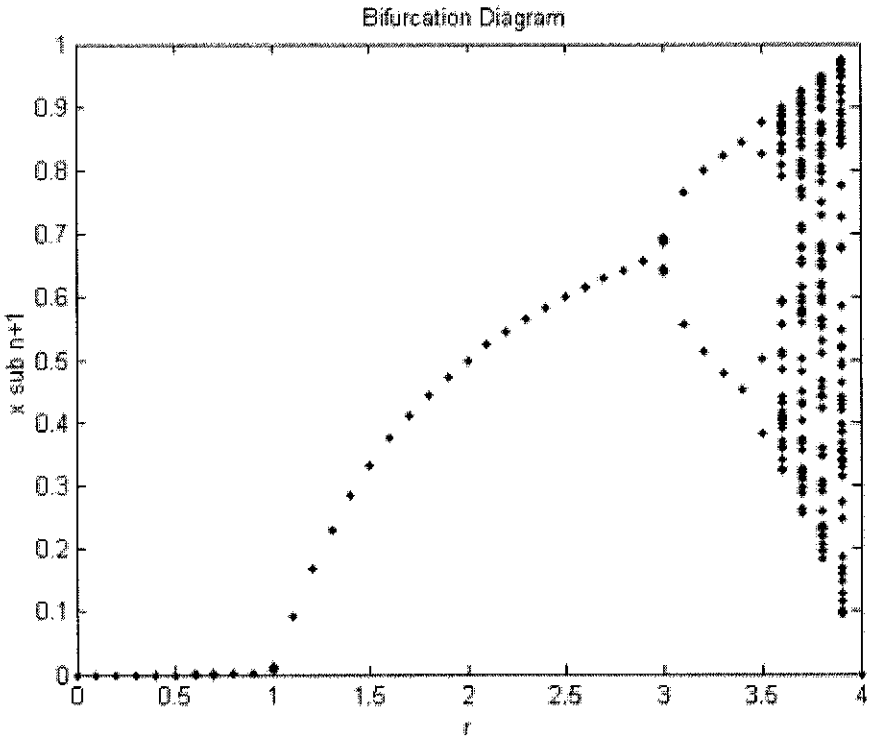


Figure 2 Bifurcation diagram for logistic map

CHAPTER 3
METHODOLOGY

3.1 RSA Algorithm (Public Key Scheme)

RSA algorithm involves many steps. The mathematics of RSA algorithm will be described here as two parts – encryption and decryption. Assuming Alice is the sender and Bob is the receiver in this example. In asymmetric encryption, Alice will take the Bob's public key for encryption. That means Bob must firstly generate a key pair.

3.1.1 Key Generation

The algorithm for key pair generation as follows:

1. Generate two large prime numbers, p and q, of more or less the same size with condition that $p \neq q$.
2. Calculate $n = pq$. This will result n with a bit length of at least 1024 bits. The integer n is also referred as *RSA modulus*.
3. Calculate $\Phi(n) = (p-1)(q-1)$.
4. Select a random integer e, the *RSA enciphering exponent*, such that $1<e<\Phi(n)$ and $\gcd(e, \Phi(n))=1$. This is done using Euclidean algorithm
5. Compute d, the *RSA deciphering exponent*, such that $1<d<\Phi(n)$ and $ed \equiv 1 \pmod{\Phi(n)}$.

Bob's *RSA public key* will be (n,e) and his *RSA private key* will be d.

3.1.2 Message Encryption

Encryption process is done by Alice, and she has to retrieve Bob's public key which is (n,e). The algorithm for encryption of message m as follows (assuming $\gcd(m,n)=1$):

1. Compute $c \equiv m^e \pmod{n}$.

2. Send c, the encrypted message to Bob.

3.1.3 Message Decryption

The decryption process is done by Bob after he receives c from Alice. The algorithm as follows:

1. Compute $m \equiv c^d \pmod{n}$, with d (the private key).

The decryption of the ciphertext is done and Bob is able to read Alice's message.

3.2 Algorithm for Public-Key Encryption Using Chebyshev Maps

From [2], the following algorithm is suggested.

3.2.1 Key Generation

The following action to generate a set of keys is done by the recipient, in this example is Alice.

- Generate a large integer s
- Select a random number $x \in [-1,1]$
- Calculate $T_s(x)$

Alice's public key is $(x, T_s(x))$ and private key is s.

3.2.2 Message Encryption

Bob, the sender will obtain Alice's public key and encrypts his message, M using that key.

- Represent number $M \in [-1,1]$
- Generate a large integer r
- Compute $T_r(x)$, $T_{rs}(x)$ and $X = M\,T_{rs}(x)$
- Send ciphertext, $c = (T_r(x), X)$

3.2.3 Message Decryption

Once Alice receives the ciphertext, she can decrypt it by performing the following steps.

- Use private key s to calculate $T_{sr}(x) = T_s(T_r(x))$
- Recover M by calculating $M = \frac{X}{T_{sr}}$

The algorithm is simple as it is based on El Gamal public-key encryption scheme. However, the software implementation is not easy as it requires careful planning to put the algorithm into a programming language desired.

3.3 Public Key Encryption Scheme Based on Logistic Map

The following steps for a fully functional public key encryption scheme are based on the work in [2].

3.3.1 Key Generation

The following action to generate a set of keys is done by the recipient, in this example is Alice.

- Generate a large integer s
- Select a random number $x \in [0,1]$
- Calculate $\varphi_s(x)$

Alice's public key is $(x_0, \varphi_s(x))$ and private key is s.

3.3.2 Message Encryption

Bob, the sender will obtain Alice's public key and encrypts his message, M using that key.

