Master Thesis

Yamen Cryptosystem: An Enhanced RSA by Using Rabin Algorithm and Huffman Coding

Author:
Abdallah Karakra

Supervisor:
Dr. Ahmad Alsadeh

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in Computing

February 2015

جامعة بيرزيت

BIRZEIT UNIVERSITY

Yamen Cryptosystem: An Enhanced RSA by Using Rabin Algorithm and Huffman Coding. By Abdallah Karakra

Approved by the thesis committee:

Dr. Ahmad Alsadeh, Birzeit University

Dr. Abdellatif Abu-Issa, Birzeit University

Dr. Radwan Tahboub, Palestine Polytechnic University (External)

Date Approved:

Declaration of Authorship

I, Abdallah Karakra, declare that this thesis titled, 'Yamen Cryptosystem: An Enhanced RSA by Using Rabin Algorithm and Huffman Coding' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at Birzeit University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

Abstract

Yamen Cryptosystem: An Enhanced RSA by Using Rabin Algorithm and Huffman Coding. By Abdallah Karakra

Today, RSA algorithm is the most widely used public-key cryptosystem around the world. It is used for security in everything from online shopping to cell phones. However, the basic RSA is not semantically secure, i.e., encrypting the same message more than once always gives the same ciphertext. For this reason, the basic RSA is vulnerable to set of indirect attacks, such as *known plaintext*, *chosen plaintext*, *timing*, *common modulus*, and *frequency of blocks* (FOB) attacks. To the best of our knowledge no one points to *FOB* attack against RSA. Moreover, RSA is known to be much slower than the standards symmetric key encryption and it does not used for encrypting large data.

In this thesis, we design and implement a swift and secure variant of RSA based on Rabin and Huffman coding called *Yamen* cryptosystem to solve aforementioned limitations of the basic RSA. A new additional randomization component Y is added in *Yamen* cryptosystem. This component is encrypted by *Rabin* algorithm to improve the security level of RSA against the *indirect attacks* and make RSA semantically secure. Moreover, *Yamen* makes the factorization problem harder against *brute force attack*, since the attackers need to break the factorization of large numbers for both RSA and Rabin. Besides, employing Huffman coding compression in *Yamen* prevents *FOB* attack and speeds up the execution time for the *Yamen*.

Yamen cryptosystem comes with three sensitive enhancement factors comparing with basic RSA. These factors are *security*, *execution time* and the *Size of the ciphertext. Yamen cryptosystem* is semantically secure. *Yamen* generates different ciphertexts for the same message. Also, our testing results over set of file sizes of 1MB, 2BM,.., to 10 MB show that *Yamen* cryptosystem is faster than basic RSA by 45% in encryption process and 99% in decryption process. Also, we found that RSA system increases the size of ciphertext by 1% compared to the original file size, while *Yamen* cryptosystem reduces the size of ciphertext by 54% from its original sizes. This reduction depends on the number of occurrences of the symbols inside the file.

الملخص

نظام التشفير يامن:
آر أس أيه المُعزز باستخدام خوارزمية رابين وترميز هافمان

اعداد : عبد الله كراكرة

اليوم، يعد نظام التفشير ال آر أس أيه مهم للغاية، فهو آلية تشفيركثيرة الإستخدام في جميع أنحاء العالم، ويستخدم في شتى المجالات ابتداء من التسوق عبر الانترنت إلى الهواتف المحمولة. ولكن هناك بعض القيود على نظام التفشير ال آر أس أيه، ومثال ذلك أن نفس الرسالة إذا شفرت أكثر من مرة بنفس المفتاح فإن النص المشفر في هذه الحالة يكون نفسه، لهذا السبب فان نظام التفشير آر أس أيه يصبح عرضة لبعض الهجمات غير المباشرة مثل العامل المشترك، النص غير المشفر المعروف، واختيار النص غير المشفر، وهجوم الوقت، وتكرار اللبنات -حسب معرفتنا لم يشر أحد مسبقا لهجموم تكرار اللبنات- وأيضا من المعروف أن نظام التفشير ال آر أس أيه أبطىء بكثير من انظمة التشفير التماثلية.

في هذه الرسالة نعرض نظام تشفير سريع وآمن من نظام التفشير ال آر أس أيه بالاعتماد على نظام تشفير رابين وترميز هافمان يسمى **بنظام التفشير يامن** لحل القيود والهجمات التي يعاني منها نظام آر أس أيه، فقد تم اضافة عنصر عشوائي على نظام التفشير ال آر أس ايه، هذا العنصر العشوائي يُشفر بواسطة نظام تشفير رابين لرفع مستوى أمان نظام التفشير ال آر أس ضد الهجمات غير المباشرة وجعله آمن دلاليا. علاوة على ذلك، **فنظام التفشير يامن** جعل من تحليل العدد إلى عوامله أكثرصعوبة، حيث يحتاج المهاجم إلى تحليل الأعداد إلى عواملها في كلا الخوارزميتين آر أس أيه و رابين لكسر نظام التفشير يامن. بجانب ذلك تم توظيف ترميز هافمان لمنع الهجمات التي قد تنتج عن تكرار اللبنات وتسريع وقت التنفيذ في نظام التفشير يامن.

فنظام تشفير يامن يحسن ثلاثة عوامل حساسة مقارنة مع آر أس أيه هي: الأمان و وقت التنفيذ وحجم النص المشفر. فنظام التشفير يامن هو آمن دلاليا، حيث يقوم بتوليد نصا مشفرا مختلفا لنفس الرسالة. بعد اختبار نظام التفشير يامن على عدة ملفات مختلفة الأحجام ابتداء من حجم واحد ميجابايت حتى حجم 10 ميجابايت أظهرت النتائج أن **نظام التفشير يامن** أكثر سرعة من ال آر أس أيه بحوالي 45% في عملية التشفير، وتقريبا 99% في عملية فك التشفير، كذلك وجدنا أن نظام التفشير آر أس آيه يزيد من حجم النص المشفر مقارنة بالنص الأصلي بـ 1% تقريبا. بينما **نظام التفشير يامن** يقلل من حجم النص المشفر مقارنة بالنص الأصلي بـ 54% تقريبا ، وهذه النسبة تعتمد على عدد الرموز المكررة داخل النص الأصلي.

Acknowledgements

I would like to express my sincere gratitude to my advisor Dr. Ahmad Alsadeh for his continuous support of my master thesis. He was very patience, motivated and always provided his excellent guidance in helping me to write the thesis.

My sincere thanks also go to Dr. Derar Eleyan, Mr. Hafeth Barghouthi for the time they spent sharing their wealth of knowledge with me. Also, I would like to convey my really thanks to Mr. Nael Qaraeen for his encouragement. He is the first one encourage me to enroll in a master program. Also, I would like to acknowledge Dr. Walid Al-Ahmad for his encouragement, he pushes me up to do my thesis in this topic.

A huge thanks go to Dr Adel Al-tawel for his valuable guidance, in Research Methodologies course, on how to write a literature. Many thanks to Dr. Iyad Tumar for his encouragement to continue in this topic, and for Mr. Mahmoud Ghannam for helping me to understand some mathematical concepts related to the numbering theory.

I would like to thank my thesis committee members (Dr. Ahmad Alsadeh, Dr. Abdellatif Abu-Issa and Dr. Radwan Tahboub) for their guidance, comments, ideas and feedback through the thesis discussion have been absolutely invaluable.

I am deeply and forever indebted to my parents and my family for their love, support and encouragement throughout my entire life. I am also very grateful to my brothers and sisters.

I would like to acknowledge my best friends; Fadi Azzam, Ali Al Jadaa, Saed Taha and Mohammad B'airat for helping me for collecting some information which is used in my thesis and they provide me with their valuable feedback.

Contents

List of Figures

List of Tables

Abbreviations

AES	**A**dvanced **E**ncryption **S**tandard
ASCII	**A**merican **S**tandard **C**ode For **I**nformation **I**nterchange
CBC	**C**ipher **B**lock **C**haining
CFB	Cipher Feedback
CMA	**C**ommon **M**odulus **A**ttack
CPA	**C**hosen-**P**laintext **A**ttack
CRT	**C**hinese **R**emainder **T**heorem
CTR	Counter
DES	**D**ata **E**ncryption **S**tandard
EOF	**E**nd **o**f **F**ile
FOB	**F**requency **o**f **B**lock
gcd	**g**reater **c**ommon **d**evisor
KPA	**K**nown **P**laintext **A**ttack
PGP	**P**retty **G**ood **P**rivacy
RSA	Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman
TA	**T**iming **A**ttack

Symbols

KU	Public-Key
KR	Private-key
$E(M,\ KU)$	Encrypt the message M by using the public-key
$D(C,\ KR)$	Decrypt the ciphertext C by using the private-key
M	Plain Message
C	Ciphertext
e	Public exponent
d	Secret exponent
R	Random component
p	Large prime number
q	Large prime number
N	RSA or Rabin factorization
$\Phi(N)$	Totient function
mod	Modules
KU_B	Bob public-key
KR_B	Bob private-key
C_B	Ciphertext whic is encrypted by Bob publick-key
Y	Randomized Component, refers to the **Y**amen Abdallah Karakra
$C'(Mixture)$	Ciphertext of the header file blinded with randomized component Y
H	Heder file
B	Binary file
B'	Binary file blinded with randomized component Y
$E(H,\ KU_B\ )_{RSA}$	Encrypt H using Bob public-key by the RSA cryptosystem
$E(Y,\ KU_B\ )_{Rabin}$	Encrypt Y using Bob public-key by Rabin cryptosystem

To my parents

To my lovely family, my wife and my son
To my brothers and sisters

Chapter 1

Introduction

Suppose Alice wants to protect her home, she could purchase high-quality locks and install some of them on her door, but unfortunately she forgot to close and to secure the house windows. A fool robber may consume times to unscrew the hinges and in the result may be failed or caught by police. On the other hand, a clever robber might simply do not spends the effort to open the locks, just climbs to the open window, steals what he wants and runs away with spoils. This example of indirect attack on household security can be happened against cryptographic algorithms in the world of network and information security.

There are two main types of attacks that can be carried out against cryptographic encryption algorithms. First type is called *direct attack* or *brute force attack*. In direct attack, the attacker tries to get the secret keys which have been used in encryption process by trying all possible combinations to find these keys. The second type is *indirect attacks* where the attackers try to bypass the encryption algorithm using indirect ways for getting the original message without generating the brute force attacks. The attackers try to exploit the weaknesses of the algorithm implementation or depending on other information available to them. Such indirect attacks are what actually happened against the **R**ivest, **S**hamir and **A**dleman (RSA) algorithm [1].

RSA is the underplaying cryptosystem for many network security protocols, such as Secure Sockets Layer (SSL) [2], Transport Layer Security (TLS) [3], Secure Shell (SSH) [4] and IPsec [5]. In addition, it is also used in several authentication agents, such as Windows 2008 server [6] and UNIX [7] operating systems. Furthermore, it is used in some database systems, such as Oracle [8] and MySQL [9]. Therefore, if the RSA cryptosystem is broken, all of these critical applications will be vulnerable to security and privacy violation.

The security of *RSA* system depends on the difficulty of factorizing large numbers. In fact, the prime factorization of large numbers is known as a hard problem in the state of the art. Until now, the *indirect attacks* are the feasible attacks against RSA. Therefore, this thesis presents some types of indirect attacks against RSA, and the way to prevent or at least mitigate them.

For simplicity and illustration purposes, we will use different examples. We will use three known characters in the world of the security; Alice, Bob and Eve. Alice and Bob are the authorized users, and they want to communicate with each other securely. Eve is an adversary and she wants to disclose the messages transferred between Alice to Bob. In other word, Alice and Bob try to find a way for exchanging messages between themselves in a secure manner to avoid people from revealing it. However, Eve tries to invest all of the available information in her hand to exploit the scramble messages transferred between Alice and Bob.

The rest of this chapter organized as follows: Section 1.1 presents the statement of the research problem and the proposed solution. Section 1.2 discusses the contribution of thesis. Finally, Section 1.3 shows the thesis organization structure.

1.1 Statement of the Problem

Most of Internet applications and security protocols depend on *RSA* cryptosytem. It is used in most of electronic commercial communications [10]. It is used for protecting emails and traffic of the web. In addition to securing some of the wireless devices and network resources [11].

RSA is not *semantically secure* [12]. That is mean, encrypting the same message more than once always gives the same ciphertext [13]. This weakness can lead to many attacks against RSA algorithm. Such these attacks, *Common Modulus* [14–16], *Known plaintext*, *Chosen-plaintext* [17], and *Timing* [18] attacks. Moreover, *RSA* does not solve blocks redundancy in the message before the encryption, hence *RSA* suffers from *Frequency of Block* (FOB) attack. If we ignore these attacks, most of the applications that use RSA will be exposed for attacks, and the security and privacy will be violated.

Another problem that can be consider as a drawback of *RSA* cryptosystem is intensive *computation time* [19]. RSA encryption and decryption processes is very time consuming comparing to symmetric key encryption like *DES* (Data Encryption Standard) [20], RSA is slower than DES by more than 1000 times [21]. Therefore, the usage of *RSA* is limited for symmetric keys encryption instead of encrypting the entire data. The data itself is encrypted by using symmetric encryption like DES and this make the encryption and decryption process more complex, Diffe states that “the restriction of public-key cryptography to key management and signature applications is almost universally accepted” [22].

Therefore, this thesis answers the following research questions:

1. How to make RSA algorithms semantically secure to thwart or at least weaken indirect attack?
2. How to enhancing the execution time for the enhancement RSA algorithm (*Yamen cryptosystem*) to be competitive comparing with the basic RSA?
3. How to extend the usage of RSA system for encrypting large data within feasible and acceptable time?

1.2 Contribution of Thesis

The fact that RSA is time intensive and semantically not secure makes RSA usage is limited to key exchange and digital signature. These limitations motivate us to find solution to solve or at least mitigate these drawbacks. We summarize our contributions as follows:

- Augment RSA cryptosystem to make it more secure against *Common Modulus*, *Known plaintext*, *Chosen-plaintext*, and *Timing attacks* [23]; by adding a randomized component to the *basic RSA* and encrypting this component by another public-key cryptosytem called *Rabin* [24]. This makes RSA semantically secure against these attacks by generating different ciphertexts for the same message. Also, this makes RSA stronger against *brute force attack* (direct attack), since attackers need to break the factorization of large numbers for both RSA and Rabin. Consequently, the attackers will require longer time than before.
- Thwart the *Frequency of Block* (FOB) attack by using *Huffman coding*. Huffman coding compress data in away to reduce the redundancy in the message, which helps to prevent this attack.
- Enhance the *execution time* for the *Yamen cryptosystem* comparing with the basic *RSA* by using *Huffman coding*. By encrypting part of the message, while blinding the other part of the same message using *XOR* operator, since *XOR* operator is always faster than multiplication, division and addition [25]. *Yamen cryptosystem* is faster than basic RSA by about 45% in encryption process and around 99% in decryption process.

- Reduce the sizes of large data using *Huffman* coding, it feasible to use *Yamen cryptosystem* for encrypting large files. *Yamen cryptosystem* reduces the size of encrypted file by 54% from the original sizes. This reduction depends on the number of occurrences of the symbols inside the file. While RSA system increases the size of ciphertext by approximately 1% compared to the original file size.

1.3 Organization of the Thesis

The thesis is structured in the following way:

Chapter 2: presents the important definitions and concepts of public key cryptography that are related to the remaining chapters of this thesis. Firstly, we present mathematical preliminaries that are necessary for understanding the public key cryptosystems. Then we discuss the encryption techniques and classify them based on the number of keys. First, symmetric key encryption and the key distribution problem. Second, public-key encryption. Where, we discuss RSA and Rabin cryptosystems, and what are the differences between them.

Chapter 3: presents the literature review about the attacks against RSA and the proposed approaches to solve or to mitigate them. We classify these approaches into three categories based on its *techniques*. First category is the *Dice solution*. Second category is called *Dice solution Follower*. Third category is the *Hungry mouse solution*. After that, we illustrate the limitations of these solutions.

Chapter 4: Presents new cryptosystem called *Yamen*, which is an enhanced version of RSA. We introduce the combination of RSA and Rabin cryptosystems for generating *Yamen cryptosystem*. We present amendment components to the basic RSA; using Huffman coding and random component to build *Yamen cryptosystem*. Also, we discuss the design model and the implementation of *Yamen cryptosystem*. After that, we present *Huffman coding* and *XOR operation* to speeding up *Yamen* system. Conclusion is given at the end of the chapter.

Chapter 5: In this chapter, we present three sensitive factors enhanced by *Yamen cryptosystem* comparing with basic RSA. These factors are *Security*, *Execution time* and the *Size of the ciphertext*. The experiments applied to get the evaluation results are summarized in section 5.1, then the finding results are given in section 5.2. Section 5.3 states the added value of *Yamen* over the basic RSA. Conclusion is given at the end of the chapter.

Chapter 6: This chapter summarizes the main idea of this thesis. Difficulties and recommendations are presented in section 6.2 and section 6.3 consequently. Outlook is presented in section 6.4.

Chapter 2

Foundation

Cryptography is the science or art for protecting data [26, 27]. For a thousand of years throughout the history, cryptography plays an important role for securing communication between people. Currently, cryptography is used for securing military, commercial and private communications [28]. Cryptography enables us to store or to transmit sensitive data over insecure channels by scrambling it [29]. The orginal data before scrambling is called *plaintext* and the scrambled data is called *ciphertext*. The process of scrambling the *plaintext* to be irretrievable to anyone other than intended persons is called encryption process. On the other hand, converting encrypted data (*ciphertext*) to its original plaintext is called decryption process.

Cryptographic techniques can be classified into two groups based on the number of used keys in encryption and decryption processes; *Symmetric (shared or single) key* encryption and *Asymmetric (public key)* encryption [30]. Symmetric key encryption uses a single key for encryption and decryption process, while asymmetric encryption uses two keys, one for encryption and the other one for decryption.

In symmetric encryption both parties must share the same keys. In other word, they must exchange the keys to using in encryption. Exchange the keys between the two parties is not an easy task, especially if the two parties far away. So, the asymmetric encryption introduced to solve the key distribution problem, since no need for the two parties to share secret keys, all communications involve only public keys, and no private key is ever exchanged or shared.

In section 2.1, we present the mathematical preliminaries that is used in public key algorithms. Then, we discuss the encryption techniques in section 2.2. After that, we explain two public keys algorithms, called RSA and Rabin cryptosystems in section 2.3. Finally, we conclude the chapter in section 2.4

2.1 Mathematical Preliminaries

Definition 1 (Prime number)

An integer number is called a prime number if it can be divided only by 1 and itself, where that number is greater than 1. In other words, Prime number is a number whose only factors are 1 and itself, where the number is greater than 1.

Example: 5 is a prime number, since it is only divided by 1 and 5.

Definition 2 (Relatively prime)

Two numbers are called relatively prime (also called mutually prime or coprime) if they do not share any factors other than 1, such as, the integers a and b are relatively prime if $gcd(a, b) = 1$.

Example: 3 and 5 are relatively prime, since the only common factor for these integers is 1. $gcd(3, 5) = 1$.

Definition 3 (Euler's Totient Function ϕ)

Euler's Totient function is defined as $\phi(n)$, where $\phi(n)$ is the number of positive integers less than n and relatively prime to n.

Example1: Let the integer number n is 8, the positive integers less than 8 and relatively prime to 8 is $\{1, 3, 5, 7\} \Longrightarrow \phi(8)$=4.

Example2 : let the number n is 7, the positive integers less than 7 and relatively prime to 7 is $\{1, 2, 3, 4, 5, 6\} \Longrightarrow \phi(7)$=6.

Euler's Totient Function ϕ properties:

- If the integer n is a prime number then ϕ(n)=n-1.
  Example: ϕ(7)=7-1=6.
- Euler's Totient function is a multiplicative function, that is, if there are two integers such as, a and b are relatively prime, then $\phi(\text{ab}) = \phi(\text{a}) \times \phi(\text{b})$ [31].
  Example: $\phi(3\times5) = \phi\ (3)\times\phi\ (5)$=2×4=8.

Definition 4 (Fermat's Little Theorem)

If p is prime and a is a positive integer not divisible by p, that is, $gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$. Also, for every integer a we have $a^p \equiv a \pmod{p}$.

Example: If $p = 7$, $a = 2$, $gcd(2, 7)$= 1 $\Longrightarrow 2^6 \equiv 1 \pmod{7}$.

Also, $2^7 \equiv 2 \pmod{7}$

Extended Euclidean algorithm

Algorithm 1 Extended Euclidean algorithm

```
procedure EXTENDED EUCLIDEAN ALGORITHM
    INPUT: two non-negative integers $a$ and $b$ with $a \geq b$
    OUTPUT: $d = gcd(a, b)$ and integers $x$, $y$ satisfying $ax + by = d$.
    If $b = 0$ then set $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, and return(d,x,y).
    Set $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$.
    While $b > 0$ do the following:
        $q \leftarrow \lfloor a/b \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$.
        $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, and $y_1 \leftarrow y$.
    Set $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$, and return(d,x,y).
```

Fact

If A and B are binary numbers and $A \oplus B = C$, then,
$C \oplus B = A$.

Example: If A=1010 , B=0100

$$A \oplus B = \mathbf{1010} \oplus 0100 = 1110 = C$$

$$C \oplus B = 1110 \oplus 0100 = \mathbf{1010}$$

2.2 Encryption

Encryption is a way of transforming regular data into a form that can be understood only by an authorized parties. The opposite of encryption is called decryption, which is transforming the encrypted data to its origin.

There are two basic encryption techniques symmetric and asymmetric. Each of these techniques is used in a variant way based on a need. Each technique has its own characteristics, but the two techniques depend on a science of protecting the data.

First, we present symmetric key encryption in subsection 2.2.1, then we discuss the main challenge for deploying the symmetric key encryption in subsection 2.2.2. After that, we explain the public key encryption in subsection 2.2.3, which is important for the remainder of this thesis.

2.2.1 Symmetric Key Encryption

Symmetric encryption, also known as a conventional, secret-key, or single-key encryption, in which the sender and receiver share the same key for encryption and decryption process. For example, Alice wants to send a message M to Bob. At the beginning, Alice and Bob must have some secret information called *key*, that key (K) is only known to Alice and Bob but not to others. Alice encrypts the message M as $E(M, K)$ to produce the ciphertext C, then Alice sends the ciphertext C to Bob, he in role decrypts the ciphertext as $D(C, K)$ to recover the original Message M. Figure 2.1 illustrates a simplified model of the symmetric encryption process, in which the symmetric encryption has five components [32]:

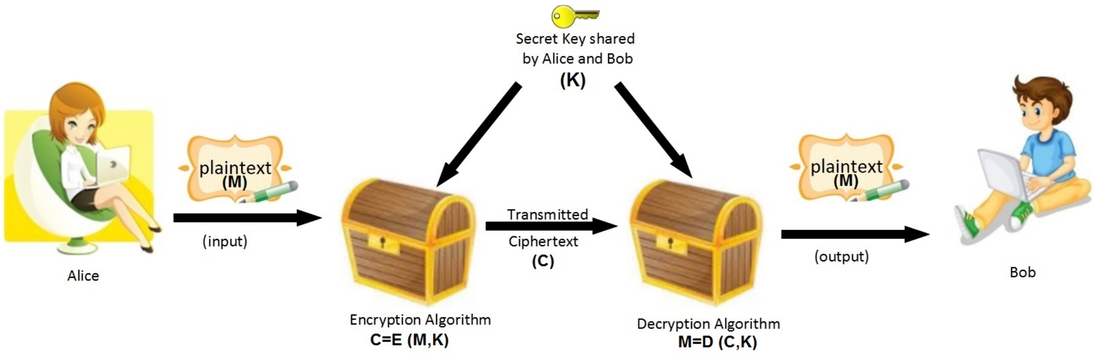


FIGURE 2.1: Simplified Model of the Symmetric Encryption Process

1. Plaintext is the original message or data that is input of the encryption algorithm.
2. Encryption Algorithm is the responsible algorithm for performing various transformations on the plaintext.
3. Secret Key is the secret information that is shared between the sender and the receiver. Also, its an input to the encryption algorithm.
4. Ciphertext is the scrambled data that is produced as an out of the encryption algorithm by using the secret key and the plaintext.
5. Decryption Algorithm is the algorithm that can convert the ciphertext data to it is original plaintext so the authorized user can retrieve it.

The security of symmetric encryption process depends on several factors. First, building strong encryption algorithm known to everyone, where no one can break it to figure out the key or to decipher the ciphertext to find the plaintext [32]. Second, the key must be long enough to avoid the possibility of finding the key through the brute force search. If the key is known to the adversary in anyway, then entire encryption process will be break. Therefore, the key must be kept secret between the two authorized communicated parties (sender and receiver). However, sharing or exchanging the same key between the two communicated parties is not a trivial task basically if they are geographically fare away and they want to communicate via insecure channel. This problem known as *Key Distribution Problem.*

2.2.2 Symmetric Keys Distribution Challenges

Key plays an important role in symmetric encryption, since the algorithm is known, but the key must be secret. To enable the two parties to exchange the encrypted data, they must share the same key, and the key must be protected and no one can access the key except the authorized parties. In other words, we need to ensure the key exchange in away, where no one except the sender and the receiver can perform the encryption and decryption process. But the question is, how the two parties share the key? The answer is that they may exchange the key *physically.* That is, Alice chooses the Key and physically delivered to Bob or visa versa. But if there is a fare distance between Alice and Bob, exchange the keys physically is not a good solution, since how many minutes, hours or days one needs for exchanging the key physically? Also, if the two parties want to change the key more than once, one of them should travel to other to deliver the new key. However, if there are more than two parties like three, four, or more, where each one of them in different countries. In this case, traveling to different countries each time they want to exchange the key is not an easy task. Thus, they may look for another approach to exchange the key. At the first time, they exchange the key physically, and then they use the recent key for encrypting the newly one. But if the attacker in somehow succeeds in gaining access to one key, then, all of the subsequent keys will be compromised, and the ciphertext will be disclosed. The next subsection presents another approach, which is the common approach to solve the key distribution problem that based on the public key encryption.

2.2.3 Public Key Encryption

Public key encryption was introduced in 1976 by Whitfield Diffie and Martin Hellman to solve the key distribution problem [33, 34]. Public key encryption, also called asymmetric encryption, where one party has a secret key called private key and the other party has a public key. The private key must be kept secret, while the public is published, so no need for the sender and receiver to share secret keys, all communications involve only public keys, and no private key is ever transmitted or shared. When Alice wants to send a secret message M to Bob, Alice uses the Bob public key for encrypting the message M as $E(M,\ \mathrm{KU}_B)$ to generate ciphertext C, while Bob uses his private key for decrypting the ciphertext C as $D(C,\ \mathrm{KR}_B)$ to recover the original Message M. Therefore, there is no secret key exchanged between Alice and Bob. Bob private key is still kept secret, while his public key is published. Figure 2.2 illustrates a simplified model of the asymmetric encryption process. In which the asymmetric encryption has six components [32]: Plaintext, Encryption Algorithm, Public key, private key, Ciphertext, and Decryption Algorithm.

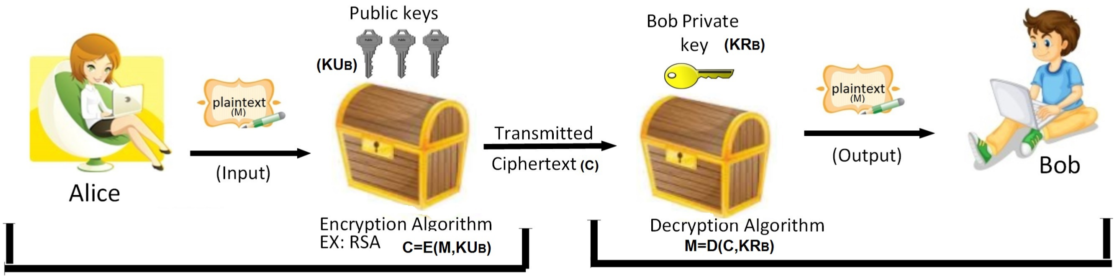


Figure 2.2: Simplified Model of the Asymmetric Encryption Process

Actually, there are many public key algorithms. such as, Diffie–Hellman[35], RSA[1], Rabin [36] , ElGamal [37], Elliptic Curve Cryptography (ECC)[38, 39], and others.

Diffie-Hellman algorithm is used for secret key exchange between two parties securely, then they use the exchanged key for encrypting the subsequent messages using symmetric key encryption [40]. The security of Diffie-Hellman depends on the difficulty of computing discrete logarithms [33]. However, the algorithm itself suffers from the *man-in-the-middle attack*, since it does not authenticate the communicating parties [41].

ElGamal public-key system depends on discrete logarithms of a large prime modulus [37], it's closely related to the Diffie-Hellman technique. ElGamal is not deterministic algorithm [42], encrypting the same plaintext gives a different ciphertext, but unfortunately the algorithm has a disadvantage related to the message size, such as the size of the ciphertext is twice the size of the original message [37].

RSA is most widely accepted as trusted public key encryption, its security depends on the idea of the hardness of factoring large integers [43, 44]. RSA uses two keys, one is published to public and the other remaining secret. The private key is kept secret and therefore need to never be distributed. However, the main disadvantage of current RSA encryption schemes is the computational overhead.

Rabin Encryption algorithm is a public key algorithm, whose security similar to the RSA, based on the hardness of integer factorization. Michael Rabin in [36] prove that Rabin is more secure than RSA, since Rabin is hard as hard of integer factorization, which is not true for the RSA.

ECC is a public-key technology that extends equal protection to the RSA using smaller key size [45]. Since the algorithm is new the confidence level to use it is not yet as high as that in RSA. The critical disadvantages that ECC suffer from that it increases the size of the encrypted message significantly compared to the RSA encryption [46].

In this thesis we concentrate on the RSA and Rabin algorithms, since they are similar in some process like encryption process. Also the two algorithms depends on the integer factorization. However, Rabin algorithm can be replaced by another approach from other public key cryptostsyems.

2.3 RSA and Rabin Algorithms

This section presents two public key cryptosytems: RSA and Rabin cryptosystems. We give some explanation on how the two cryptosystems work and what is the difference between them. Also, what is the advantages and the disadvantages for each one.

2.3.1 RSA Algorithm

RSA is the most popular and widely used asymmetric cryptosystem in the world [47]. RSA cryptosystem developed by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman in 1977. RSA letters stands for the initials of surnames of the inventors [48]. Security of RSA depends on idea of the hardness of factoring a large integers [43, 44].

The basic operation of RSA is represented as follows: if Alice wishes to send a secret message M to Bob; Alice must use the Bob public key (KU_B) to encrypt the message, and Bob uses his private key (KR_B) to decrypt the encrypted message C, as the following steps.

Step1: Bob generates the key pair $(KU_B$, $KR_B)$ by using Algorithm 2:

Algorithm 2 Keys Generation Process

```
procedure
    Choose two distinct large prime numbers $p$, $q$.
    Compute $N$ as $N = pq$.
    Compute Euler's totient function $\phi(N)$ as $\phi(N) = (p-1)(q-1)$.
    Choose a random integer $e$ (public exponent), such that
    $1 < e < \phi(N)$ and $gcd(e, \phi(N)) = 1$.
    Calculate the private exponent $d$ such that $ed = 1 \mod \phi(N)$.
```

Step2: Bob keeps the private key KR_B=(N, d) secret, while publishes the public key KU_B=(N, e).

Step3: Encryption process is done by the sender. In this scenario, the sender is Alice, such as Algorithm 3:

Algorithm 3 Encryption Process

```
procedure
    Alice writes the message $M$ she wishes to send to Bob,
     where $M$ between 0 and $N - 1$.
    She uses a Bob public key ($KU_B$) to encrypt the message $M$ by computes $C$,
     such as $C = M^e \bmod N$.
    She sends the ciphertext $C$ to Bob.
```

Step4: Decryption process is done by the receiver. In this scenario, the receiver is Bob, as Algorithm 4:

Algorithm 4 Decryption process

```
procedure
    Bob receives the ciphertext $C$ from Alice.
    He uses his private key ($KR_B$) to decrypt the ciphertext $C$,
     to recover the original message $M$ as $M$=$C^d \bmod N$.
```

As shown in the steps before, the secret key remains with Bob, not given to Alice nor to any one from the public.

Let's illustrate the RSA idea with the following example:

Example: (RSA encryption and decryption process)

Key pair generation: Bob does the following

1. Selects p=11 and q=7
2. Computes N as N=11×7=77
3. Computes $\phi(N)$ as $\phi(N)$ =10×6=60
4. Selects e=37
5. Computes d as d=13
6. The public key is KU= (77,37)
7. The Private key is KR= (77,13)

The Private key (77,13) is kept secret, also, Bob hides p,q and $\phi(N)$ while the public key (77, 37) is publish.

Encryption Process: Alice does the following

1. Selects the message M=15
2. By using Bob public key, she calculates the ciphertext C=15^{37} mod 77=71
3. Sends the ciphertext C to Bob

Decryption Process: Bob does the following

By using his private key, he computes the message as M=71^{13} mod 77=15

2.3.2 Rabin Algorithm

Rabin algorithm has been developed by Michael Rabin in January 1979, and the security of a Rabin algorithm like that of RSA security, is based on the difficulty of factoring large integers. As Srivastava and Mathur state in [49], the main disadvantage of Rabin algorithm is the extra complexity which is required for decryption process to identify the corresponding plaintext M from the four possible roots. That is, there are four possible output roots m_i where i=1, 2, 3, 4 generated from the decryption process. Thus, we need extra time to know which m_i represents the original message M.

The basic operation of Rabin algorithm is represented in the following scenario. If Alice wishes to send a secret message M to Bob, Alice must use the Bob public key (KU_B)

for encrypting the message, and Bob uses his private key (KR_B) for decrypting the encrypted message C, as the following steps.

Step1: Bob generates the keys (KU_B, KR_B) by using Algorithm 5:

Algorithm 5 Keys Generation

```
procedure
    Bob chooses two distinct large prime numbers $p$ and $q$ on the form $4k + 3$
    Computes $N$, such as $N = pq$.
```

Step2: Bob keeps the private key KR_B =(p, q) secret, while publishes the public key KU_B =N.

Step3: Encryption process is done by the sender. In this scenario, the sender is Alice. as Algorithm 6:

Algorithm 6 Encryption Process

```
procedure
    Alice writes the message $M$ she wishes to send to Bob,
     where $M$ in the rang 0 and $N - 1$.
    She uses Bob public key ($KU_B$) to encrypt the message $M$ by computing $C$ as
     $C = M^2 \bmod N$.
    She sends the ciphertext $C$ to Bob.
```

Step4: Decryption process is done by the receiver. In this scenario, the receiver is Bob. as Algorithm 7:

Algorithm 7 Decryption Process

```
procedure
    Bob receives the ciphertext $C$ from Alice.
    He uses his private key ($KR_B$) to decrypt the ciphertext $C$
     and recovers the original message $M$ as:
       1) Compute $R = C^{(p+1)/4} \bmod p$
       2) Compute $S$=$C^{(q+1)/4} \bmod q$
       3) Find $a$, $b$ where $ap + bq$=1
    Use Chinese Remainder Theorem [50] to find four square roots
      (there are four possible original messages).
       1) Find $X$ as $X$= ($apS + bqR$) mod $N$ =$m_1$, where $m_1$ is the first root.
       2) Find –$X$ as –$X$= $N$-$X$= $m_2$, where $m_2$ is the second root.
       3) Find $Y$=($apS$ - $bqR$) mod $N$ = $m_3$, where $m_3$ is the third root.
       4) Find –$Y$ as –$Y$ = $N$-$Y$ = $m_4$, where $m_4$ is the fourth root.
       5) Choose which $m_i$, where $i$=1, 2, 3, 4, is the correct root (plaintext).
```

The problem in the Rabin algorithm is, which one of the four roots represents the correct message?

The answer is to pad the message in a manner that stand out of the four possible messages fits the padding. For example, adding special character or something else at the end of the message before encryption, and after decryption, only one from the four possible messages will contain that special character. Another way, we can replicate the last part of the message and adding them at the end of the message before the encryption; only the correct message contains the replicating part. Let's illustrate the Rabin idea with the following example:

Example: (Rabin encryption and decryption process)

Key pair generation: Bob does the following

1. Selects p=307 and q=311 (307, 311 on the form $4k$+3). The Private key is (307,311)
2. Computes N as N=307×311=95477
3. The public key is (95477)

The Private key (307,311) is kept secret, while the the public key (95477) is publish.

Encryption Process: Alice does the following

1. Selects a message M=231, the binary representation for M=$(11100111)_2$.
2. Pads the message before encryption (to know which one of the four roots represents the correct message).
3. Adds 3 zero's at the end of the M, M'=$(11100111000)_2$=(1848).
4. Calculates the ciphertext C=1848^2mod 95477=73409
5. Sends the ciphertext C to Bob

Decryption Process: Bob does the following

By using his private key, he computes the message M as the following:

- Compute $R = C^{(p+1)/4} \bmod p \Longrightarrow R{=}73409^{(307+1)/4} \bmod 307 = 6$
- Compute $S{=}C^{(q+1)/4} \bmod q \Longrightarrow S{=}73409^{(311+1)/4} \bmod 311{=}\ 18$
- Find Extended Euclidean algorithm
  $\Rrightarrow$ (-78)(307)+(77)(311)=1 $\Longrightarrow$ a=-78 and b=77

Cont. (Rabin encryption and decryption process)

Use Chinese Remainder Theorem to find four square roots (there are four possible original message)

- Find X as X=(apS + bqR) mod N
  $\Rightarrow$(-78 × 307 ×18 + 77×311×6)mod 95477 $\Longrightarrow m_1$=94562
- Find $-X$ as $-X$=N-X = 95477-94562=915 $\Longrightarrow m_2$ = 915
- Find Y=(apS-bqR) mod N
  $\Rightarrow$ (-78×307×18 − 77×311×6) mod 95477 $\Longrightarrow m_3$ =93629
- Find $-Y$ as $-Y$=N-Y = 95477-93629 = 1848 $\Longrightarrow m_4$=1848

convert m_1, m_2, m_3 and m_4 to binary
$\mathrm{m}_1=(94562)_{10}=(10111000101100010)_2$
$\mathrm{m}_2=(915)_{10}=(111001001)_2$
$\mathrm{m}_3=(93629)_{10}=(10110110110111101)_2$
$\mathrm{m}_4=(1848)_{10}=(11100111000)_2$

select the root (m), that contains 3 zero's at the end of it's binary.
$\mathrm{m}_4=(1848)_{10}=(11100111000)_2$
By removing the 3 zero's from the end of m_4 we get $(11100111)_2$.
convert $(11100111)_2$ to decimal, $(231)_2$.
$\Rightarrow M=\mathrm{m}_4$=231 is the correct message

2.3.3 RSA Versus Rabin Algorithms

The hardness of breaking *Rabin* cryptosystem is equivalent to the hardness of factoring [36, 51–55], while breaking *RSA* not possible to be equivalent to the hardness of factoring [36, 52, 53, 55, 56]. In *Rabin*, the $4k$+3 makes the the factorization problem more complex, since $4k$+3 gives both prime and non-primes numbers, hence the attacker needs extra complexity to check if the number is prime and if the number is on the form of $4k$+3.