- Represent number $M \in [-1,1]$
- Generate a large integer q
- Compute $\varphi_q(x)$, $\varphi_{q+s}(x)$ and $X=M\,\varphi_{q+s}(x)$
- Send ciphertext, $c=(\varphi_q(x),X)$

3.3.3 Message Decryption

Once Alice receives the ciphertext, she can decrypt it by performing the following steps.

- Use private key s to calculate $\varphi_{s+q}(x)= \varphi_s\,(\varphi_q(x))$
- Recover M by calculating $M = \dfrac{X}{\varphi_{s+q}}$

3.3.4 Implementation on MATLAB

Implementation of this scheme in MATLAB faces one major problem – the limitation of maximum 16 decimal places for floating point numbers using long g format. 16 decimal places is sufficient to prove the workability of the scheme and the concept, however it does not provide security. Thus, as it can be seen in codes (refer to Appendix A), the student has created a method which allows virtually unlimited decimal places for the operation. This is done using arrays to represent decimal numbers. For example, 0.125669 is represented as [0 1 2 5 6 6 9], whereby each element of the array corresponds to the decimal digit. Arithmetic operations – such as multiplication, addition, etc. – have to be re-written to ensure the operations affects the array as a whole, rather then individual elements of the array as in usual MATLAB array operations. For example, 0.31 * 0.456 is done such that [0 3 1] multiplies with [0 4 5 6] equals to [0 1 4 1 3 6] (which corresponds to 0.14136), rather than error message returned by MATLAB if using normal array multiplication (*) due to different array dimensions or normal matrix element-by-element multiplication.

This task requires a lot of extra coding, and the codes are based on the format that the decimal place is located after the first element of the array. Thus, if given an array [3 7 6 4 2], it corresponds to 3.7642. The reason on how the student came to this formatting is due to observation of the numeral range of the numbers used in the scheme. All calculated numbers are within 0 and 1 range, except for r, which is within 3.57 and 4 (the chaotic region); and all multiplication or division results never exceeds 4. Large integers, such as s & q, are never involved in arithmetic calculations, only as number of iterations. To simplify further, this MATLAB implementation uses a message representation M between 0 and 1. With these observations, the new array arithmetic operations are simpler and consequently allow decimal places representation more 16 decimal places than allowed in a direct MATLAB representation of floating point numbers.

The scheme begins with initialization by generating x_0, r and s, by using the *randint* built in to MATLAB. As explained above, x_0, r and s are represented in array form. Then, x_0 is mapped to logistic equation, which is done by passing it to a separate function. The rest of the code follows what the scheme lays out on Section 3.4.1, with most operations done using function calling.

For display of a full floating number that is more than 16 decimal places, the student have written a function which will convert the array representation of the decimal number a string output of the floating point number. This function allows the decimal number to be seen clearly rather reading the elements of the array representation of floating number.

CHAPTER 4
RESULTS & DISCUSSION

4.1 MATLAB Implementation of RSA & AES Algorithms (Hybrid System)

The MATLAB implementation of the RSA algorithm is based on the work by Thunyawat Rajatasereekul and Voranon Kiettrisalpipop from Oregon State University. It is then further modified to accommodate the objectives of the project. The program accepts a string and it will generate the public and private key pair. Using the keys, it will encrypt and decrypt the input string and show all the aforementioned steps of RSA algorithm. The modification made to the program is for the purpose of integration with AES encryption. This is to achieve a hybrid system. For the essential codes, refer to **Appendix D**.

The program has a graphical user interface (GUI) and the screenshot is as follows (refer to *Figure 3*):

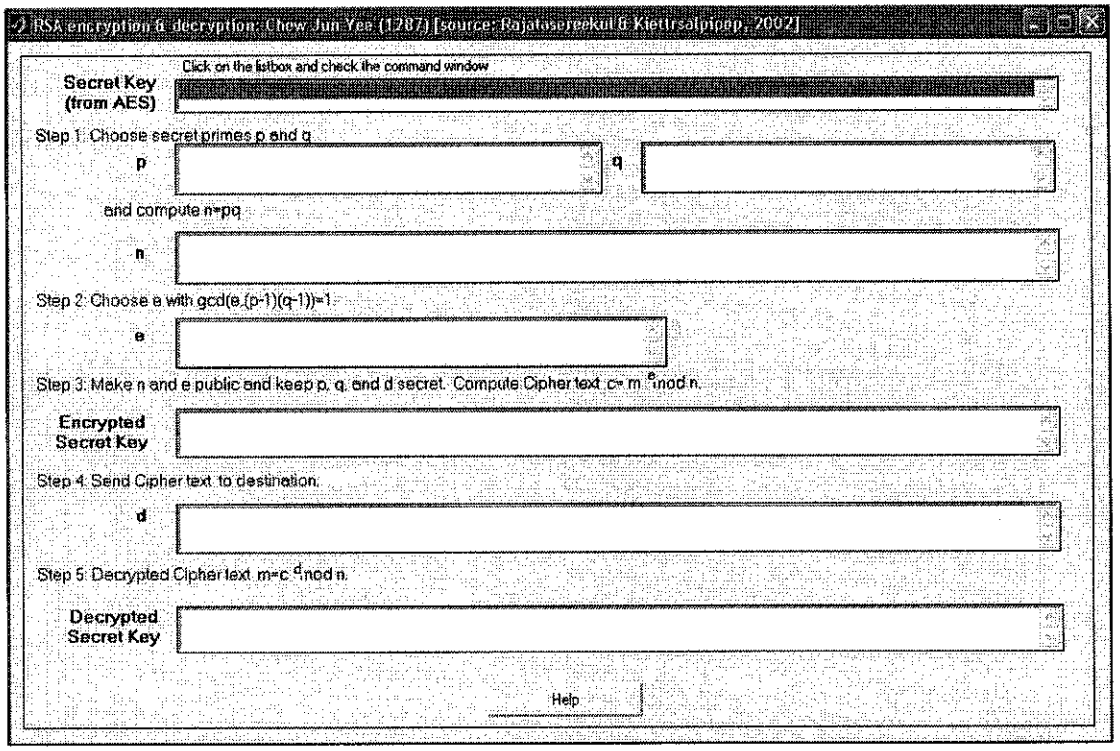


Figure 3 Screenshot of the MATLAB GUI Implementation of RSA Algorithm

The first listbox of the RSA program shows the secret key from AES which is obtained from Ms Easwari's M-files. The program initialized from a function named pro2. This function will then call AES function. The AES part is further discussed in Ms Easwari's report. At the point whereby the AES secret key is generated, it is passed to this function and it will be displayed out. *Figure 4* shows the secret, display in ASCII character format.