Rabin is more efficient than *RSA* in encryption process [55]. The reason is that the encryption process in *Rabin* needs to compute square of the message *modulo* N, while *RSA* needs to compute of *eth* powers. In other words, $C = M^2 \bmod N$ in Rabin, while $C = M^e \bmod N$ in RSA.

Decryption process of *Rabin* cryptosystem produces four results; one of them is correct, while the *RSA* cryptosystem produces one correct result [52, 55]. This point is advantage for *RSA* while disadvantage for the *Rabin* in term of the execution time, but it is an advantage for *Rabin* in term of security since it adds additional complexity for the attacker to distinguish the right root.

2.4 Chapter Summary

We have surveyed the necessary concept related to the public key encryption. We have reviewed a significant mathematical preliminaries will be used in this thesis. Also, we presented the encryption techniques and classifying them based on the number of keys into two groups symmetric key and asymmetric key encryptions. We discussed the common problem in symmetric key encryption, the key distribution problem, and how the asymmetric encryption solves this problem. After that. We have drilled down in public key encryption and introduced the two public key cryptosystems namely *RSA* and *Rabin*. We also, discussed how the two algorithms work and we stated the the basic differences between the two. In the next chapter, we will present a literature review about the RSA attacks, what are the proposed solutions for these attacks and what are the limitations for these solutions.

Chapter 3

Known Attacks on RSA and Mitigation Approaches

In this chapter we survey some of attacks against RSA and the proposed approaches to solve or to mitigate them. We classify these approaches into three categories based on its *techniques*. The first category is the *Dice solution*. Second category called *Dice solution follower*. and third category, *Hungry mouse solution*. After that, we illustrate the limitations for these solutions.

The chapter is organized as follows: Section 3.1 discusses the attacks against RSA. Section 3.2 illustrates the proposed approaches to solve or to mitigate these attacks. Section 3.3 shows the limitations of these approaches. Finally, we conclude the chapter in section 3.4.

3.1 Attacks Against RSA

RSA encryption algorithm is secure as no one gets other than the public key, otherwise the algorithm not secure [57]. For example, $(d, p, q, \phi(N))$ are the important four RSA parameters, they form the RSA trap-door, if any one of them is known, then the RSA will break completely [16].

There are two type of attacks against RSA, *Direct attack* which is called *brute-force*, and the other type is *Indirect attacks*. The attacker in the first category tries to get the secret keys. But, the attackers in the second category try to exploit the weaknesses of the algorithm implementation or depending on other information available to them to generate their attacks.

Unfortunately, the basic version of RSA is a deterministic algorithm. This means, the message has always the same encryption for the same key. This property enable attackers to successfully launch many kinds of Indirect attacks against the algorithm, such as *known plaintext attack*, *chosen-plaintext attack* and others. The thesis will focus on the following Indirect attacks, which are important for the remaining of this thesis.

3.1.1 Common Modulus (CMA)

Sometimes we want to prevent generating different modulus N=pq within the same company, so the chairperson of the company distributes the same N for all or most of the employees. To achieve the security, the chairperson issues different (e, d) pair for each employee, since it is not secure if an encrypted message is sent for one of the employees can be read by others. That is, the public key for *ith* employee is KU_i = (N, e_i) and the private key is KR_i = (N, d_i). For instance, if Alice wishes to send the same message M to Bob and John, she encrypts the message M by using Bob's public key as $E(M,\ KU_B)$ and encrypts the same message by using John's public key as $E(M,\ KU_J)$ where $KU_B = (N,\ e_B)$ and $KU_J = (N,\ e_J)$.

The questions that arise, what is the harm if the same message is encrypted by using the same N with different (e, d)? Is there any useful information can be retrieved by attackers to get the original message? de Vries answered this question in [58]. If the same message M is encrypted twice by using RSA cryptosystem using the same modulus N with different public exponent KU_i= (N, e_i) where i=1, 2, and $gcd(e_1,\ e_2)$ =1, the attacker can recover the message M efficiently from the two ciphertexts C_1 and C_2 by using *Extended Euclidean Algorithm.* That is, suppose that C_B=M^{e_B} mod N, C_J=M^{e_J} mod N be the ciphertexts corresponding to message M sent by Alice to Bob and John, Where $gcd(e_B,\ e_J)$ =1.

In this case the attacker can recover the original message by using Extended Euclidean Algorithm as $M = ({C_B}^a \times {C_J}^b)$mod N , where $e_B \times a+ e_J \times b$=1. In short, the attacker uses a Common Modulus attack to get the plaintext (original message) when the same message is encrypted by two RSA keys that share the same *modulus* N, but different *Public exponent* e. Algorithm 8 summarizes the process of *Common Modulus attack* [59].

Algorithm 8 Process of Common Modulus Attack

```
procedure
    Input:    A modulus $N$, $e_1$, $e_2$, $C_1$ and $C_2$, where gcd ($e_1$, $e_2$) =1.
    Process:  1. By using Extended Euclidean Algorithm, find a and b where
                 $e_1 \times a + e_2 \times b = 1$.
              2. Find the Message $M$ where $M = {C^a}_1 \times {C^b}_2 \bmod N$.
    Output:   An original message $M$.
```

3.1.2 Known Plaintext Attack (KPA)

In this type of attack, the adversary may be able to capture a set of plaintexts with its corresponding ciphertexts [32, 60], to build the set $S = \{(P_1, C_1), (P_2, C_2), (P_3, C_3),..., (P_i, C_i)\}$, where $P_i \in$ plaintext and $C_i \in$ ciphertext.

To understand the concept of the *Known plaintext* attack, suppose Alice sends a message to Bob where Eve is in the middle, for somehow Eve knows that message starts with “Dear Bob” and ends with “Regards”. Thus, Eve collects the plaintext with its corresponding ciphertext to build the above set S. For example, let $S = \{(D, 02)$, $(e, 01)$, $(a, 04)$, ..., $(b, 10)\}$. Eve can used any later captured data to find the plaintext P$_i$ if the corresponding C_i is in the set S. If Alice sends the ciphertext “0201” to Bob; Eve searches the corresponding plaintext for “0201” in the set S to find the message “De”. Algorithm 9 summarizes the process of *Known Plaintext attack*.

Algorithm 9 Process of Known Plaintext Attack

```
procedure
    Input:    $C_i$ (ciphertext).
    Process:  Based on the pre-built set $S$, the adversary compare $C_i$ that captured
              from the sender with its corresponding plaintext in the set $S$.
    Output:   $P_i$ (plaintext).
```

3.1.3 Chosen-Plaintext Attack (CPA)

This attack can be happened, when the adversary chooses arbitrary message, and in somehow she is able to get the source system and insert the chooses message into the system to encrypt it [32]. That is, Eve chooses arbitrary message M, and in somehow she gets the source system and insert this message into the system to encrypt it, the system generates the ciphertext for that message, now Eve has the plaintext with corresponding ciphertext. Eve can build sets of plaintexts-ciphertexts as $S = \{(P_1, C_1), (P_2, C_2),..., (P_i, C_i)\}$, where $P_i \in$ plaintext and $C_i \in$ ciphertext. Eve can use any later captured data from this system to find the plaintext P_i if the corresponding C_i is in set S. Algorithms 10 and 11 summarize the process of *Chosen-Plaintext attack*.

Algorithm 10 Build/Update a Set of Plaintext-Ciphertext

```
procedure
    Input:    Choose plaintext $P_i$.
    Process:  1. Insert the plaintext into the source system (victim).
              2. The system encrypts the message to generate $C_i$ (ciphertext).
              3. Attacker builds or updates the set $S = \{(P_1, C_1), (P_2, C_2),...$
                 $,(P_i, C_i)\}$.
    Output:   set $S = \{(P_1, C_1), (P_2, C_2),..., (P_i, C_i)\}$.
```

Algorithm 11 Process of Chosen-Plaintext Attack

```
procedure
    Input:    $C_i$ (ciphertext).
    Process:  compare $C_i$ that generated from targeted system (victim) with
              its corresponding plaintext in the set $S$.
    Output:   $P_i$ (plaintext).
```

3.1.4 Timing Attack (TA)

As introduced by Wing H. Wong [61], the attacker obtains the information based on the implementation of the algorithm itself, without exploiting any weakness in the mathematical approach that the algorithm applies. The attacker exploits the variance of the time in cryptographic operations. That is, the computations performed by a cryptographic algorithm takes a different amount of time based on the input and the value of the secret parameter in addition to the performance of the system that involved in this computation. If RSA private key operations can be timed accurately, statistical analysis can be used to obtain the secret key involved in the computations.

In RSA algorithm, the attacker needs to know the nature of the targeted system has been used to compute $C^d \bmod N$. The attacker can measure the amount of required time for this computation, after analysis the variance of the time. Due to the Boneh [62], the attacker can recover the private key d one bit at a time until the secret exponent d is known. Also, according to Kocher [18], attackers can recover a private key by computing how long a computer takes time to decrypt a message.

3.1.5 Frequency of Block Attack (FOB)

RSA is a kind of block cipher cryptosystem even it is not intended to be used as a block cipher. RSA is typically used for encrypting small pieces of data, such as symmetric key that is then used to encrypt the entire of the data. Nevertheless, RSA cryptosystem works on block cipher manner where the message is divided into a number of blocks based on the size of the block. The block size can be chosen between 1 to $N-1$ for some N.

We found a new attack that could be carried out against RSA. We call it *Frequency of Blocks Attack (FOB)* attack. This attack exploits the repeated blocks in RSA. Suppose the block size is 127 bytes and the message size is 635 bytes, so the number of blocks are 5. If two blocks are repeated within the same message, most likely each of them will have the same ciphertext. If the attacker in somehow known that the message contains repeated block, may exploit this weakness to find the plaintext. To the best of our knowledge, no one points to this type of attack in RSA. The reason may refer to the fact that RSA is usually used for key exchange and digital signature, instead of encrypting the majority of data.

FOB attack exists in symmetric key encryption, and many techniques used to solve it, such as using cipher block chaining (CBC) mode, cipher feedback (CFB) mode and counter mode (CTR) [63].

3.2 Proposed Approaches for Preventing RSA Attacks

In this thesis, we classifies the approaches for solving or mitigating RSA attacks into three categories based on its techniques. First category is called *Dice approach* [1]; this category uses randomized component R which is added to the message M raised to the power e as RM^e before the mod operation, or raise R with e and multiplying the result with M as R^eM before the mod operation. The second category is called *Dice approach Follower*; it follows the *Dice approach* to solve or to mitigate RSA attacks, except in the *Common Modulus* attack which can be solved by never send exact messages to receivers in certain conditions. Finally, *Hungry mouse approach* [2]; this category uses a random component, such as alphabet component which is appended to the plain message M to generate new message M' then perform the encryption on M'. The main purpose for these proposed approaches is to make RSA cryptosystem semantically secure.

[1] Dice approach: when you throw the dice, each time you get a random number from 1 to 6 [64]. This is similar to the random component which is added to RSA, each time you get a random number between 0 to $n-1$ [12].

[2] Hungry mouse approach refers to the Hungry mouse story, where the mouse has no foods and she grew very thin. The mouse tries to find foods, when she looked here and there, she found a basket full of corn. She found a small hole in the basket, and the hole is suitable for the mouse to creep from it to inside the hole. Since she very hungry, she began ate and ate until she had grown. But unfortunately when the mouse trying to go out of the basket, she could not. She was too fat to pass through the hole [65]. This approach, like the mouse. When we add a component to the message before the encryption may the size of the message with component will be greater than the original range of the RSA size, and in this case we can encrypt the message but we cannot decrypt it.

3.2.1 Dice Approach

Dice approach is proposed to secure RSA against some types of attacks that mentioned in section 3.1. According to Jean Coron et al in [66]; *Dice approach* is a very common solution which is used for encrypting messages by using RSA. Several people follow this solution, such as Malek Kakish in [12] and David Pointcheval in [67]. Algorithm 12 clarifies how the *Dice approach* works.

Algorithm 12 Dice approach

```
procedure ENCRYPTION PROCESS
    Choose two distinct large prime numbers $p$, $q$.
    Compute $N$ as $N = pq$.
    Compute Euler's totient function $\phi(N)$ as $\phi(N) = (p-1)(q-1)$.
    Choose a random integer $e$ (public exponent), such that $1 < e < \phi(N)$ and $gcd(e, \phi(N)) = 1$.
    Select the message $M$ where $0 < M < N - 1$.
    Select the random component $R$ where $0 < R < N - 1$.
    Compute the ciphertext $C$ as $C = (M^e \times R) \bmod N$ .
    Compute the ciphertext for the random component $R'$, as $R' = (R^e) \bmod N$.
```

As it it shown in the *Dice approach* algorithm, the first step is to find two large prime numbers p and q, then calculate N such that, N=pq. After that, based on *Euler Totient function*; compute ϕ(N), where ϕ(N)=$(p-1)(q-1)$. Finally, select a random integer e such that, e is greater than 1 and less than ϕ(N), where gcd(e, ϕ(N))=1. To make the algorithm semantically secure, choose a large random integer R, where R between 0 and N-1 to compute R'.

If Alice wants to send semantically secure message to Bob, she computes the ciphertext C=$M^eR \bmod N$ and then compute R'= $R^e \bmod N$. After that, she sends the ciphertext that contains (R', C) to Bob.

Another form for Algorithm 12 is computing $C = MR^e \bmod N$. In this scenario, the encryption is faster than the first scenario $C = M^eR \bmod N$, since in the first scenario we need to raise each block from the message to *eth* roots and the message may contains several blocks and may each block has a different M^e, while in the second scenario just compute R raised to the power e once and each time multiply R^e for all block need to be encrypted. Or we can use $C = M(R+1)^e \bmod N$, just we compute R for the first time then each time we want to encrypt the message the system itself chooses and incremental random component.

3.2.2 Dice Approach Follower

According to de Vries in [58], to avoid *Common Modulus* attack; sender should avoid to send similar message to more than one receiver. Algorithm 13 illustrates the *Dice Approach Follower* for solving RSA attacks:

Algorithm 13 Dice Approach Follower

```
procedure ENCRYPTION PROCESS
    if The sender uses Common Modules then
        Never send identical messages to more than one receiver

    else
        Apply Dice Approach Algorithm
```

As shown in the *Dice Approach Follower* algorithm, we need to check if the attack is *Common Modules*. If it is, then we ignore sending the message. Otherwise handle the attack by following the *Dice Approach Algorithm*.

3.2.3 Hungry Mouse Approach

Dice approach and Dice approach follower use integer random components, while the hungry mouse approach uses alphabet component. In alphabet there are 52 probability from a to z and from A to Z, while in numbers there are 10 probabilities from 0 to 9, so random alphabet is more secure and complex than random number.

According to Jindal and Gupta in [68], to make RSA secure against indirect attacks, random alphabet R is added to the message M to generate new message M'. Then we encrypt the newly M' instead of M. Algorithm 14 illustrates how to solve RSA indirect attacks by using Hungry mouse approach:

Algorithm 14 Hungry Mouse Approach

```
procedure ENCRYPTION PROCESS
    Choose two distinct large prime numbers $p$, $q$.
    Compute $N$ as $N = pq$.
    Compute Euler's totient function $\phi(N)$ as $\phi(N) = (p-1)(q-1)$.
    Choose a random integer $e$ (public exponent), such that
    1 $< e < \phi(N)$ and $gcd(e, \phi(N)) = 1$.
    Select the message $M$ where $0 < M < N - 1$.
    Select random alphabet $R$.
    Append $R$ to the $M$ to generate $M'$ as $M' = M + R$ .
    Compute the ciphertext $C$ as $C = (M'^e) \bmod N$
```

Line 8 in Algorithm 14; random alphabet R is added to the message M to generate new message M' [3]. Line 9 encrypts M' to generate ciphertext C. The rest of the algorithm as disused above in this chapter.

3.3 Limitations of the Proposed Approaches for Solving RSA Attacks

The Proposed approaches for handling the RSA attacks that mentions in section 3.2 are good approaches, but actually they have some limitations.

Dice Approach Limitations

Actually the algorithm is semantically secure, since the output of the algorithm depends on the value of randomized component R, but this algorithm suffers from the **FOB** attack, since the algorithm does not solve the problem of block redundancy inside the same message. Attackers can invest this threat for generating their attacks.

Adding randomized component to the RSA and encrypting this component by the same RSA cryptosystem is less secure, because if the attacker can solve the RSA factorization integers, RSA cryptosystem will no be secure and will not benefit from this addition of randomization component. Breaking the RSA factorization mean breaking this randomized component and the entire system will be compromised.

Dice Approach Follower Limitations

The *Dice Approach Follower* has the same limitations of *Dice solution*, since it is follow the *Dice Approach* for solving some of RSA attacks. Although, the way of solving the *Common Modules* attack is good, but it is not sufficient. If the message is emergency, such as reporting bank robbery message, we cannot avoid sending this message to the receivers, even if all of the receivers have same N and different e and d.

Hungry Mouse Approach Limitations

RSA necessitates that M should be between 0 and $N-1$ [69–71], so as this solution if $M' > N-1$, then the algorithm will no work, the following example illustrate limitation:

[3] Another form for *Hungry Mouse Approach Algorithm* by computing M' as $M' = MR$ instead of $M' = M+R$

Example

Let $p = 11$ and $q = 3$

Compute N as N=p×q=33

Compute $\phi(N)$=$(p-1)$ ×(q-1)=$(11-1)$ ×$(3-1)$=20

Select $e = 3$, and compute $d = 7$

public key=$(N = 33, e = 3)$

private key=$(N = 33, d = 7)$

Let $M = 30$ and R = "A", where "A" in ASCII=65

Compute $M' = 30 + 65 = 95$

Encryption: $C = 95^3$ mod 33=2

Decryption: $M' = 2^7$ mod 33=29

Find $M = 29 - 65$=-36, but actually M is 30 not -36

Note: R could be word or character.

To make this algorithm works properly M' should be between 0 and $N-1$, where M'=M+R. Thus, the message should be small in order to choose a good randomized component, or the randomized component should be small for large message, and there is no great benefit in these two cases. However, since the redundancy is possible in this approach, so its not secure against *FOB attack*.

3.4 Chapter Summary

We have discussed the important attacks against RSA. Such attacks, *Common Modulus*, *Known plaintext*, *Chosen-plaintext*, and *Timing* attacks. In addition, we surveyed interesting approaches for solving these attacks, such as *Dice approach*, *Dice approach follower*, and *Hungry mouse approach*. However, we found some of limitations for these approaches and presented them in this chapter. Moreover, we have discussed the *Frequency of Block* attack. To the best of our knowledge no one mention this type of attacks against RSA.

In the next chapter we will introduce our proposed solution which is called *Yamen cryptosystem* for counter-measuring the aforementioned attacks against RSA.

Chapter 4

Enhanced RSA Cryptosystem (Yamen Cryptosystem)

This chapter presents a new method for combining two public-key algorithms *RSA* and *Rabin* to produce an enhanced version of RSA public-key cryptosystem namely *Yamen Cryptosystem*. A new additional randomization component is added to RSA algorithm. This component is encrypted by *Rabin* algorithm to improve the security level of RSA against *indirect attacks* which are discussed in Section 3.1 and make RSA semantically secure. We employ *Huffman compress* algorithm in the enhanced RSA to reduce data redundancy before adding the randomized component which make the enhance version more secure against *FOB attack*. Also, *Huffman compress* is used to improve the execution time for the enhanced RSA.

Section 4.1 presents the components used in building *Yamen cryptosystem*. Section 4.2 presents a new design model and implementation of *Yamen cryptosystem*. Section 4.3 discusses how *Yamen system* makes the data as short as possible to save space. Section 4.4 discusses how *Yamen* speeds up the encryption and decryption execution time. Finally, the chapter summary is given in section 4.5.