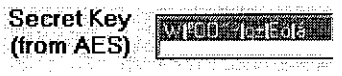


Figure 4 AES Secret Key Output

Then, the program will calculate the prime numbers; p and q. (Refer to *Figure 5* and *Figure 6*)

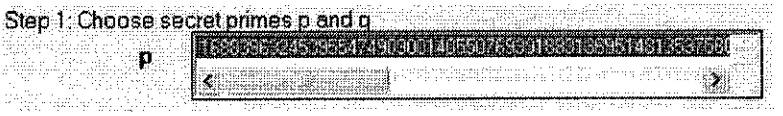


Figure 5 Prime p

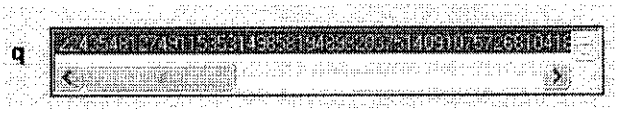


Figure 6 Prime q

The integer n is then calculated. (Refer to *Figure 7*)

Figure 7 RSA Modulus, n

The following step will be computation of e. (Refer to *Figure 8*)

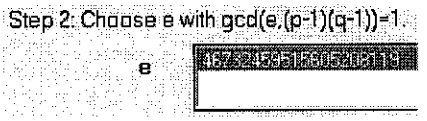


Figure 8 Enciphering Exponent, e

The encryption process ensues to obtain *c*. (Refer to *Figure 9*)

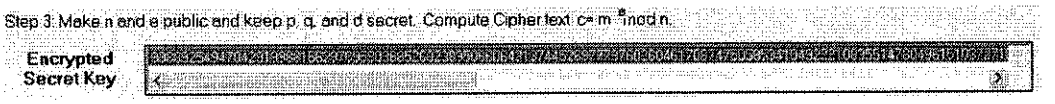


Figure 9 Ciphertext, *c*

After encryption, *c* is sent and the receiver uses his or her private key *d* to decrypt. (Refer to *Figure 10*)

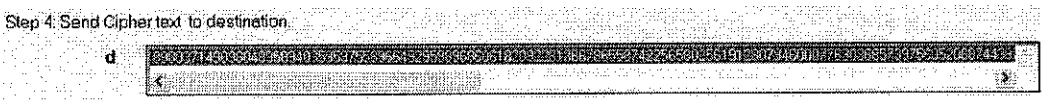


Figure 10 Deciphering Exponent, *d* (Public Key)

The decryption result *m* will be the secret key so that the receiver can use it to decrypt the encrypted message. (Refer to *Figure 11*)

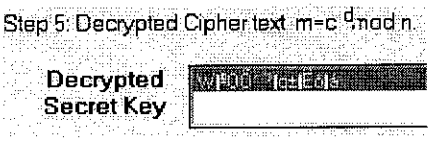


Figure 11 Decrypted Ciphertext, the AES Secret Key

4.1.1 Analysis of MATLAB Implementation

From Figure 12, the encrypted message, c (which is $c \equiv m^e \pmod{n}$) appears to look random and noise-like. A further testing involves auto-correlation of the encrypted message, X. Auto-correlation is defined as:

$$r_{xx}[l] = \sum_{n=-\infty}^{\infty} x[n]x[n-l] = x[n] \otimes x[-l] \tag{12}$$

The result is seen in Figure 13. In a pure white noise, the auto-correlation will produce a delta function, with the peak at 0. In comparison, Figure 13 shows an almost linearly rising and decreasing graph and peaks at 0. This shows that it does not resemble an exact pure white noise, but shows that is a noise-like signal.

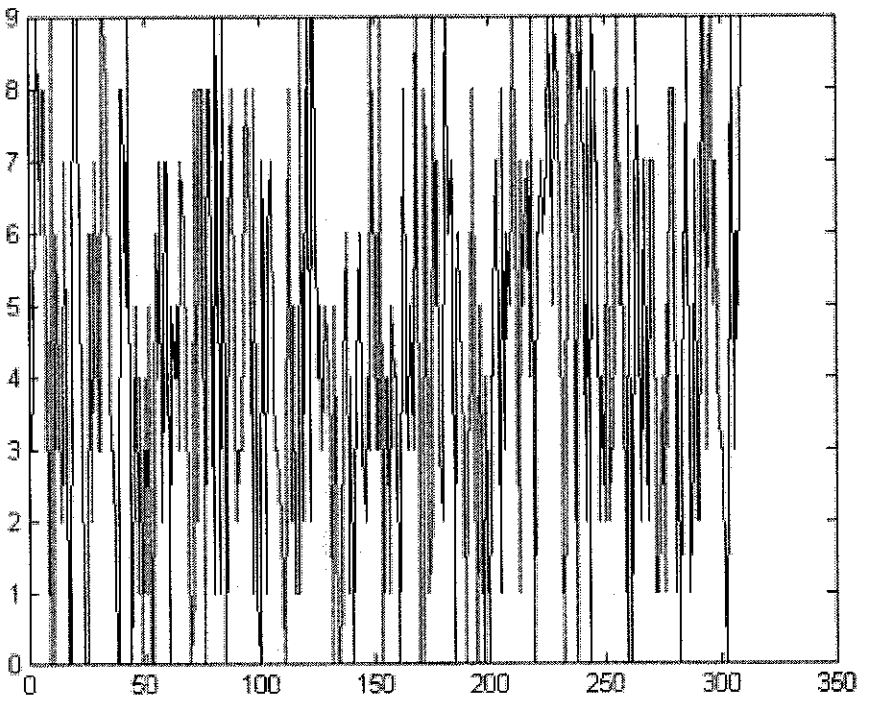


Figure 12 Ciphertext, c

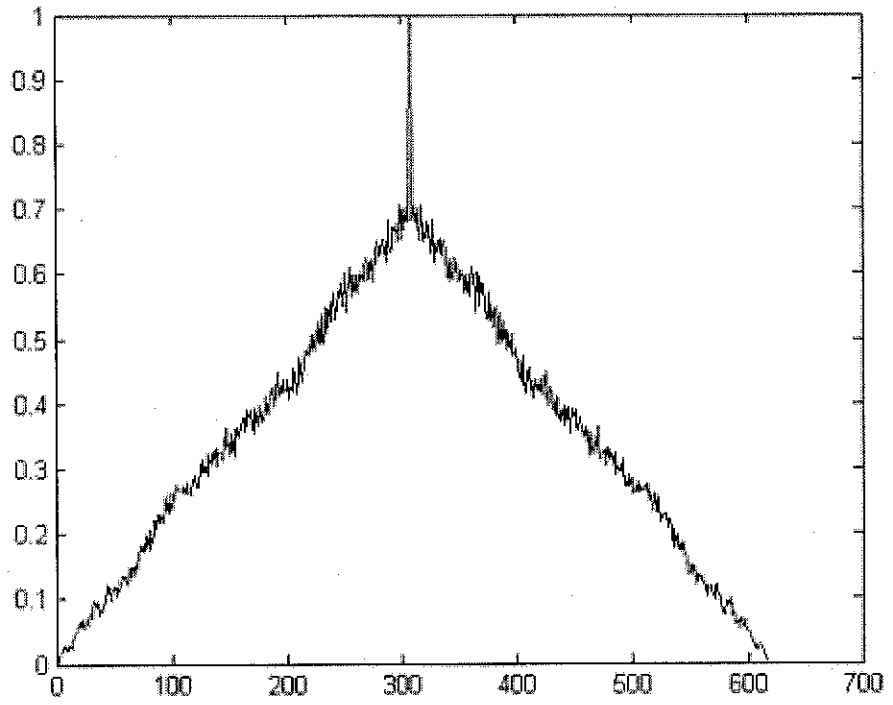


Figure 13 Auto-correlation of normalized ciphertext, c

4.1.2 Precision of the Implementation

This implementation supports up to 1024 bits precision which translates to 309 digits of integer.

4.1.3 Performance Observation

This program gives an overview of how a hybrid system will perform. The strength of a hybrid system comes from both types of cryptographic systems, in this case – RSA and AES. A GUI provides a user-friendly interactivity and understanding of the encryption process. The prominent weakness of the system is the slow encryption process due to:

- Not optimized AES encryption
- GUI processing

Another weakness lies in the generation of keys, in both RSA and AES schemes. MATLAB is not designed to have a secure random number generator.

4.2 MATLAB Implementation of Public Key Encryption Based on Chebyshev Maps

An implementation of the algorithm is carried on MATLAB. The objective is to replicate the result as presented in [2]. The given values are as of below:

$s = 2^{64}36^{36}113^{21}23^{81}59^{48} \approx 2^{910}$

$r = 3^{54}7^{33}41^{55}133^{19}31^{70}$

$x = 0.25749480$

$M = 0.11111111114444444444$

The expected values are as of below:

$T_s(x) = -0.0176128306$

$T_r(x) = 0.9921943793$

$T_r(T_s(x)) = 0.6571609510$

$X = -0.7301788346$

However, the result obtained through MATLAB implementation does not match the expected values. Through thorough checking, it is found out that the precision used in MATLAB cannot match the precision used in [2] which is 2048-bit precision. It uses GNU MP, which is a library for arbitrary precision arithmetic. As discussed in earlier sections, MATLAB allows up to 16 decimal places of a floating point number. For this implementation, the method used in MATLAB implementation of logistic map is not employed because that it was done later. The solution to the limitation of MATLAB decimal places is thus the same as discussed in Section 3.3.4, by using arrays. Due to the fact that Chebyshev Polynomial of the First Kind is a chaotic map

in nature, any small variant in the initial condition will cause a huge difference in the end result. The huge difference is what been observed here.

The solution to this problem is to reduce the size of the values s and r to a smaller number. By doing so, the method proposed in [2] is confirmed working, however at the cost of security. The smaller the number used the easier to break the system. However, the objective here is to prove that the scheme proposed works and the conclusion is positive: the scheme is proven working.

4.3 MATLAB Implementation of Public Key Scheme Based on Logistic Map

A MATLAB implementation has been to done to prove the workability of this scheme. Due to limitation of MATLAB precision, the chosen s, r and M are small. For demonstration sake, any number within the specified range will work. The values for x_0, r, s, q and M are as below:

```
x1 = 0.18
r = 3.76693
s = 3
q = 4
M = 0.45919599
```

The MATLAB results are as of below:

```
xs = 0.92991988510162608549168

tq = 0.24548665582650122810589184588147581336790859366476 8

tqs =

0.79446830435085821822607255490677912950952650021843210973368029535326153359251552367018209387606241892184178989166328144889322447856779719479641233618543577449360746750794582919501486892523025863423341970192027905477 8368

X =

0.36481665954001364686795743066224780008646523569903814887694595956823332964693342248209770007769141975860987333267431327157315861872817341498376141116288400405000882971370401790357575580084134459150286305110878744163521692434432

tsq =

0.79446830435085821822607255490677912950952650021843210973368029535326153359251552367018209387606241892184178989166328144889322447856779719479641233618543577449360746750794582919501486892523025863423341970192027905477 8368

M2 = 0.45919599
```

From the MATLAB results, it can be seen that the public key is

$[x_0, \varphi_s(x), r]$ = [0.18, 0.9299198851016260854916 8, 3.76693]

the private key is s=[3].