4.1 Yamen Cryptosystem Components

In this thesis, RSA cryptosystem is combined with Rabin algorithm to make RSA stronger against the attacks presented in Section 3.1. The security of RSA and Rabin cryptosystems depends on the intractability of the integer factorization [72, 73]. Therefore, if the attackers want to break the enhanced RSA, then they need to break the factorization of large numbers for both RSA and Rabin. Consequently, the attack will require more time from the attacker than before.

Combined the aforementioned algorithms enhances the security of enhanced RSA algorithm. A randomized parameter called Y is added to the basic RSA to make it semantically secure. This means that the same message could have different encryption outputs. Therefore, an attacker cannot differentiate between two ciphertexts from each other even if the attacker knows or chooses the corresponding plaintexts. Also, the combined algorithm deploys the compression algorithm (Huffman coding) to reduce the data redundancy and to make all of the characters have the same level of distribution and this solve the *FOB* attack. To enhance the execution time, we encrypt the header file and blinding the binary file results from compressed message instead of encrypt the entire message. The binary file of the compressed message is blinded by the randomization component Y to make it *semantically secure* (The binary file for the same message will be different based on the value of Y).

Yamen cryptosystem has four components: RSA Algorithm, Rabin algorithm, Huffman coding, and Random component. RSA and Rabin presented in section 2.3. Section 4.1.1 discusses the Huffman coding in *Yamen Cryptosystem*. Section 4.1.2 presents the natural of the random component used by *Yamen cryptosystem*, and how it make the algorithm semantically secure.

4.1.1 Huffman Coding in Enhance RSA (Yamen Cryptosystem)

According to the Simmons in [74], cryptographers consider data compression algorithms as a ciphering scheme. In [75], Shannon suggests reducing the redundancy in data before encryption to protect it against statistical analysis. In this thesis, we use Huffman codes to achieve Simmons and Shannon suggestions. However, you may replace Huffman with another suitable and more effective compression algorithm.

Huffman coding is a common method for data compression [76]. The compression process facilitates storing and transmission large data. Huffman coding has been developed by David Huffman in 1952. It is a lossless data compression algorithm [77]. This mean, the original data can be recovered exactly from the compressed data. The algorithm is used to compress data (symbols, alphabet, ...) to generate variable-length codes instead of fixed-length codes for each symbol. Given a set of symbols in a file, the algorithm performs some statistical analysis to construct a table that contains the frequencies of occurrence for each symbol. The algorithm uses the constructed frequency table to build Huffman tree, which is used to assign each symbol with it is appropriate code length based on the symbol occurrence.

The result of applying Huffman coding on the data file is two files *binary file* (B) and *header file* (H), where the binary file depends on the header file to retrieving the original data. If we lose the header file, then we cannot retrieve the true data. Actually, *Yamen* system depends on this fact for enhancing the speed in encryption and decryption process. The following example describes Huffman coding process.

Let us take the message: `Hello, this message is created by Alice.`

There are three steps for generating the header file and binary file for that message.

Step 1: Create a frequency table based on statistical analysis. column 2 in Table 4.1 illustrates the frequency for each character in the message.

Symbol	Number of Ocurence	Code	Code Length	Total Length
sp	6	101	3	18
e	6	110	3	18
s	4	000	3	12
c	2	0101	4	8
l	3	1000	4	12
i	3	1110	4	12
t	2	0110	4	8
d	1	00111	5	5
A	1	00110	5	5
a	2	11110	5	10
o	1	00100	5	5
m	1	10010	5	5
H	1	01000	5	5
,	1	01110	5	5
r	1	00101	5	5
y	1	01001	5	5
À	1	01111	5	5
g	1	111110	6	6
b	1	100111	6	6
.	1	111111	6	6
h	1	100110	6	6
Total	41			167

Table 4.1: The Symbol's Frequencies in Details

Step 2: Based on the stored frequency table, construct a code tree. Create a parent node with a frequency that is represent the sum of the two smaller symbols frequencies as Figure 4.1.

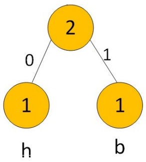


Figure 4.1: First Two Smallest Symbols into Leaves

Repeat the process in a loop by combining the two smallest symbols to reach the tree in Figure 4.2

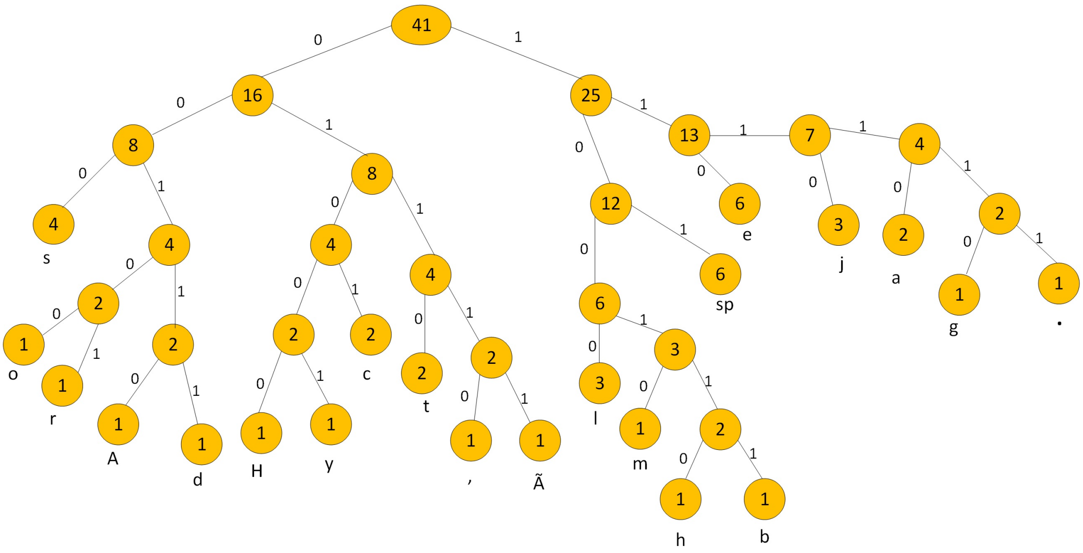


Figure 4.2: Code Tree According to Huffman

To form a Huffman code, traverse the tree to symbols you want. For instance, the output is 0 each time the left branch is taken, and the output is 1 each time when the right branch is taken. Column 3 in Table 4.1 depicts the Huffman code, while column 4 is the length of the Huffman code in bits. Column 5 represents the total length in bits which is computed by the formula $\textit{Total Length} = \textit{Symbol Frequency} \times \textit{Code Length}$.

Step 3: Construct the header file and the binary file for the message. The header file H is the file that contains all symbols or the ASCII for each symbol from the origin data file, where each symbol inside is assigned with it is occurrence. The Header file contains unique symbols, where no symbol is repeated twice. The binary file B is the file, where each symbol inside it is replaced with its code. For instance, consider a message before compression is: `Hello, this message is created by Alice.`

The number of bits in this message = 41 symbols × 8 bits/symbol = 328 bits, since each symbol represents in 8 bits (fixed length). We take 41 bits instead of 40 bits, since we add the extra bit for EOF character, which is represented in Table 4.1 by the symbol À.

Header File:
For simplifying, we write the header file on the form (character and its binary code [4]), where the binary code for each character in the message exist inside the Table 4.1
`sp101e110s000c0101l1000i1110t0110d00111A00110a11110o00100m10010H01000,0111`
`0r00101y01001À01111g111110b100111.111111h100110`

Binary File:
The corresponding message "`Hello, this message is created by Alice.`" is presented in the binary file as follows:
`01000110100010000010001110101011010011011100001011001011000000011110111110`
`11010111100001010101001011101111001101100011110110011101001101001101000111`
`00101110111111011110`

The number of bits for the compressed message in the binary file equal to 167 bits. Each symbol represents in the binary file is assigned to certain length of binary code based on its frequency of occurrence. In our message example, we represent the character e by 110, the character a with 11110, and so on. From the example, e has a different length of binary code comparing with a, since e repeated in the message 6 times, while a repeated in the message 2 times. The process in which the different length of binary code assigned to each symbol based on the number of occurrence of that symbol called (variable length).

Figure 4.3, summarizes the Huffman coding Process by compressing data to generate the header file and binary file.

[4] In our application *Yamen Creyptosystem*, we replace the symbols by their equivalent of ASCII code in the Header File. We replace d with *100* in decimal and *space* with *32* and so on. Also, we replace the binary code with frequency of characters occurrence. This changes depend on the programmers, no matter how you want to store the header file (character: binary code) or (ASCII: frequency of occurrence).

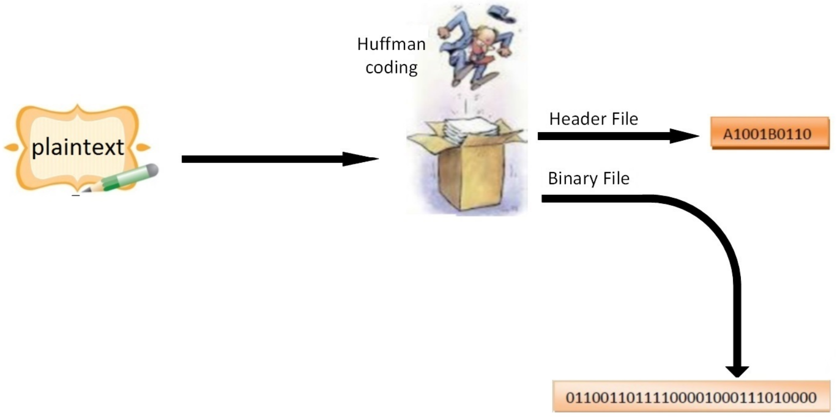


Figure 4.3: Summarize the Huffman coding Process

Figure 4.3 depicts, the plaintext is passed to Huffman code, which is compressed the plaintext to generate header file and binary file.

To un-compress the message, first read the header file, and use its information to build the Huffman tree as the tree in the Figure 4.2. Second, read the binary file bit by bit, then begin from the root of the tree, when finding 0 bit, move to the left on the tree, when finding 1 bit, move to the right on the tree until finding a leaf node (we have found the symbol). Then repeat the process for all remaining bits until all message characters are retrieved.

4.1.2 Random Component in Yamen Cryptosystem

RSA is deterministic algorithm. Encrypting the same message more than one time by using RSA cryptosystem with the same key leads to the same ciphertext. Accordingly, an attacker may generate different types of attacks to recover the original message without knowing the keys. To prevent this type of attack, each time the encryption of the same message with the same RSA key should lead to different ciphertext. This is the idea behind of using random component in *Yamen cryptosystem*

For making the encryption semantically secure, we use randomized component in the encryption algorithm. We use the letter Y as symbol for the randomized component, where Y is a random integer.

Characteristics of Y:

- Random integer used once for each message (nonce).

- We can use any random number generators to generate the random number. In *Yamen* system we used congruential generator[5] [78]. We select the seed[6] and pass the seed to the congruential generator function and the function itself generate the number as the following:

  $r_{n+1} = a \times r_n + c \pmod{m}$. Where:

  r_0 is a seed.

  r_1, r_2, r_3, ..., are the random numbers.

  a, c and m are constants, In Yamen system we select $a = 159$, $c = 806$ and $m = 2^{32}$

- We use Y for blinding[7] the ciphertext of the header file and blinding the binary file. In the case of Y less than the ciphertext we repeat Y many times to be as the length of the ciphertext and if the Y greater than the ciphertext, we remove the number of bits from Y to be as the same length of the ciphertext. And the same idea applying when blinding the binary file.

Yamen cryptosystem uses the random number to make the algorithm semantically secure, each time the message is encrypted, different ciphertext is obtained. Therefore, it is hard for an attacker to learn from the ciphertext about the original message, because ciphertext for the same message looks different. The following scenario explains how *Yamen* system make the algorithm semantic secure. Suppose Alice wishes to send a secure message M to Bob, and Alice wants the encryption of the message to be semantically secure, she does the following:

1. Alice encrypts the message by using Bob's public key to generate the ciphertext $C = E(M, KU_B)$.
2. Alice chooses a random integer, such as Y.
3. Alice computes $C' = C \oplus Y$
4. Alice sends C' to Bob.
5. Bob computes $C = C' \oplus Y$, to get the original ciphertext.
6. Bob uses his private key to decrypt $D(C, KR_B)$.

[5] congruential generator is an example of random generator, but any trusted generator rather than this generator can be used

[6] Seed is any integer used to initialize a random number generator.

[7] Blinding is a technique used by *Yamen cryptosystem* to scramble the message by integer number, to preventing attacks from knowing the actual ciphertext for header file and the actual binary file.

The ciphertext depends on the value of Y, suppose we encrypt the message twice, such as the ciphertext for the first time is C_1 and the ciphertext in the second time C_2, since we encrypt the same message this means $C_1 = C_2$. Let $C'_1 = Y_1 \oplus C_1$ and $C'_2 = Y_2 \oplus C_2$, if $Y_1 \neq Y_2$ then $C'_1 \neq C'_2$. This is what we call it *Semantically Secure Based on Randomized Component.*

In this thesis, we will exploit the concept of semantically secure as:

1. Compress the message by using Huffman codes.
2. The results from the compression are header file and binary file.
3. Encrypting the header file by using *RSA Cryptosystem* and the result is a ciphertext C, to make it semantically secure we choose a random component called Y and then calculate new ciphertext C' as $C' = C \oplus Y$. In this thesis, we call C' *Mixture.*
4. To make the binary file B semantically secure as the header file H, we blind B to gets B' using the same Y that has been used in calculating C'. To generate B' we do the formula $B' = B \oplus Y$ to make B different.
5. The Random component Y, should be secret, so we use Rabin encryption algorithm, to encrypt Y.

Figure 4.4, illustrates the process of adding a randomized component to the Huffman header file in addition to blinding the Huffman binary file.

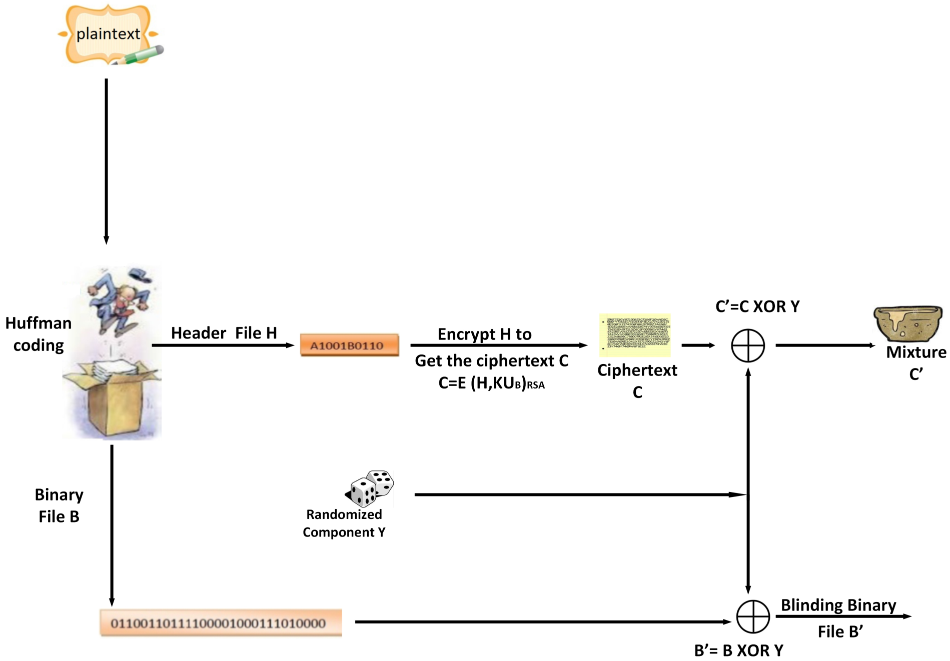


FIGURE 4.4: Adding Randomized Component to the Huffman Header File and blinding Binary File

In Figure 4.4, the plaintext message is entered to the Huffman coding compression algorithm. Huffman coding compresses the message to generate header file H and binary file B. The header file passes to the RSA algorithm for encryption. The resulting ciphertext C, that is generated from encryption process is XOR-ed with the random component Y to generate the mixture C'. Also, the binary file B is XOR-ed with the same Y to generate blinding binary file B'.

4.2 Design Model and Implementation of Yamen Cryptosystem

Section 4.2.1 presents new models for *Yamen cryptosystem.* One of them represents the encryption process and second one represents the decryption process. In the next section 4.2.2 we present the way to implement *Yamen* system encryption and decryption process.

4.2.1 New Design Model for Yamen Cryptosystem

Figure 4.5 depicts the modified version of the encryption process for the basic RSA Cryptosystem. If Alice wants to send a message (M) to Bob, she chooses a randomized parameter (Y), then she compresses the message by using the Huffman code to generate the *header file* (H) and *binary file* (B). She computes the ciphertext (C) for the *header file*, such as C=$E(H,\ K\mathrm{U}_B\ )_{RSA}$. To make the encryption semantically secure she computes the *Mixture* (C'), such as C'=$C\oplus Y$, also she blinds the binary file B with Y to generate B' as B'=$B\oplus\ Y$. Alice encrypts Y by *Rabin Cryptosystem* as $E(Y,\ K\mathrm{U}_B\ )_{Rabin}$, to increase the complexity against the adversary. Finally, Alice sends packet contains ciphertext of Y, C', and B' to Bob.

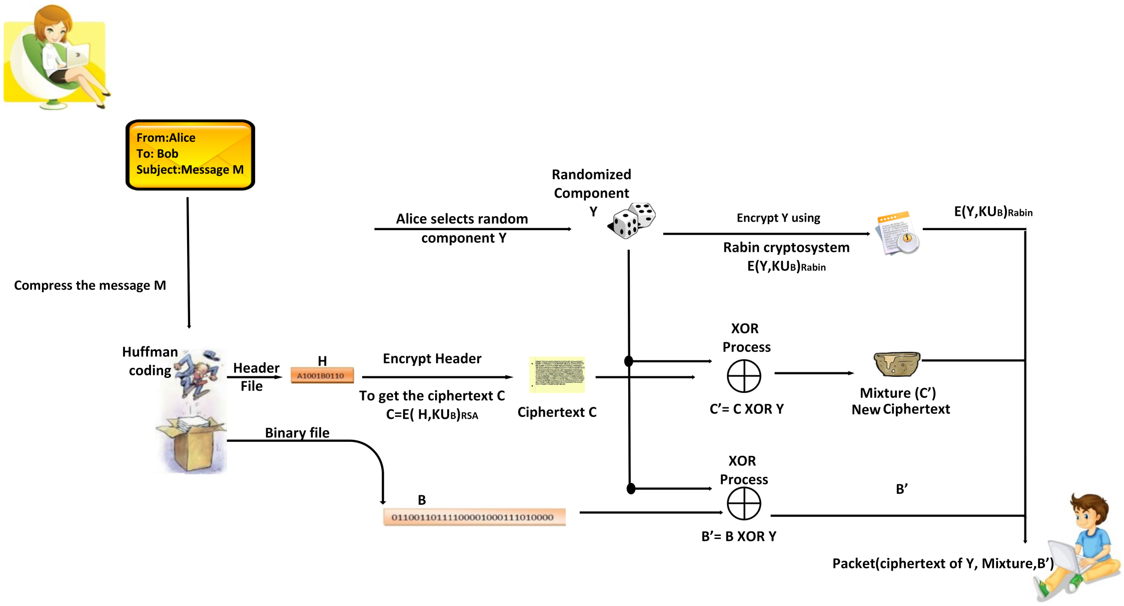


Figure 4.5: Modified RSA cryptosystem (Model for Encryption Process)

Figure 4.6 depicts the modified version of the decryption process for the basic RSA, Bob uses his Rabin private key to decrypt the randomized component Y as Y= $D(Y,\ K\mathrm{R}_B)_{Rabin}$, then he removes Y from the *Mixture* (C') to get the encrypted header C as C =$C'\oplus Y$. Bob gets the header file H by using his RSA private key to decrypt C as H=$D(C,\ K\mathrm{R}_B)_{RSA}$, after that Bob recovers B from B' as B=$B'\oplus Y$, then he uses H to recover the original message M from the binary file B (uncompressed the binary file).

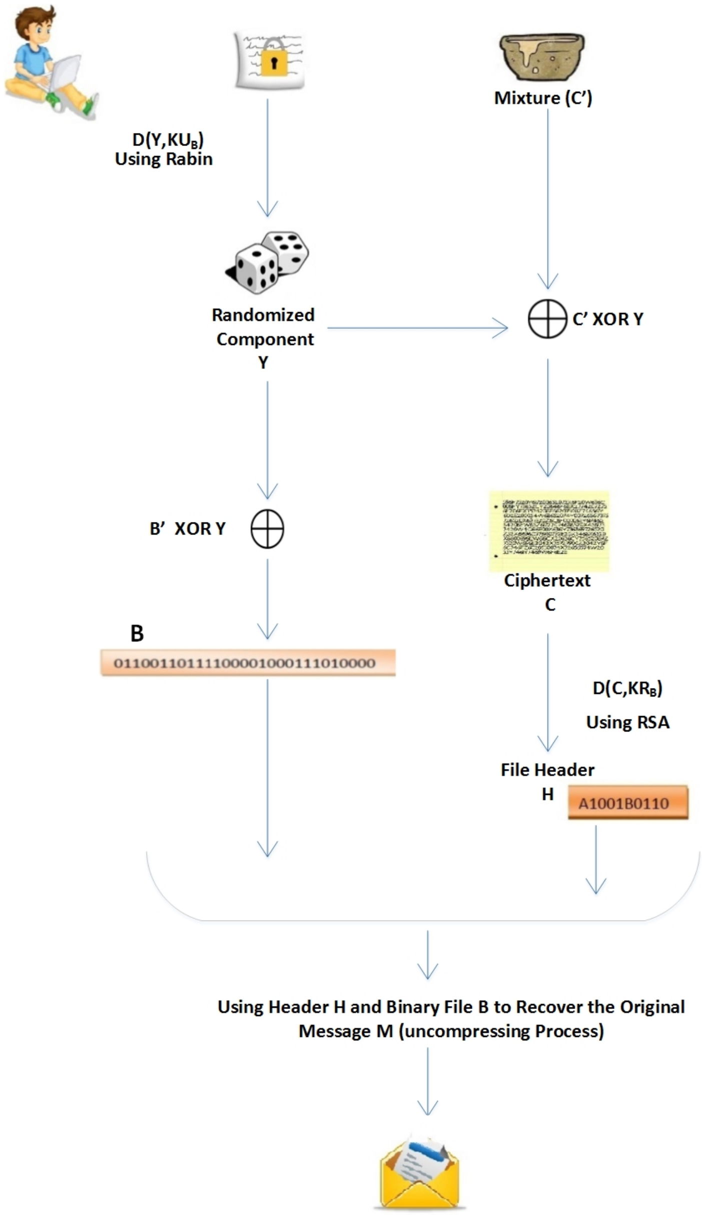


Figure 4.6: Modified RSA cryptosystem (Model for Decryption Process)

4.2.2 Implementation of Yamen Cryptosystem

Yamen cryptosystem is a desktop application which has been implemented using Java programming language, using *Eclipse JUNO version 4.2.2*. *Yamen Cryptosystem* contains two view each of these view contains seven phases. The *Educational View* that can be used by instructors to teach and demonstrate the way that *Yamen cryptosystem* works, where every executed step is visible. Due to the *Yamen* system, students can analyze each step as long as they want, then perform their own tests. *Business View* could be used by the companies to encrypt their data. All steps are executed by the system in the background, no need to think how the system works, just run the system by feeding it the needed files and then determine the encrypt or decrypt process. However, the two views contain generating keys and random component phase, compression phase, encryption phase and blinding phase in the case of encryption process, or un-blinding phase, decryption phase and final uncompressing phase in the case of decryption process. The Figure 4.7 shows a snapshot for *Yamen cryptosystem* user interface.

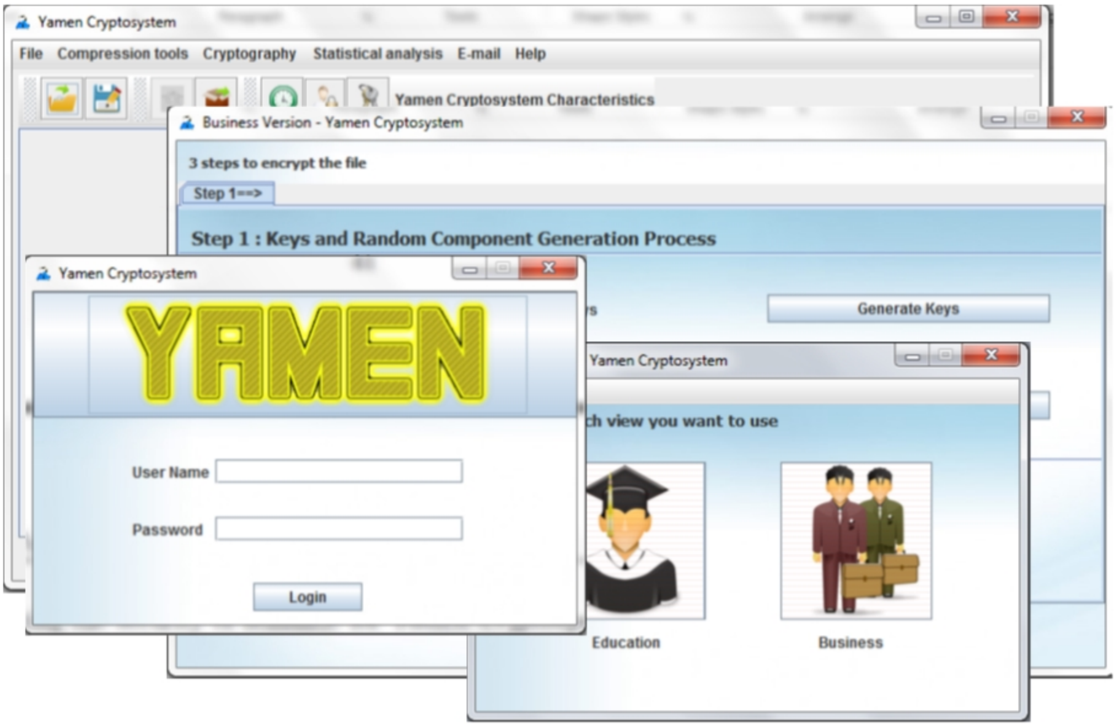


FIGURE 4.7: Snapshot for Yamen Cryptosystem

Because there are different programing languages around the world. The best way to make the programmers understand your code is to use an algorithm. For this reason, we do not focus on any programing languages to disucss *Yamen Cryptosystem*.

The following algorithms, present the encryption and decryption processes in *Yamen cryptosystem*:

Step 1: Algorithm 15, presents keys generation process on the receiver side.

Algorithm 15 Generate Yamen cryptoystem Keys

```
procedure GENERATE KEYS
    Compute RSA public key $(e, N)$
    Compute RSA private key $(d, N)$
    Compute Rabin public key $(N)$
    Compute Rabin private key $(p, q)$
```

Keep the private keys secret and publish the public keys.

Step 2: Algorithm 16, presents random component generation process on the sender side.

Algorithm 16 Generate Random Component

```
procedure GENERATE $Y$
    Select a seed, where the seed is any integer
    Pass the seed to the congruential generator function
    Generate $Y$ from congruential generator function
```

Step 3: Algorithm 17, presents message compression process using Huffman code on the sender side.

Algorithm 17 Compress Message

```
procedure COMPRESS THE MESSAGE M
    Pass the message to the Huffman Code
    Generate Binary file $(B)$ from Huffman Code
    Generate Header file $(H)$ Huffman Code
```

Step 4: Algorithm 18, presents message encryption process on the sender side.

Algorithm 18 Encrypt the message

```
procedure ENCRYPT THE MESSAGE M
    Pass the header file $(H)$ to RSA cryptosystem
    Encrypt $H$ by RSA and the result is $C$
    Blind $C$ by $Y$ to generate the mixture $C'$ as $C'=C \oplus Y$
    Blind binary file $B$ by $Y$ to generate $B'$ as $B'=B \oplus Y$
    Encrypt $Y$ by Rabin Cryptosystem
```

Send C', B' and encryprted Y to the receiver side. To decrypt the message, the receiver does the following:

Step 1: Algorithm 19, presents message decryption process on the receiver side.

Algorithm 19 Decrypt The Message

```
procedure Decrypt Message M
    Decrypt Y using Rabin cryptosystem
    Compute C from the mixture C' as $C=C' \oplus Y$
    Compute B from B' as $B=B' \oplus Y$
    Pass C to RSA decryption to generate H
```

Step 2: Algorithm 20, presents message uncompress process on the receiver side.

Algorithm 20 Uncompress The Message

```
procedure Uncompress The Message M
    Pass the H and B to Huffman code
    Used H to build Huffman tree
    Decode B to recover M using the Huffman tree
    The output is the message M
```

4.3 Reduced Space Results

Yamen cryptosystem, used Huffman coding compression algorithm. This algorithm compress the message to reduce its size. *Yamen* system exploits the Huffman coding as:

1. Reduce the redundancy in data. That is, *Yamen cryptosystem* used *Huffman* to represents each character with its occurrence (Number of times the character is repeated) instead of storing the same character many times. This make the message as short as possible.

2. The output of the compression are header file H and binary file B. The header file is not useful without the binary file and vice versa [79]. Also, in most cases the binary file is larger than the header file. *Yamen* depends on those two facts in its encryption. In most cases, the result of applying RSA encryption technique is a ciphertext which is greater than the plaintext itself because RSA uses a padding scheme[8]. To save space as possible, *Yamen* just encrypts the header file instead

[8] Padding scheme is the number of bits/bytes added to the message to define the block boundary.

of encrypting the binary file or the entire true message. In addition, *Yamen* uses blinding technique by applying the XOR operation on the ciphertext of the header file and the binary file. The size of the XOR output is equal to the size of the largest input operand. In other words, the size of output of XOR-ed two numbers each of them is 8 bit is 8 bits. And, if one of the numbers is 9 bits and the other is 15 bits, then the output is 15 bits. No extra bit greater than the larger operand numbers. We can solve the hungry mouse approach using XOR operation. Instead of using addition or multiplication, we can use the XOR operation to make the additive random component not affect on the message size. Which is mean, the message with a random component always less than the modules N

Yamen system is not only used for securing the message, but also, it is used to make the ciphertext as short as possible. This saves space and make transmission the data over the network faster. In addition, less space is used to store the data in an effective way.

4.4 Speeding Up Yamen Cryptosystem

RSA operations are power and modulo arithmetic operations of large numbers operation, so these operations reduce the speed of RSA [80] and make RSA slow in restricted environments [81]. Therefore, RSA usage is limited for encrypting session keys and digital signatures instead of encrypts the large data [82]. Large data is encrypted by using symmetric key encryption. Hence, session keys are encrypted using RSA and communicated parties use these keys to encrypt the large data using one of symmetric key encryption algorithms.

Yamen cryptosystem uses two things to speedup RSA, *Huffman coding* and *XOR operation.Yamen* cryptosystem uses Huffman coding to compress the data, so the result of this compression are header file and binary file. *Yamen* encrypts just the header file instead of encrypting the entire message. Also, it blinds the binary file using an XOR operation, since XOR has been always faster than multiplication, division, subtraction and addition [25]. So, *Yamen cryptosystem* is more faster than RSA. In *Yamen cryptosystem* no need for using symmetric key algorithm, since the large data reduce to small one, thus *Yamen* system itself could encrypt/decrypt the large data in less time, so *Yamen cryptosystem* is more effective than basic RSA.

4.5 Chapter Summary

We have presented the methods and components used in *Yamen cryptosystem. Yamen* uses randomized component called Y to make it semantically secure. Also, it depends on *Rabin* cryptosystem for encrypting this component. *Yamen cryptosystem* uses Huffman code to remove the redundant symbols from the plaintext messages before the encryption. This save space and make the message more secure against *FOB* attack. Also, *Yamen cryptosystem* uses Huffman and XOR operation to speedup the algorithm computation and keep it superior the basic RSA.

In the next chapter, we present our testing results of *Yamen cryptosystem* comparing to basic *RSA*.

Chapter 5

Performance of Yamen Cryptosystem

Yamen cryptosystem comes with three sensitive enhancement factors comparing with *basic RSA*. These factors are *Security*, *Execution time* and *ciphertext size.* Section 5.1 presents the overview of experiment setup, then main finding results are presented in section 5.2. Also, the main differences between *Yamen* cryptosystem and the basic *RSA* are summarized in section 5.3. Chapter summary is given in section 5.4.

5.1 Overview of Experiment Setup

In this section we summarized the overview of experiment setup. For testing the security of semantic for *Yamen cryptosystem*, we generated a test file and two different random components Y_1 and Y_2. We applied the encryption process on the same file with these components and each time we got different encryption. Proofs and evidence of the claim that *Yamen cryptosystem* is *semantically secure* presented in the section 5.2.1.

For testing the execution time, we need a random files and random component, so we generate a random test files with different sizes, then generated keys with 1024 bits for RSA and Rabin. Also, we generate a random component for *Yamen cryptosystem*, then we run the application on Windows 7, Model: Latitude E5420, Processor: Core (TM) i7-2640M, CPU 2.80 GHz, Memory 8 GB and System type is 32-bit operating system. We observed that *Yamen* system is faster than basic RSA in the case of decryption and encryption. Also, we executed the same experiment on two different PC's with different characteristics and we got the same results.

To reduce ciphertext size, *Yamen cryptosystem* uses *Huffman* compression algorithm, to compress the file before the encryption process. We found the produced ciphertext from *RSA* is greater than the original plaintext, while the produced ciphertext from *Yamen* system is less than the orginal plaintext.

5.2 Results Analysis

This section presents the finding results with respect to security, execution time, and the size of the ciphertext.

5.2.1 Security Issue

Yamen cryptosystem is semantically secure and mitigates the RSA attacks presented in section 3.1.

1. Yamen System is Semantically Secure

Choosing a message M and two different random components Y_1 and Y_2, leads to different encryption for the same message. Let the original message is
"`Hello, this message is created by Alice.`"

The ciphertext of header file in decimal values (Without using any random components):

```
2693501899457580877460174812454487875225019780748904308842659828080766478
7108011549972799414729590362475610248778830538113536194202183289739885607
4023400474060461443195028474832945729657280154836425983745483707887208177
5832900869569483855704632151493693790436559477411340065007460335770828453
4213636220749520277877213268497269896155252383181767231822071472860623852
0306459213430622178770239758565117998465711496426570005631447067558488639
4142327315716563952470724068298553015671841026245837581283520835875982176
3808951354876446748193791455993743308591376928305582307130108337417685742
7831838938822331805507081410261 7
```

Binary File before encryption (Without using any random components).

```
0100011010001000000100011101010110100110111000010110010110000000111101111
0110101111000010101010010111011110011011000111101100111010011010011010001
11001011101111111011110
```

Let the random number $Y_1 = 965$, the binary representation of Y_1 is 1111000101, which is generated by congruential random generator with $Seed = 1$.

The corresponding ciphertext of the header file:

```
7734603758932941297561426833292196189106980827565416859381798210543977923
5243168726398192097154081364801786432403546561237891815822318668170278984
9390088457830633354914604559785226527992633390115038346893391081804277692
5375041720803533464001127765064630314735537555115588310973666674185109147
7170671015871700853958204778955772384997651883360185517641966838459525403
1424663681073282483583416897003840101053423920000082512775228717613428545
6089712091163832504924491356184847019270277133580535041683550116584353528
0612265704177173015880139439005719033405680804886159322356072065553944602
44138101147668978242314169137085
```

The blinded Binary file which is the result of XOR-ing the Binary file with random number Y_1

```
1011011111110100011111001011110010001000001100111011011101011110111110000
1110000000100000010100011100100110110100100101010010110001100010110101101
1001010010101000101111
```

Let the random number $Y_2 = 1124$, the binary representation of Y_2 is 10001100100, which is generated by congruential random generator with $Seed = 2$.

The corresponding ciphertext of the header file:

```
9782207142968026859824128908670379720101147739778918285036680122868886140
6892791721142728846016878407545699924014653970658311844999587170366633124
1783601811185460844339618416225103366924374904940818636146218692896055297
0009018862915881952234918955650788122336734298758609261755241478956752113
2390010710082723266743773046299960377977892208551444224720432873830983909
0908568690547183972591552405724934255184371092948432510560440238951507282
5703931854627486456847374004542735678052660320223376274164604004700322544
3122779217738948137963940548693457061929957706770692286981808750973680211
31937725122215257143211876668 53
```

The blinded binary file for Y_2:

```
11001010000110011011000110011001000010111000101000000001000011000110011001
```

```
0010001100001011101100000101010010111111011110100100001000001011111110101
1010111100110011111010
```

As we can see, encrypting the same message gives different ciphertext. Therefore, *Yamen cryptosystem* is semantically secure. The attacker cannot know if this ciphertext related to one message or to the different messages. While is not the case for the basic RSA where encrypting the message with same key always gives the same ciphertext.

Mathematical Proof to show that Yamen cryptosystem *is semantically secure:*
Suppose the ciphertext for the message M is C.
Select two different random components Y_1 and Y_2.
Compute C_1'=$C \oplus Y_1$, and C_2'=$C \oplus Y_2$. Because $Y_1 \neq Y_2$ and the C is the same for the two Y's. This means $C_1' \neq C_2'$. This is what we want to proof.

2. Yamen Cryptosystem Mitigate and Thwart Basic RSA Attacks Using Evidence and Proofs

In this section we describe how *Yamen cryptosystem* resists or mitigate attacks describes in section 3.1 through evidence and proofs.

1. Common Modulus Attack

   The basic RSA is not secure against this type of attack while *Yamen cryptosystem* mitigate it. The *Common Modulus Attack* can be used to recover the message that was encrypted using two RSA keys by using the same modules N with different public exponent e.

   Theorem 1: In [16, 83], Let $N = pq$ be a RSA modulus and let $< e_1, N >$ and $< e_2, N >$ be two public keys such that $gcd(e_1, e_2) = 1$. Suppose a plaintext M is encrypted with both public keys. If we know C_1=M^{e_1} mod N and C$_2$=M^{e_2} mod N then we can compute *M.*

   Proof: For knowing e_1 and e_2, find two integers a and b such that $a \times e_1 + b \times e_2 = 1$ using the Extended Euclidean Algorithm. Then compute: ${C_1}^a{C_2}^b \equiv M^{a \times e_1} M^{b \times e_2} \equiv M^{a \times e_1 + b \times e_2} \equiv M \text{ mod } N$

   This implies that any party can obtain the public keys and the corresponding ciphertexts could be capable to intercept all the messages which would be encrypted twice to different users.

In the case of *Yamen cryptosystem*, the attacker cannot find the message since C'_1 is not the actual C_1, actually $C'_1 = C_1 \oplus Y$ and $C'_2 = C_2 \oplus Y$. Applying the formula ${C'_1}^a {C'_2}^b \bmod N \equiv M'$. The attacker find M' which not the correct message M, since the attacker has C'_1 and C'_2 which are not the actual C_1 and C_2. Therefore, *Yamen cryptosystem* is protected from common modulus attack.

2. Known Plaintext Attack

In this attack, the adversary obtains not only the ciphertext of various messages, but besides the plaintext of those messages. The adversary exploit this type of attacks to get any new captured messages encrypted with the same key.

Proof:
In [84, 85], given: $P_1, P_2, P_3, ..., P_i$ and corresponding ciphertexts $C_1 = E(K, P_1), C_2 = E(K, P_2), ..., C_i = E(K, P_i)$. Build the set $S = \{(P_1, C_1), (P_2, C_2), (P_3, C_3), ..., (P_i, C_i)\}$, where $P_i \in$ plaintext and $C_i \in$ ciphertext.