The ciphertext, c is

$[\varphi_q(x),X]$ = [0.24548665582650122810589184588147581336790859366476 8,
0.36481665954001364686795743066224780008646523569903814887694595956 8
23332964693342248209770007769141975860987333267431327157315861872817
34149837614111628840040500088297137040179035757558008413445915028630
51108787441635216924344 32].

Alice successfully decrypts the message M.

This MATLAB implementation has proven the workability of the scheme.

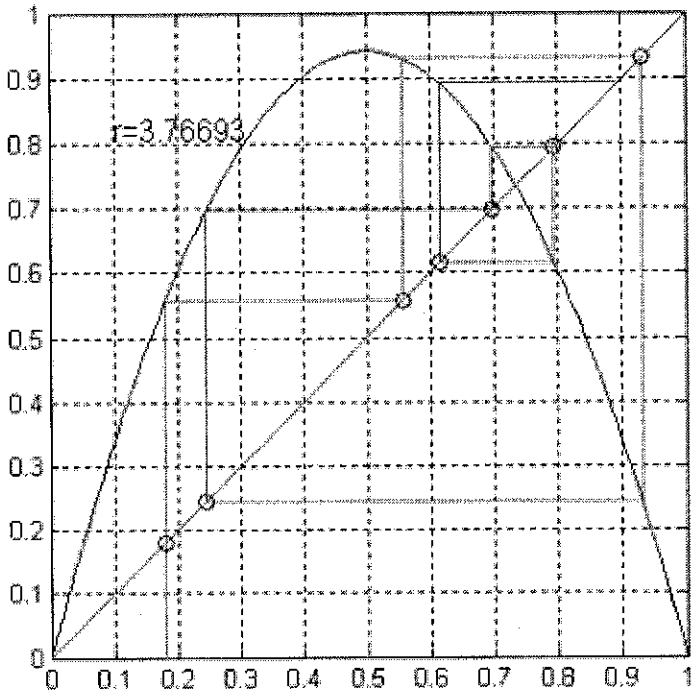


Figure 14 Web diagram using initial x_0=*0.18, r=3.76693* and a total iteration of 7 (s+q)

In Figure 14, it shows the results of the implementation in form of a web diagram for a logistic map. It shows how the initial point being mapped for each iteration. The parabolic curve corresponds to r*x*(1-x) and the linear line corresponds to y=x. The other line shows the trajectory as the initial point being mapped to a new point. Only at a certain range of r (which discussed earlier) will the graph show this chaotic behaviour as seen in this cobweb diagram. Out of this range of r, the behaviour as seen on a cobweb diagram will show the trajectory flies to infinity or the trajectory will stick to a defined orbit and loop on that.

4.3.1 Analysis of MATLAB Implementation

From Figure 15, the encrypted message, X (which is equals to $M \varphi_{q+s}(x)$) appears to look random and noise-like. The importance of checking whether an encrypted message appears noise-like is because the objective of a chaotic encryption is to make the encrypted output appears as noise-like as possible. A further testing involves auto-correlation of the encrypted message, X. Auto-correlation is defined in Equation 12. The result is seen in Figure 16. In a pure white noise, the auto-correlation will produce a delta function, with the peak at 0. In comparison, Figure 16 shows an almost linearly rising graph, peaks at 0 and almost linearly decreasing components of the auto-correlation result. This shows that it does not resemble an exact pure white noise, but shows that is a noise-like signal.

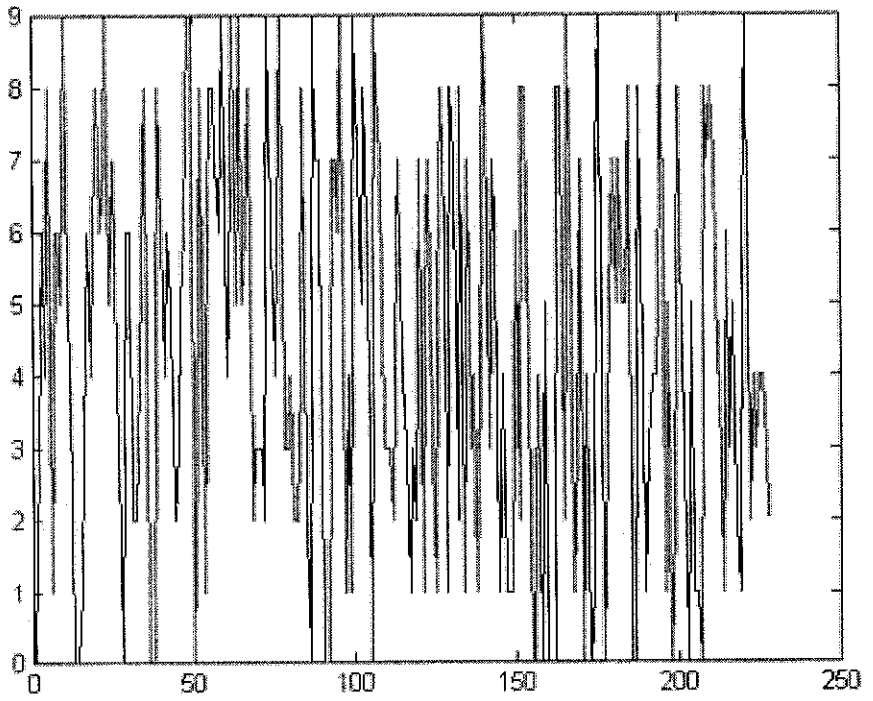


Figure 15 Encrypted message, X

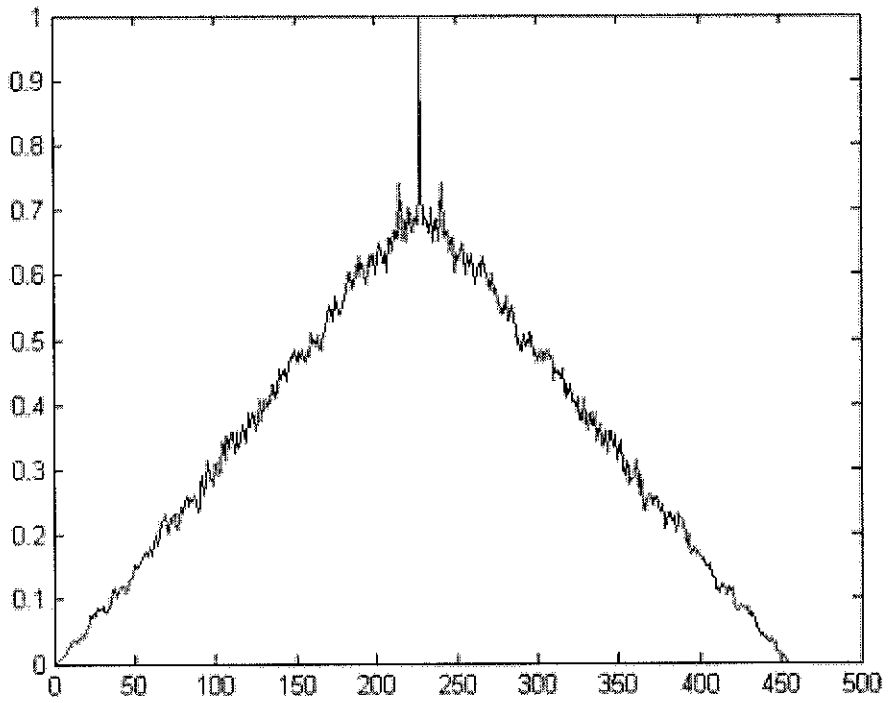


Figure 16 Auto-correlation of the normalized encrypted message, X

4.3.2 Precision of the Implementation

Precision wise, this implementation allows virtually unlimited decimal places representation as opposed to only 16 decimal places as restrained by MATLAB core. The higher the number of decimal places, the more precise the implementation. However, high precision comes with the price of much slower performance (see next section, Section 4.3.3).

4.3.3 Performance Observation

The performance of this implementation varies because it is dependent on the length of x_0, r, s, q and M. The longer the length (the decimal numbers are represented in arrays), the slower is the entire processing due to many calculations involved. An enormous slowdown in processing is seen when 10 decimal places for each x_0, r, s, q and M are set. Thus, on a machine with high processing speed and huge memory, the

implementation allows high precision at a high speed. However, for a normal machine, this is not very practical. For faster processing, precision, and thus security, must be compromised. This implementation, however, can be further enhanced by optimizing the codes or perhaps by using different language. Thus, this implementation is excellent to show the workability of the scheme and to prove it is possible to have higher precision than what is offered in MATLAB itself.

4.4 Comparison between Public Key Encryption Based on RSA Algorithm (MATLAB Comparison) and Logistic Map

The comparison between public key encryption based on logistic map and RSA algorithm is done based on MATLAB platform as a common platform. For RSA algorithm implementation, it was able to achieve 1024-bit of integer for its calculation. However, initially for the logistic map implementation, it can only go up to 16 decimal places of floating point number before MATLAB cuts off and exhibit rounding error. Thus, the student has written MATLAB codes that will allow theoretically unlimited decimal places by using arrays. Yet, this high precision costs performance by slows down the processing time until to an unpractical level. Yet, the codes written has given the scheme as high precision as needed for comparison, only limited by the machine.

On the ciphertext of RSA and its auto-correlation are almost identical to the encrypted message of Logistic Map scheme. Both appear noise-like and their auto-correlation shares a similar shape. It is safe to say that the encrypted message of logistic map scheme has the more or less the same degree of noise-likeliness as the ciphertext of RSA scheme based on the observation on the auto-correlation graph.

CHAPTER 5
CONCLUSION & RECOMMENDATION

5.1 Recommendation

The MATLAB implementation of the RSA algorithm can be improved in terms of efficiency and security. As discussed, both programs are not optimized for speed and security although it follows industrial standards. It was developed for academic demonstration only.

The implementation of the new scheme can be further optimized for performance without compromising security. The current codes provides the means to have higher security level, but is limited by machine factor. With optimization, it can show the scheme can perform encryption at a practical level with security worries at bay.

A thorough cryptanalysis will be needed on the new public key encryption scheme based on logistic map. This is essential for a good encryption scheme.

5.2 Conclusion

The student has learned the fundamentals of cryptography and specifically on RSA algorithm. In addition, the student has ramped up on learning JAVA programming and produced a JAVA implementation of the RSA algorithm. An integrated working MATLAB implementation of RSA algorithm with AES algorithm was also jointly produced with Ms Easwari. This simulates the real life application of both asymmetric and symmetric encryption in a single unit.

A new public key encryption based on logistic map has been proposed. The scheme is based on the work in [2]. MATLAB implementation has proven the workability of the scheme. In addition, it has also solve a major problem faced when implementation due to limitation of MATLAB floating point precision. The scheme can have a higher level of security, comparable with RSA, with only limitation of computing power and memory.