Because the basic RSA is a deterministic algorithm, encrypting the same message more than once with the same key gives the same ciphertext. Based on the pre-built set S, the adversary can used any later captured data to find the plaintext P_{i+1} if the corresponding C_{i+1} is in the set S. Accordingly, the adversary who obtain partial of plaintext, can guess the other parts.

Yamen cryptosystem is not deterministic, since it depends on random component that makes the ciphertext always different, even if the same message is encrypted more than one time with the same key. The attacker cannot guess the right plaintext-ciphertext pair.

Given: $P_1, C_1 \oplus Y_1 = E(K, P_1), P_2, C_2 \oplus Y_2 = E(K, P_2), ..., P_i, C_i \oplus Y_i = E(K, P_i)$. Where $Y_1 \neq Y_2 \neq Y_3, ..., \neq Y_i$. Since each message is encrypted with different Y, the attacker cannot guess further new plaintext based on a pre-built set of plaintext-ciphertext. Therefore, *Yamen cryptosystem* is protected from *Known plaintext Attack*.

Because $Y_1 \neq Y_2 \neq Y_3, ..., \neq Y_i$, each message has different Y, attackers cannot build a set of plaintext with corresponding ciphertext. Thus, *Yamen cryptosystem* is protected from *Known plain-text attack*.

3. Chosen-Plaintext Attack

Same as *Known plaintext attack*, except that the message is chosen by the adversary.
Given: $\{P_1, C_1 = E(K, P_1)\}, \{P_2, C_2 = E(K, P_2)\}, ..., \{P_i, C_i = E(K, P_i)\}$. Build

the set $S = \{(P_1, C_1), (P_2, C_2), (P_3, C_3), ..., (P_i, C_i)\}$, where $P_i \in$ plaintext and $C_i \in$ ciphertext. Based on the pre-built set S, the adversary can used any later captured data to find the plaintext P_{i+1} if the corresponding $C_{i+1}$is in the set S.

Proof:
Given: $P_1, C_1 \oplus Y_1 = E(K, P_1), P_2, C_2 \oplus Y_2 = E(K, P_2), ..., P_i, C_i \oplus Y_i = E(K, P_i)$. Where $Y_1 \neq Y_2 \neq Y_3, ..., \neq Y_i$. Since each message is encrypted with different Y, the attacker cannot guess further new plaintext based on a pre-built set of plaintext-ciphertext. Therefore, *Yamen cryptosystem* is protected from *Known plaintext Attack.*
Yamen cryptosystem is protected from this type of attack because it depends on a random component, and each message has different Y. That is encrypting the same message more than once produces different ciphers, so attackers can not build a unique set like S that contains plaintext with corresponding ciphertext.

4. Timing Attack

   Kocher in [18] observed that it is possible to obtain the private exponent d without factoring the modulo N by using some knowledge of probability and statistics by measuring the time taken by a hardware, such as smart card or a computer to do the RSA decryption or signature that uses the modular exponentiation algorithm. Proof of the cryptanalytic attacks on RSA can be found in [16].

   In *Yamen cryptosystem* the attacker still needs the random component Y to decrypt the ciphertext, finding d does not allow the attacker to decrypt the message without Y. Also, since Y is XOR-ed with the ciphertext the time is vary depends on the random component and this add some confusion to the attacker.

5. Frequency of Blocks Attack (FOB)

   The basic RSA suffers from *FOB* attack. When one of the blocks repeated within the same message, then the block has the ciphertext similar to that in the first block, the main cause of this problem return to the fact that basic RSA is deterministic algorithm (same message has the same ciphertext). The following example shows how basic RSA not secure against this attack. Suppose the key size is 1024 bits, and the block size 127 bytes for plaintext, message is: (254 bytes which is mean two blocks).

   Eve is one of the attackers women Eve is one of the attackers women Eve is one of the attackers women Eve is one of Eve is one Eve is one of the attackers women Eve is one of the attackers women Eve is one of the attackers women Eve is one of Eve is one
   ciphertext: (similar ciphertext)

   ```
   4053882913553927190690506668220673161420225073939765930225459447848
   ```

```
3282115799689651534813170580476050291062144129409655940405346888192
1736939839169058202563036392391903179066158778280043472508624772649
2525132481258420593884037592215909406040659102274340666328699951450
2390033350405048055647531449064810056597
```

```
4053882913553927190690506668220673161420225073939765930225459447848
3282115799689651534813170580476050291062144129409655940405346888192
1736939839169058202563036392391903179066158778280043472508624772649
2525132481258420593884037592215909406040659102274340666328699951450
2390033350405048055647531449064810056597
```

Yamen cryptosystem is protected from this type of attack, since it compresses the message before the encryption which is made all of the characters in the message has a uniform distribution. The following ciphertext produced for the same message after applying *Yamen* cryptosystem.

ciphertext:(each block has different ciphertext)

```
2284419579637540474199177876330350409831242577008055804649922610692
9245927132456833007726603602225646033540909831360219319114207313778
9369250486972759219798253776040409747725396231205670779564085983040
7637374593055233764837387354176158316097812091477351593320923805486
0496880956721402740261189602575811769744
```

```
9867466537872031755884492656409497450867000762637858322895785630503
1509845130341447379726550799975759941856995112902800716411844281778
5063152897674454198732730809501185749758902123962038647028000572531
6097770097906489985877906710530401328785349299993340141988935516494
2961708957998119411447113878266731369808
```

Table 5.1 illustrates how *Yamen cryptosystem* immunizes the presented attacks in section 3.1.

	Attacks Against RSA	Mitigation Approach	Attack Possibility Against Yamen
1	Common Modulus	Using randomized component Y	Not possible
2	Known Plaintext		
3	Chosen-Plaintext		
4	Timing		
5	Frequency of Blocks Attack	Using Huffman coding	Not possible
6	Brute Force Attack	Using Rabin cryptosystem	May possible but needs long time

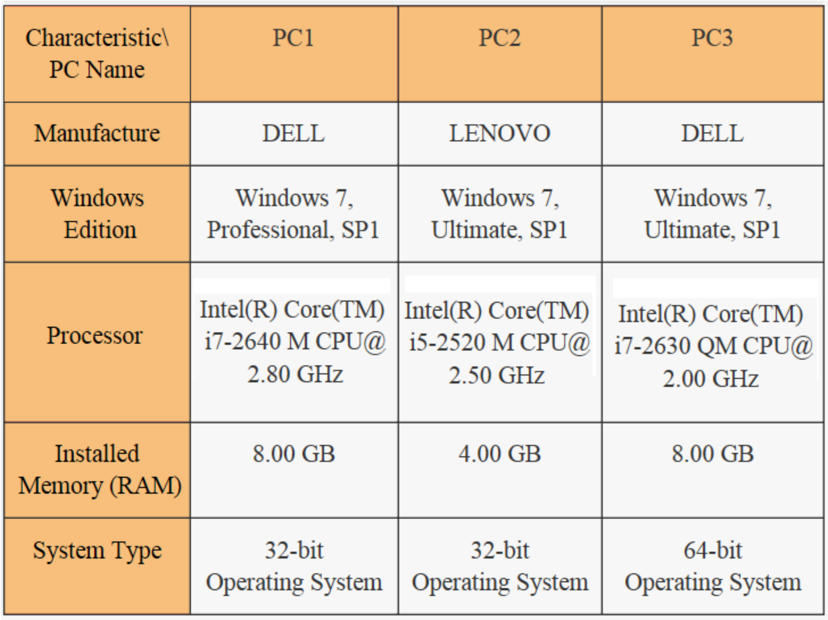
TABLE 5.1: Yamen Cryptosystem is Immune Against the RSA Attacks

5.2.2 Execution Time Issue

RSA is used for exchanging symmetric keys between two parties far away from each other in a secure manner, then the communicating parties use a symmetric encryption algorithm for encrypting data depends on that symmetric keys. Since RSA is not practical and takes long time for encrypting and decrypting large data [86, 87].

Yamen cryptosystem can be used for encrypting large data. Therefore, there is no need for using two algorithms one for exchanging keys and the other for encrypt data, all larges messages will be reduced to small one by using *Yamen* system. In addition *Yamen* consumes less time than RSA for encryption, since in *Yamen* encrypts only the header file and blinding the binary file instead of encrypting the entire message. *Yamen* decryption process is faster than the decryption in RSA. *Yamen* encrypts and decrypts only the header file but the basic RSA needs to encrypt and decrypt the entire file.

To test the performance of *Yamen cryptosystem* comparing to basic RSA, we carried out three experiments on different PCs. Table 5.2 shows the characteristic for each PC.



Characteristic\ PC Name	PC1	PC2	PC3
Manufacture	DELL	LENOVO	DELL
Windows Edition	Windows 7, Professional, SP1	Windows 7, Ultimate, SP1	Windows 7, Ultimate, SP1
Processor	Intel(R) Core(TM) i7-2640 M CPU@ 2.80 GHz	Intel(R) Core(TM) i5-2520 M CPU@ 2.50 GHz	Intel(R) Core(TM) i7-2630 QM CPU@ 2.00 GHz
Installed Memory (RAM)	8.00 GB	4.00 GB	8.00 GB
System Type	32-bit Operating System	32-bit Operating System	64-bit Operating System

TABLE 5.2: Systems Properties

We select N, d and e as below:

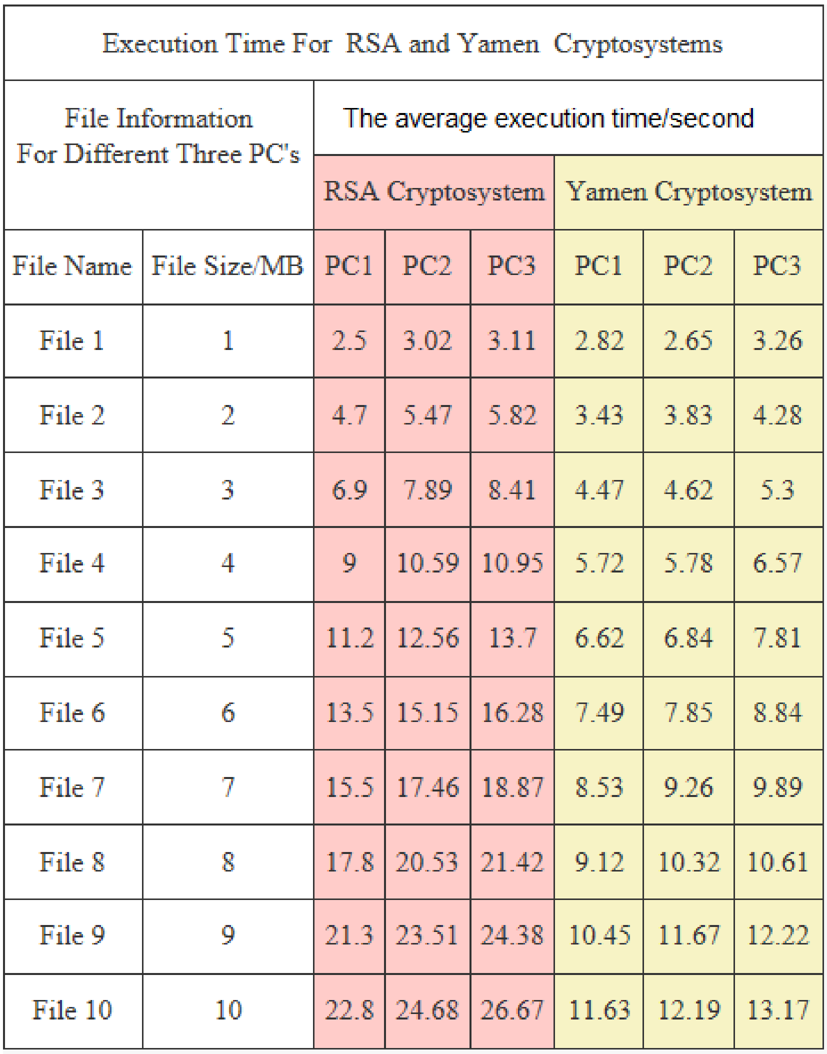
We select N= pq be the product of two large primes of the same size ($n/2$ bits each). N=1024 bits, i.e. 309 decimal digits, where p=512 bits and q=512 bits.

RSA modulo (N):
114182124423925571361043639556535604438240742181297672510878199076
918140197799380138229518857632779498535203863157511896892122416732
533192329558431299159176326515525586122202347184358738524535511184
314934775231769225233918596533563686874486037306537022338789262140
62922550133755886562848038808678002455068831

RSA public exponent (e):
65537 (standard public exponent)

RSA private exponent (d):
166402714554055439228119657812300159908088153039286825785647447912
422777519444265025898368037787917815968038222228267410350895097763
465602627464268632721834278448219311328890068866109537678429602966
152465707355953090505744976911681260706338107018743044908710508040
96933553719368719860127462476578927521870593

The following tables and charts, show how *Yamen* system is significantly faster than RSA at the three different PCs.



Execution Time For RSA and Yamen Cryptosystems							
File Information For Different Three PC's		The average execution time/second					
		RSA Cryptosystem			Yamen Cryptosystem		
File Name	File Size/MB	PC1	PC2	PC3	PC1	PC2	PC3
File 1	1	2.5	3.02	3.11	2.82	2.65	3.26
File 2	2	4.7	5.47	5.82	3.43	3.83	4.28
File 3	3	6.9	7.89	8.41	4.47	4.62	5.3
File 4	4	9	10.59	10.95	5.72	5.78	6.57
File 5	5	11.2	12.56	13.7	6.62	6.84	7.81
File 6	6	13.5	15.15	16.28	7.49	7.85	8.84
File 7	7	15.5	17.46	18.87	8.53	9.26	9.89
File 8	8	17.8	20.53	21.42	9.12	10.32	10.61
File 9	9	21.3	23.51	24.38	10.45	11.67	12.22
File 10	10	22.8	24.68	26.67	11.63	12.19	13.17

TABLE 5.3: Execution Time of Encryption Process For RSA and Yamen Cryptosystems

Table 5.3, shows the average execution time for encrypting ten files on different PC's by using two cryptosystems, *basic RSA* algorithm and *Yamen cryptosystem*. Each file is encrypted three times, so we compute the average of these reading times. We notice that the execution time directly proportional to the file size. In other words, if the file size increase then the execution time for encrypting the file increase too. The reason that *Yamen* system is faster than *RSA* refers to the fact that *Yamen* does not encrypt the entire message, it encrypts the header file and blinding the binary file which is generated from compression phase.

Figure 5.1, represents a line chart to show the behavior of *RSA* and *Yamen* in encryption process through different files on different PC's. Actually, the figure represents the average execution time. That is, each point on the chart represents the average of execution time for three PC's for each cryptosystem. For instance, the point (1, 2.88) represents the average of (2.5,3.02,3.11) for *RSA*, and the point (1,2.91) represents the average of (2.82,2.65,3.26) for *Yamen*.

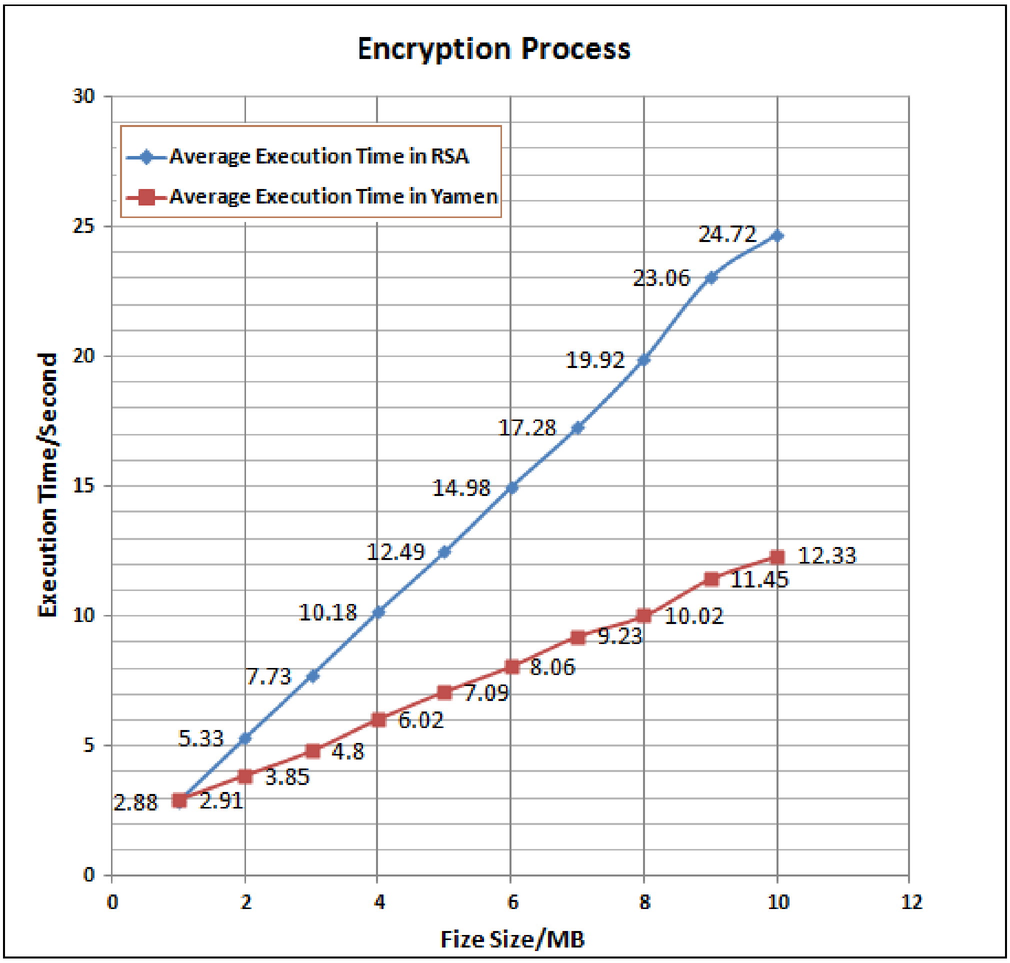
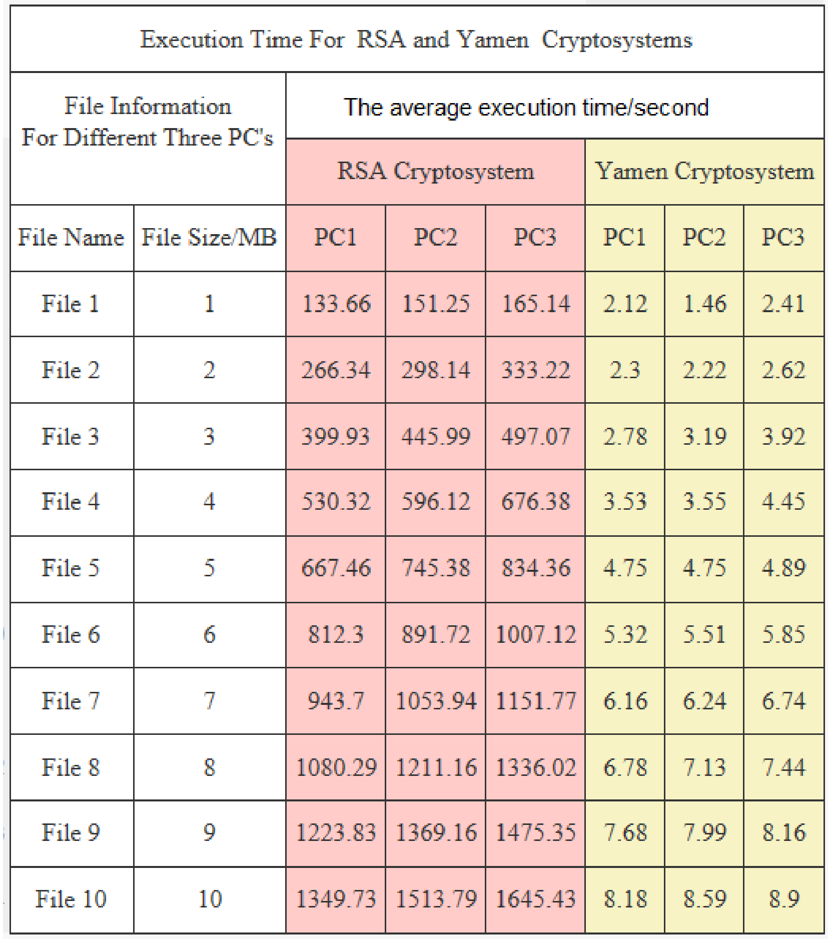


FIGURE 5.1: Line Chart For Execution Time of Encryption Process For RSA and Yamen Cryptosystems

Figure 5.1 shows that *Yamen* is slower than *RSA* in encryption process when the file is one megabyte. Such that, *Yamen* is 2.91s, while *RSA* is 2.88s when the file is 1MB. The reason behind this returns that *Yamen* encryption process passes through four phases: analyzing (doing statistical analysis for data), compressing, encrypting and blinding data. However, since the file is relatively small the analyzing, compressing and blinding may take more time than the encryption itself.

Table 5.4, shows the average execution time for decrypting ten files on different PC's by using two cryptosystems, *basic RSA* algorithm and *Yamen cryptosystem*.



Execution Time For RSA and Yamen Cryptosystems							
File Information For Different Three PC's		The average execution time/second					
		RSA Cryptosystem			Yamen Cryptosystem		
File Name	File Size/MB	PC1	PC2	PC3	PC1	PC2	PC3
File 1	1	133.66	151.25	165.14	2.12	1.46	2.41
File 2	2	266.34	298.14	333.22	2.3	2.22	2.62
File 3	3	399.93	445.99	497.07	2.78	3.19	3.92
File 4	4	530.32	596.12	676.38	3.53	3.55	4.45
File 5	5	667.46	745.38	834.36	4.75	4.75	4.89
File 6	6	812.3	891.72	1007.12	5.32	5.51	5.85
File 7	7	943.7	1053.94	1151.77	6.16	6.24	6.74
File 8	8	1080.29	1211.16	1336.02	6.78	7.13	7.44
File 9	9	1223.83	1369.16	1475.35	7.68	7.99	8.16
File 10	10	1349.73	1513.79	1645.43	8.18	8.59	8.9

TABLE 5.4: Execution Time of Decryption Process For RSA and Yamen Cryptosystems

The average of three reading for each file are depicted in the Table 5.4. The execution time is directly proportional to the file size. If the file size increase then the execution time for decrypting file increase too. There is a big difference in the execution time between encryption and decryption, this return to the two reasons. First, private exponent d is larger than public exponent e. Second, when encrypting the file using *RSA*; the encrypted file (ciphertext) is greater than original file. We discuss the file size in section 5.2.3.

By comparing the Table 5.3 with Table 5.4, you see that the execution time of the decryption process in *Yamen* less than the encryption time for the same cryptosystem. The behavior of *RSA* and *Yamen* in decryption process is depicted in Figure 5.2.

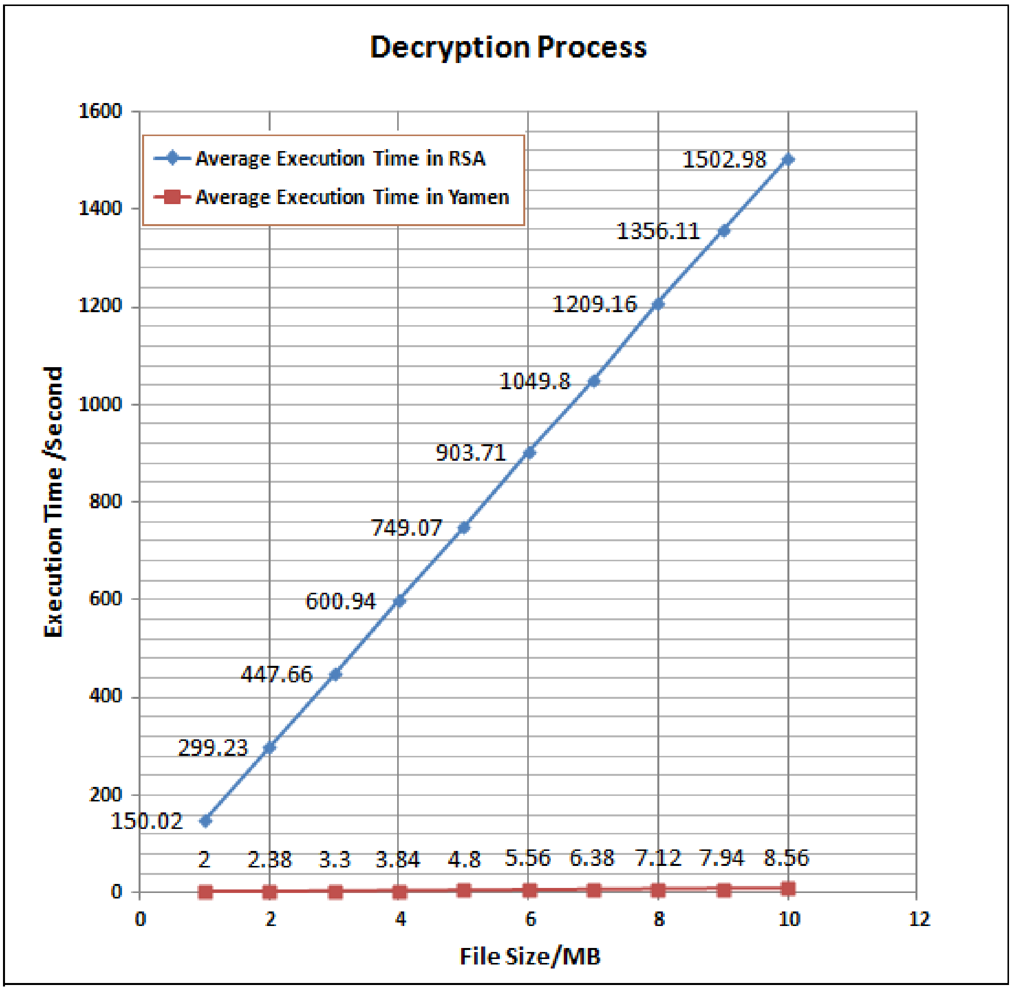


Figure 5.2: Line Chart For Execution Time of Decryption Process For RSA and Yamen Cryptosystems

Generally, the encryption is faster than decryption or may equally, but this is not the case for *Yemen* system as show in Figure 5.2. The reason for this observation is the existence of four phases in *Yamen* encryption process (analyzing, compression, encryption and blinding). On the other hand, there are three phases in *Yamen* decryption process that make its decryption faster (un-blinding, decryption and un-compression). However, in all cases *Yamen* is faster than the basic RSA in decryption process. RSA needs to decrypt the entire message, but *Yamen* just needs to decrypt the header file which is smaller than the entire message.

Our testing results from the three experiments shows that, *Yamen* cryptosystem is faster than basic RSA by 45% in encryption process and 99% in decryption process.

Note: This section represents just the average of three reading files through discussion in the process of encryption and decryption. Three different reading for each file can be found in detail in Appendix B.

5.2.3 Space Issue

Yamen cryptosystem uses *Huffman* compression algorithm to reduce file sizes, so the files generated from *Yamen* system are smaller than the original files, which is helping for reducing resource usage, such as data storage space or transmission capacity. Table 5.5 shows the size of ten files what happened to the size after the encryption by using RSA and *Yamen cryptosystems*.

Table 5.5 RSA and Yamen Ciphertext Table

Original File		Ciphertext of RSA and Yamen Cryptosystems	
Size in MB	**Size in KB**	**RSA Ciphertext in KB**	**Yamen Ciphertext in KB**
1	1024	1033	469
2	2048	2065	938
3	3072	3097	1407
4	4096	4129	1876
5	5120	5161	2344
6	6144	6193	2813
7	7168	7225	3282
8	8192	8257	3751
9	9216	9289	4219
10	10240	10321	4688

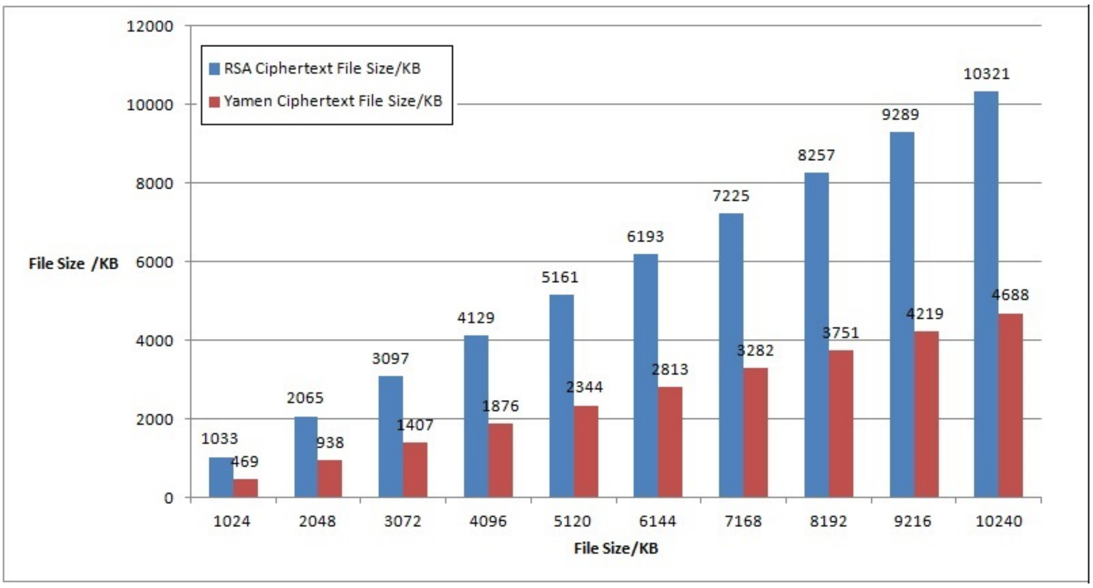


Figure 5.3: RSA and Yamen Ciphertext Size

Figure 5.3 demonstrates that *Yamen cryptosystem* is the better choice for encrypting large data in a secure and fast manner over a public network. *Yamen cryptosystem* results in an efficient use of the space since *RSA* system increases the size of ciphertext by 1% compared to the original file size, while *Yamen cryptosystem*, reducing the the size of the ciphertext by 54% compared to the original file size. However, the percentage of file size reduction by using *Yamen cryptosystem* depends on the number of occurrences of the symbols inside the file, because using the Huffman coding in compression process, and Huffman in role depends on the symbols occurrence inside the file.

Note: This section represents the sizes of the files without details. The size of each header file and binary file exists in Appendix B in Table B.1.

5.3 Yamen Cryptosystem Comparing with the Basic RSA

The enhanced version of RSA is more secure, faster, and has shorter encrypted message size comparing with the basic RSA. *Yamen* cryptosystem has the following characteristics:

1. Semantically secure compering to the basic RSA.

2. Secure against *Frequency of Block Attacks.* According to Shannon[88, 89], reducing the redundancy of data before encrypting it improve the security of a cryptosystem. Thus, Huffman code reduces the redundancy of data.

3. More secure against *Brute Force* attack than the RSA, since the attacker needs to break the factorization of large numbers for both RSA and Rabin. Consequently, the attacker would require longer time than the one for breaking the basic RSA.

4. Faster encryption and decryption compare with basic RSA, since RSA encrypts the entire message, while *Yamen* encrypts the header file and blind the binary file instead of encrypting the entire message.

5. Short cipher size. *Yamen* system compresses data before the encryption process, this could facilitate storing and transmission large data, while we cannot do that in the RSA.

6. Capability for encrypting large data. Because RSA is much slower in encryption and decryption process comparing to symmetric encryption techniques, it is used for encrypting the keys that transmitted between two parities while the whole message encrypted by using symmetric encryption such as, *AES* encryption algorithm. While in *Yamen* system no need to any symmetric algorithm for encrypting the data.

5.4 Chapter Summary

We have discussed three sensitive factors that impact the usage of the public key cryptosytems, *Security*, *Execution time* and the *Size of the ciphertext.* These factors were enhanced by *Yamen cryptosystem.* We did three experiments on different PCs and we have found that *Yamen* system superior the basic RSA. *Yamen* is more secure, faster than *RSA*. Also, We found that *Yamen* system is better than RSA related to the space issue especially if we have a small bandwidth or small space storage.

Chapter 6

Summary and Outlook

RSA is popular and most widely used cryptosystem through the years until now, it is used in different applications and protocols. Even though, RSA seems to be robust and secure public key algorithm, attackers succeeded to exploit some properties of RSA algorithm and its implemetation to carry out some attacks. Such attacks are *Common Modulus*, *Known plaintext*, *Chosen-plaintext*, and *Timing* attacks. Moreover, *RSA* does not solved blocks redundancy in the message, hence *RSA* suffers from *Frequency of Block* attack. Also, RSA is a time consuming algorithm, its speed very slow comparing with symmetric key encryption algorithms like *AES* algorithm.

The thesis presented the earlier proposed solutions used to solve or at least to mitigate the RSA attacks. Also, it has categorized them into three categories based on their techniques into *Dice solution*, *Dice solution follower*, and *Hungry mouse solution*. In addition, the thesis described the limitations of these solutions.

We suggested a new modifications to enhance *RSA* cryptosystem called *Yamen cryptosystem*. This enhanced RSA combines basic RSA with another cryptosystem called *Rabin*. Rabin is used to secure the randomized component, this component is added to the RSA and used to make it semantically secure, to get different ciphertexts for the same message. Since the message may have redundant characters, we used a Huffman coding algorithm to remove the redundancy in the message. For minimizing encryption/ decryption time of the enhanced RSA, we encrypt part of the message which is called the header file and blind the other part which is called a binary file instead of the encrypting the entire message. Binary file and Header file are produced as output from the Huffman coding algorithm, and this algorithm is based on statistical analysis to generate these files.

The thesis states some evidence and mathematical proofs to show that *Yamen* system is more secure than basic RSA. In addition, some experiments done and shows that *Yamen* system is faster and more effective when it is used for transferring large encrypted text over public networks.

We found that *Yamen cryptosystem* more secure than RSA. *Yamen* has different ciphertexts for the same message compered to the basic RSA that has the same ciphertext for the same message. Our testing results showed that *Yamen cryptosystem* is faster than basic *RSA* by 45% in encryption process and 99% in decryption process. Also, we found that *RSA* system increases the size of ciphertext by 1% compared to the original file size, while *Yamen cryptosystem*, reducing the size of ciphertext by 54% from the original size. However, the percentage of file size reduction by using *Yamen cryptosystem* depends on the number of occurrences of the symbols inside the file.

6.1 Summarization of Yamen Characteristics Against RSA Attacks and Other Proposed Approaches

Table 6.1 summarize the security of *Yamen* against *RSA* attacks. Table 6.2 summarize the comparison results of *Yamen* against other proposed approaches to solve *RSA* attacks.

Table 6.1 illustrates how *Yamen cryptosystem* immunizes the presented attacks in chapter 3 (section 3.1).

Table 6.1 Yamen Cryptosystem is Immune against the RSA Attacks

	Attacks Against RSA	Mitigation Approach	Attack Possibility Against Yamen
1	Common Modulus	Using randomized component Y	Not possible
2	Known Plaintext		
3	Chosen-Plaintext		
4	Timing		
5	Frequency of Blocks Attack	Using Huffman coding	Not possible
6	Brute Force Attack	Using Rabin cryptosystem	May possible but needs long time

Table 6.2 shows the comparison between the *Yamen* cryptosystem and other proposed approaches.

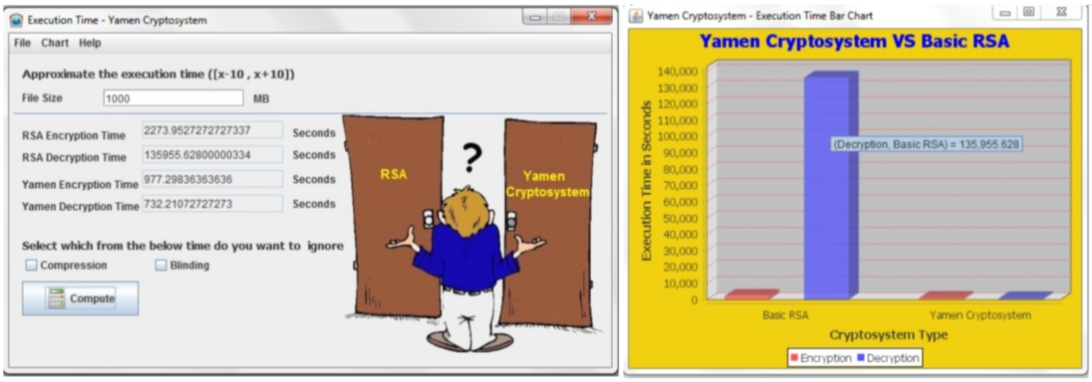
Table 6.2 Yamen Cryptosystem Comparing with Other Proposed Approaches

Proposed approaches	**Approaches Advantages**	**Approaches Limitations**
Dice approach, section 3.2.	1. Semantically secure algorithm. 2. Solves the common modules attack in a good way comparable to the dice follower approach.	1.Uses the same RSA to encrypt the random component Y not secure enough. since, breaking the RSA factorization mean breaking this randomized component Y. 2.Adds additional delay to the RSA encryption process. 3.Suffers from the FOB attack.
Dice approach follower, section 3.2.	Semantically secure algorithm.	1. Solves the common modules attack by never send the exact messages to receivers is insufficient, since we cannot send an emergency message 3.3. 2.Includes the disadvantages of Dice approach.
Hungry mouse approach, section 3.2.	1.Semantically secure algorithm. 2.The size of the randomized component is complex comparing to the Dice approach and Dice approach follower.	1. If the message greater than N, the algorithm no longer works. 2. Adds additional delay to the RSA encryption process. 3. Suffers from the FOB attack.
Yamen cryptosystem, section 5.2.	1.Semantically secure algorithm. 2.Generates complex ciphertext. 3.Competitive execution time. 4.Produce smaller size of ciphertext. 5.Solves FOB attack. 6.Stays for a long time against brute force attack.	When the file is less than 1MB RSA is faster in encryption process.

6.2 Difficulties and Obstacles

This section points to the difficulties and obstacles, that have been faced when doing this thesis. These difficulties summarized as the following:

RSA is much slower public key encryption algorithm. A lot of time was consumed when doing the experiments. Most of the time lost during the decryption process. For this reason, we execute our experiments up to the ten files with different sizes starting with one megabyte up to the ten megabytes. We cannot execute the experiments on large files, since we have no enough time to wait *RSA* to decrypt large size files. Depends on our results, we made a correlation between the times we got from the experiments and the predicted time. By using *linear regression* numerical method, we develop a method for predicting the required time for encryption/decryption of any file depends on our PC characteristics. Figure 6.1 presents the prediction method in *Yamen* system to find the execution time for 1000MB using *Yamen* and *RSA*.



Execution Time - Yamen Cryptosystem

Approximate the execution time ([x-10 , x+10])

File Size 1000 MB

RSA Encryption Time 2273.9527272727337 Seconds
RSA Decryption Time 135955.62800000334 Seconds
Yamen Encryption Time 977.29836363636 Seconds
Yamen Decryption Time 732.21072727273 Seconds

Select which from the below time do you want to ignore

□ Compression □ Blinding

Yamen Cryptosystem - Execution Time Bar Chart

(Decryption, Basic RSA) = 135,955.628

Figure A

Figure B

FIGURE 6.1: Execution time for Yamen and RSA

As the Figure 6.1, execution time for *RSA* encryption is about 2273.95 seconds, and the time decryption is about 135955.628 seconds. while *Yamen* required 977.29 seconds in encryption, and 732.2107 seconds in decryption.