All objectives are met for this project.

REFERENCES

Books/Papers

1. Kocarev; *Chaos-Based Cryptography: A Brief Overview*; IEEE; 2001
2. Kocarev, Tasev; *Public-Key Encryption Based on Chebyshev Maps*; IEEE; 2003
3. Kocarev, Jakimoski, Stojanovski, Parlitz; *From Chaotic Maps to Encryption Schemes*; IEEE; 1998
4. Kennedy, Rovatti, Setti; *Chaotic Electronics in Telecommunications*; Florida, USA; CRC Press; 2000
5. Tsimring, Tenny; *Security Issues in Chaos-based Communication and Encryption*; 2003; UCSD, UCLA, Standford
6. Nash, Duane, Joseph, Brink; *PKI: Implementing & Managing E-Security*; California, USA; Osborne / McGraw-Hill; 2001
7. Mollin; *RSA & Public-Key Cryptography*; USA; Chapman & Hall / CRC; 2003
8. Mel, Baker; *Cryptography Decrypted*; New Jersey, USA; Addison Wesley; 2001
9. Naccache, Paillier; *Public Key Cryptography – 4th International Workshop on Practice and Theory in Public Key Cryptosystems*; PKC 2002; Germany; Springer; 2002
10. Kohda, Fujisaki; *Jacobian Elliptic Chebyshev Rational Maps*; Physica D, 148:242-254; 2001

Websites

1. Interactive Chaos (http://order.ph.utexas.edu/standardmap/index.html)
2. Background of Chaos (http://www.math.arizona.edu/~lega/UG/1998-1999/sync2/node2.html)
3. Logistic Map - Wikipedia (http://en.wikipedia.org/wiki/Logistic_map)
4. Logistic Map – Wolfram Research (http://mathworld.wolfram.com/LogisticMap.html)
5. RSA Security (http://www.rsasecurity.com/)
6. PKI Whitepapers (http://www.pkiforum.org/whitepapers.html)
7. The PKI Page (http://www.pki-page.org/)
8. An Introduction to Encryption & PKI (http://www.itsecurity.com/papers/upaq.htm)

9. Data Security & Cryptography - Oregon State University (http://islab.oregonstate.edu/koc/ece575/)
10. Cryptography – RSA (http://pajhome.org.uk/crypt/rsa/)
11. Open Source PKI Book (http://ospkibook.sourceforge.net/)
12. The Java Developers Almanac 1.4 – java.security (http://javaalmanac.com/egs/java.security/pkg.html)
13. Topics in Pure and Experimental Mathematics: Number Theory and Cryptography Module (http://www.ma.umist.ac.uk/avb/117topics.html)

APPENDIX A
PUBLIC-KEY ENCRYPTION BASED ON LOGISTIC MAP (MATLAB IMPLEMENTATION)

```
%logistic_pk_scheme.m
%Public Key Implementation Based on Logistic Map
%Chew Jun Yee
%Universiti Teknologi PETRONAS
%5/4/2004

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long g                   %To set the output format
clear                           %To clear the workspace data

%initial value
ra1=randint(1,1,[2 5]);
x1=0;
x1=cat(2,x1,randint(1,ra1,[0 9]));
x1_dec=arr2dec(x1)

%constant
ra2=randint(1,1,[2 5]);
r=3;
r=cat(2,r,randint(1,1,[6 9]));
r=cat(2,r,randint(1,ra2,[0 9]));
r_dec=arr2dec(r)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Receiver: Alice
%Generate public & private keys

s=randint(1,1,[2 5]);

xs=mapping(x1,r,s); %find xs
xs_dec=arr2dec(xs)

%private key: [s]
%public key: [x(1),xs,r]
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Sender: Bob
%Obtain public key

q=randint(1,1,[2 5]);
t1=x1;          %info from public key

tq=mapping(t1,r,q); %find tq
tq_dec=arr2dec(tq)

%encryption
tqs=xs;          %info from public key

tqs=mapping(tqs,r,q);
tqs_dec=arr2dec(tqs)

%message
ra2=randint(1,1,[8 13]);
M=0;
M=cat(2,M,randint(1,ra2,[0 9]));
M_dec=arr2dec(M)

X=mul(M,tqs);
X_dec=arr2dec(X)

% acorr_cipher=xcorr(X);
% plot(acorr_cipher)

%ciphertext=[tq,X]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Alice received ciphertext

tsq=tq;         %info from ciphertext

%decryption
tsq=mapping(tsq,r,s); %find tsq
tsq_dec=arr2dec(tsq)

M2=division(X,tsq);          %info from ciphertext
M2_dec=arr2dec(M2)       %message successfully recovered
```

```
%mapping.m
function y = mapping(x,r,trans)

% Usage
% Inputs: r - constant
%         trans - number of iterations before outputting
%
% based on the work by C. Savage
%
% modified by Chew Jun Yee
% 3/4/2004

for i = 1:trans-1        % transient iterations
    temp1=mul(r,x);
    temp2=subtraction(x);
    x = mul(temp1,temp2);           % logistic map
end

return
```

```
%mul.m
function y = mul(array1,array2)

lar1=length(array1);
lar2=length(array2);

if lar2>lar1
    temp_array=array2;
    array2=array1;
    array1=temp_array;
    lar1=length(array1);
    lar2=length(array2);
    lar=lar1;
else
    lar=lar1;
end
```

```
count=lar;
count1=lar1;
count2=lar2;

leng=lar1+lar2-1;

%initialization
for k=1:leng
    arrmul(k)=0;
end

result=arrmul;

for m=1:lar2
    arr2=array2(count2);
    A{1,m}=multiplication(array1,arr2,arrmul,(count+lar2-1));
    count2=count2-1;
    count=count-1;
end

for q=1:lar2
    result=addition(result,A{1,q});
end

y=result;

return
```

```
%multiplication.m
function array_return = multiplication(array1, array2, arrmul,
count)

lar1=length(array1);
lar2=length(array2);

%mulitplication

if lar2>lar1
    lar=lar2;
else
```

```
    lar=lar1;
end

% count=lar;
count1=lar1;
count2=lar2;

%initialization
% for k=1:(count1+count2-1)
%     arrmul(k)=0;
% end

for n=1:lar

    if count>0

        arrmul(count)=(array2(count2)*array1(count1))+arrmul(count);

        if arrmul(count)>=10
            if count>1
                arrmul(count-
1)=floor(arrmul(count)/10)+arrmul(count-1);
            end
            arrmul(count)=rem(arrmul(count),10);
        end

    end

    count=count-1;
    count1=count1-1;

end

array_return=arrmul;

return
```

```
%addition.m
function arradd = addition(arr1,arr2)
```

```
arradd=arr1+arr2;

arrlen=length(arradd);

arrcount=arrlen;

for p=1:arrlen

    if arradd(arrcount)>=10
        if arrcount>1
            arradd(arrcount-
1)=floor(arradd(arrcount)/10)+arradd(arrcount-1);
        end
        arradd(arrcount)=rem(arradd(arrcount),10);
    end

    arrcount=arrcount-1;

end

return
```

```
%subtraction.m
function y = subtraction(array)

% 1-x

k=length(array);

result=0;

for m=2:k
    result=cat(2,result,[9]);
end

result(k)=result(k)+1;

y=result-array;
```

```
% division.m
```

```
function y = division(array1, array2)

format long g

arr1len=length(array1);
le1=arr1len;
arr2len=length(array2);
le2=arr2len;

arr1rep=0;
arr2rep=0;

for k=1:arr1len
    arr1rep=arr1rep+(10^(k-1))*(array1(le1));
    le1=le1-1;
end

for m=1:arr2len
    arr2rep=arr2rep+(10^(m-1))*(array2(le2));
    le2=le2-1;
end

y=arr1rep/arr2rep;
```

```
%arr2dec.m
function dec = arr2dec(array)
format compact

n=length(array);
dec='';

for k=1:n
    if k==2
        dec=strcat(dec,'.');
    end
    temp=num2str(array(k));
    dec=strcat(dec,temp);
end
```

APPENDIX B
MATLAB IMPLEMENTATION OF RSA ALGORITHM WITH AES

Partial RSA only, other parts are used for GUI. For AES, please refer to Ms Easwari's report.

```
% Computation core of RSA algorithm

function [c,out,p,q,n,e,d] = rsacore(val)

c=[];
out=[];
p=[];
q=[];
n=[];
e=[];
d=[];

format long g

cut_length = 50; % Chunk of characters to be transmitted in one time
bit = 1024; % Number of bit for RSA modulus n
e_bit = 64; % Number of bit for e

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Computation of p q e n d %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

X = bit/2;
X = num2str(X);

maple('x:=',X);
maple('y:=2^x');
maple('z:=y*2');
maple('pp:=rand(y..z)');

%%%%%%%%%%%%%%%%%%%%
% Generate p and q %
%%%%%%%%%%%%%%%%%%%%
```

```
temp = randint(1,1,3000);
for j = 1:temp
    maple('pp()');
end
maple('p:=nextprime(pp())');
maple('q:=nextprime(pp())');

%%%%%%%%%%%%%
% Compute n %
%%%%%%%%%%%%%
maple('n:=p*q');

%%%%%%%%%%%%%
% Compute e %
%%%%%%%%%%%%%
maple('temp:=(p-1)*(q-1)');
TEMP = 0;
while TEMP ~= '1'
    XX = num2str(e_bit);
    maple('xx:=',XX);
    maple('xx:=2^xx');
    maple('zz:=rand(xx..(10*xx))');
    maple('e:=nextprime(zz())');
    TEMP = maple('gcd(e,temp)');
end

%%%%%%%%%%%%%
% Compute d %
%%%%%%%%%%%%%
maple('d:=e&^(-1) mod temp');

p = maple('p');
q = maple('q');
n = maple('n');
e = maple('e');
d = maple('d');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      End of Intialization      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Make n and e public                     %
% Encryption of m using available n and e %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m = val;
[m_int2 padd] = mesgcut(m,cut_length);
m_int2 = double(m_int2);
m_char = intconcat(m_int2);
[s1 s2] = size(m_char);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Repeatedly tramsmiss of 50(cut_length) characters message chunk %
% from verylong message                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c = [];
out = [];
for j = 1:s1
    %%%%%%%%%%%%%%%%%%%%%%%
    % Compute Cipher text %
    %%%%%%%%%%%%%%%%%%%%%%%
    maple('m:=',m_char(j,:));
    maple('c:=((m&^e) mod n)');
    c_temp = maple('c');
    c = [c c_temp];

    %%%%%%%%%%%%%%
    % Decryption %
    %%%%%%%%%%%%%%
    maple('m:=((c&^d) mod n)');
    m_out = maple('m');

    if length(m_out) == ((cut_length*3)-1)
        m_out = ['0' m_out];
    elseif length(m_out) == ((cut_length*3)-2)
        m_out = ['0' '0' m_out];
    end

    l = 1;
```

```
    m_out_int = [];
    for k = 1:length(m_out)/3
        m_out_temp = [m_out(l) m_out(l+1) m_out(l+2)];
        m_out_int_temp = str2num(m_out_temp);
        m_out_int = [m_out_int m_out_int_temp];
        l = l+3;
    end
    out = [out char(m_out_int)];
end

out = out(1:((s1*cut_length)-padd));
global decrypted_key
decrypted_key= double(out)
```