We did not find a lot of resources about using *RSA* to encrypt large messages. Maybe the reason refers to the fact that *RSA* is used to encrypt the session keys instead of encrypts the true message.

We did not find a lot of resources about using *Rabin* algorithm. Maybe the reason refers to the fact that *Rabin* is not becoming popular, because *RSA* was developed first. Also, Rabin is not a practical algorithm to use for encrypting messages. Since, extra complexity is used to identify the corresponding plaintext from the four possible roots.

To benefit and to make *Rabin* practical, it has been used for encrypting only the random component instead of entire messages.

6.3 Recommendations

Depends on the experiments results presented before in this thesis, we recommend to use *Yamen cryptosystem* in the following cases:

- For using semantic and secure algorithm to work in fast way for encrypting or decrypting large data over a public network.
- For sending the same sensitive data to more than one party.
- For encrypting data without the use of symmetric and asymmetric keys algorithms together.

6.4 Outlook

The presented results in this thesis have demonstrated the effectiveness of the *Yamen* cryptosystem approach, it could be further developed in a number of ways:

- Replace *Huffman coding* with extended *Huffman coding* may increase the *Yamen* cryptosystem execution time for encryption and decryption process.
- Use another secure random generator rather than *congruential generator*, may increase the security of *Yamen* cryptosystem.
- Develop new version of *Yamen* cryptosystem to work on mobile.

The research presented in this thesis seems to have raised more questions that it has answered, but there are many questions we can answer by doing the following:

- Integrate Yamen with security protocols such as PGP and test the performance and security implications.
- Make compresson between *Yamen* cryptosystem and *symmetric key encryption* technique as *AES*.
- Testing *Yamen* cryptosystem against others indirect attacks such as, *Low Private Exponent*, *Low Public Exponent* to make it more secure.
- Testing *Yamen* cryptosystem against *Chosen ciphertext attack* to mitigate it.

Appendix A

Snapshots for Yamen Cryptosystem Processes

There are two views in *Yamen* system, *Educational* view and *Business* view. The below figures show the encryption process for *Yamen* cryptosystem design:

A.1 Encryption Process: Educational View

Phase 1: Generate Keys and Random Component

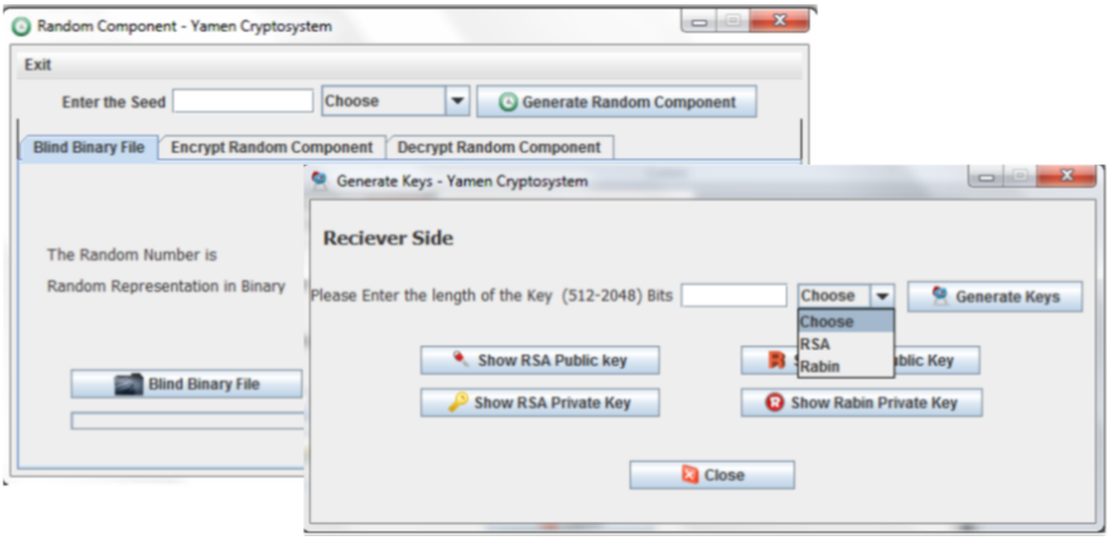
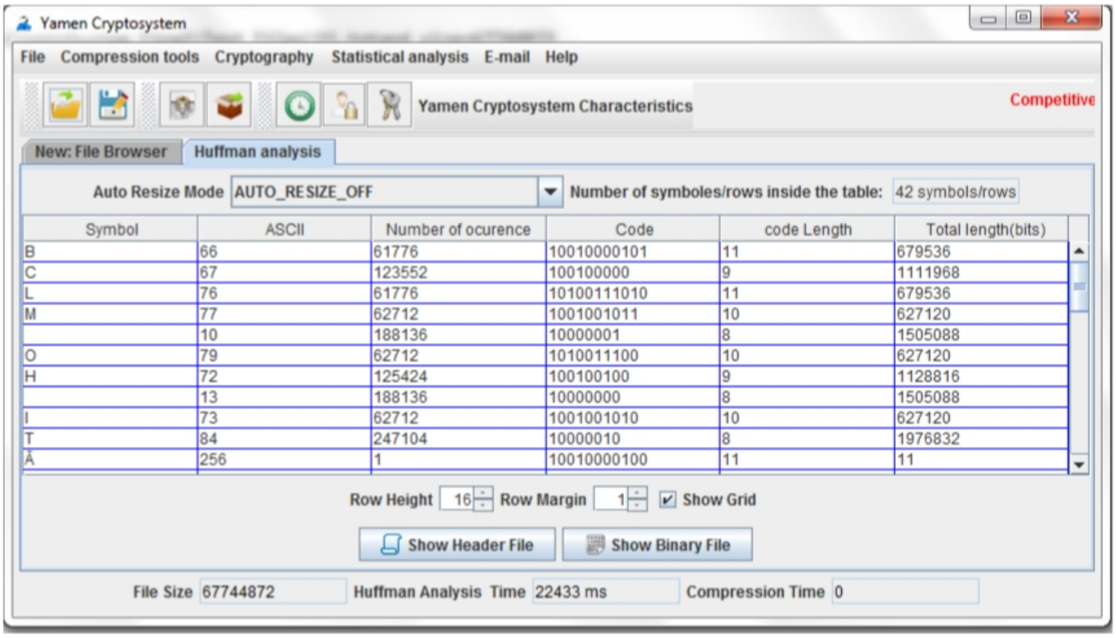


Figure A.1: Generate Keys and Random Component Process

Phase 2: Compression Process



Auto Resize Mode AUTO_RESIZE_OFF

Number of symboles/rows inside the table: 42 symbols/rows

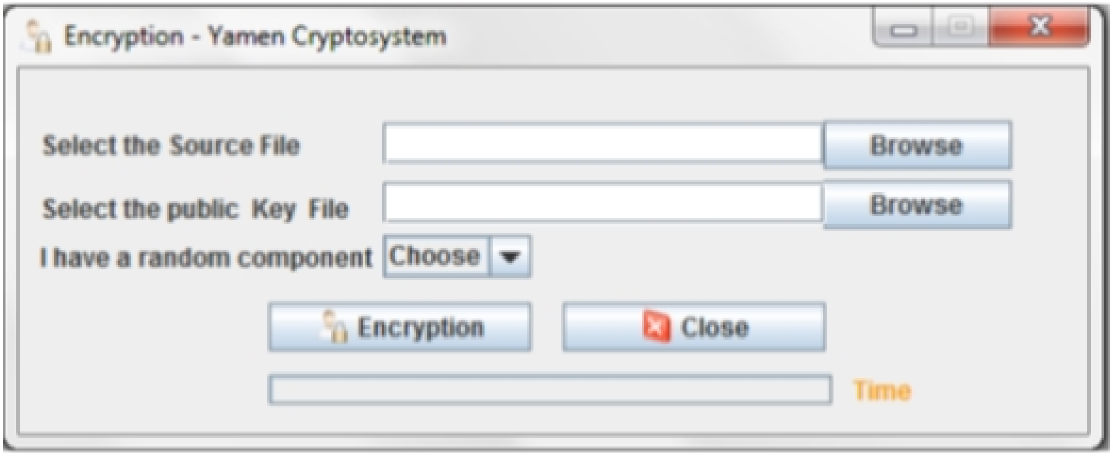
Symbol	ASCII	Number of ocurence	Code	code Length	Total length(bits)
B	66	61776	10010000101	11	679536
C	67	123552	100100000	9	1111968
L	76	61776	10100111010	11	679536
M	77	62712	1001001011	10	627120
	10	188136	10000001	8	1505088
O	79	62712	1010011100	10	627120
H	72	125424	100100100	9	1128816
	13	188136	10000000	8	1505088
I	73	62712	1001001010	10	627120
T	84	247104	10000010	8	1976832
Ā	256	1	10010000100	11	11

Row Height 16 Row Margin 1 ☑ Show Grid

File Size 67744872 Huffman Analysis Time 22433 ms Compression Time 0

Figure A.2: Yamen Cryptosystem Compression Process

Phase 3: Encryption Process



Select the Source File

Select the public Key File

I have a random component Choose

Figure A.3: Yamen Cryptosystem Encryption Process

Phase 4: Blinding Process

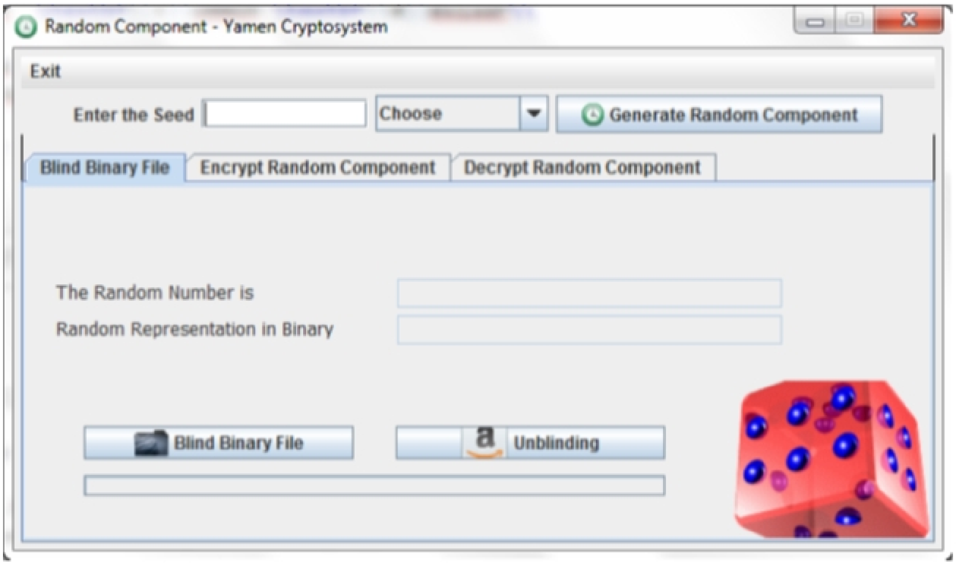
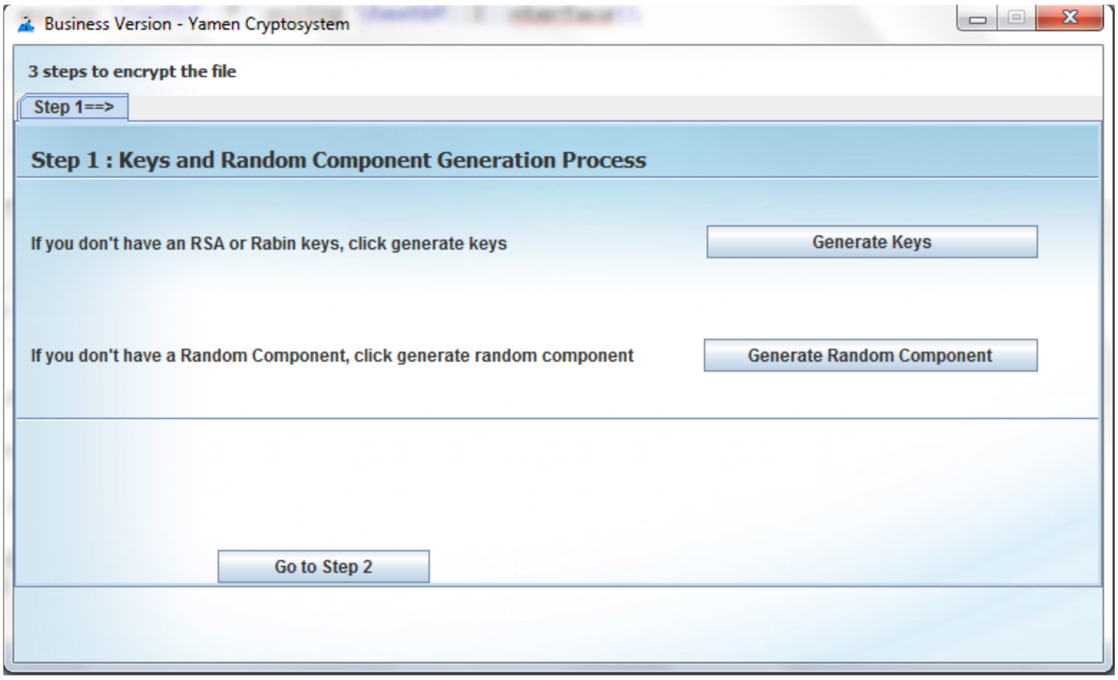


FIGURE A.4: Blinding Process

A.2 Encryption Process : Business View

Phase 1: Keys and Random Component Generation


Business Version - Yamen Cryptosystem
3 steps to encrypt the file
Step 1==>

Step 1 : Keys and Random Component Generation Process

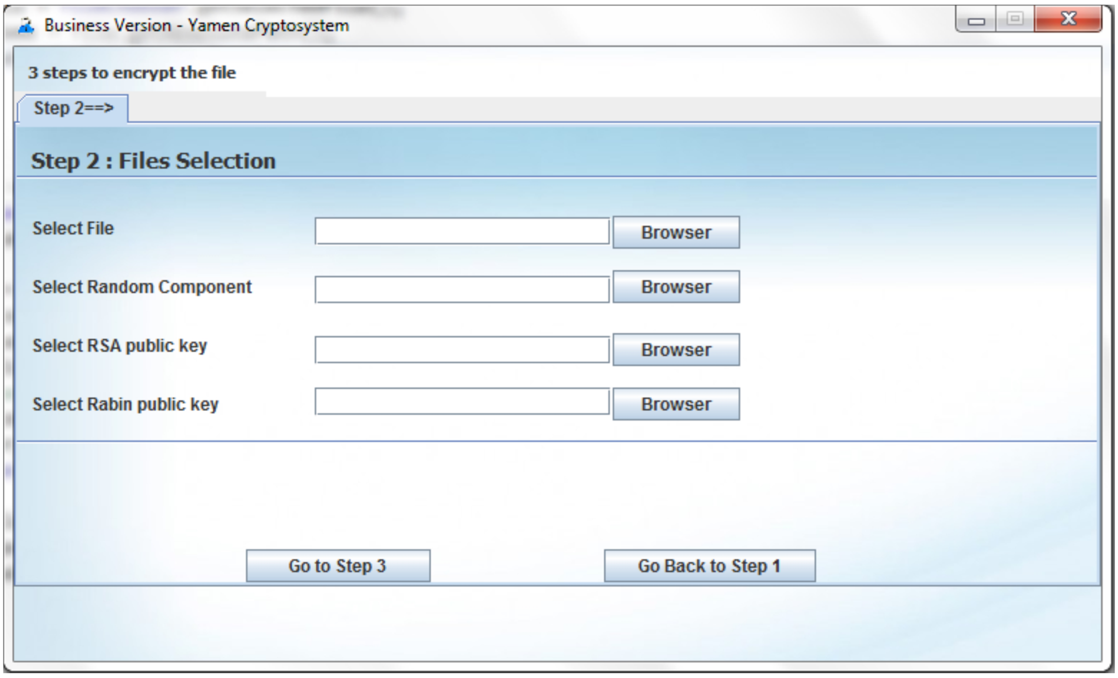
If you don't have an RSA or Rabin keys, click generate keys

If you don't have a Random Component, click generate random component

Generate Random Component
Go to Step 2

FIGURE A.5: Keys and Random Component Generation

Phase 2: Files Selection Process



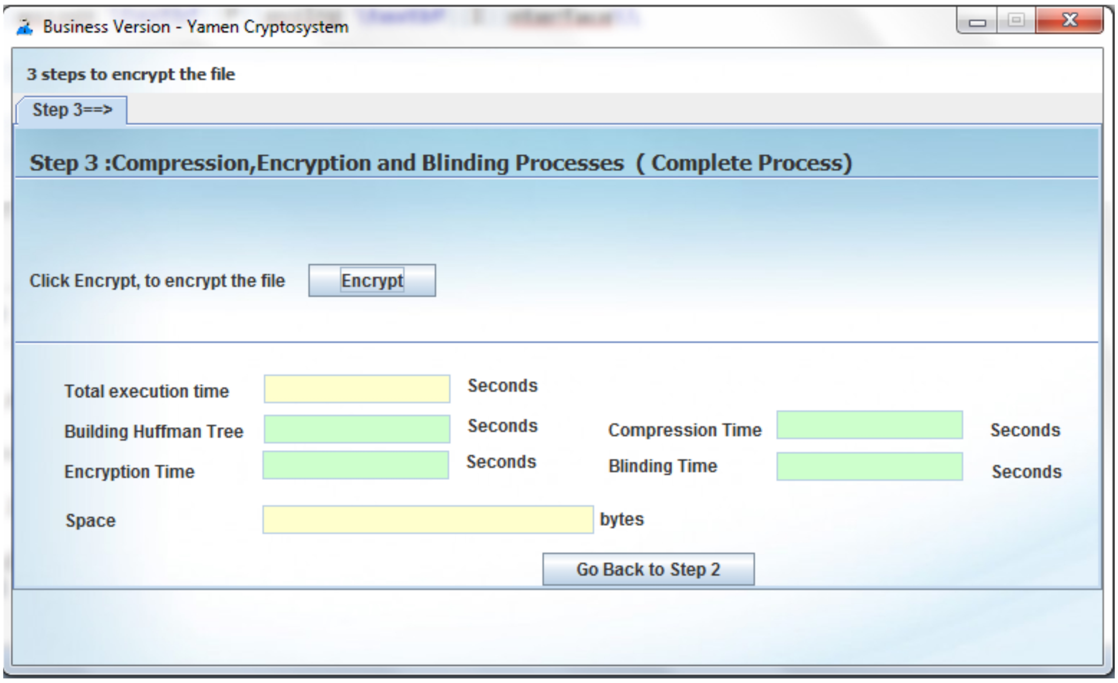
Business Version - Yamen Cryptosystem

3 steps to encrypt the file

Step 2 : Files Selection

Figure A.6: Files Selection Process

Phase 3: Generate Encryption File Process



Business Version - Yamen Cryptosystem

3 steps to encrypt the file

Step 3 :Compression,Encryption and Blinding Processes (Complete Process)

Click Encrypt, to encrypt the file

Figure A.7: Generate Encryption File Process

Appendix B

Experiments : Details Tables And Charts To Show That Yamen Cryptosystem Is Faster Than RSA.

In this appendix, we describe our experiments result in detailed tables and charts. These experiments, show that *Yamen cryptosystem* is faster than *RSA*.

Table B.1 The size of Header File and Binary File Generated From Huffman Coding

File Size in Kilobyte	File Size in Byte	Header File Size in Byte	Binary File Size in Byte
1024	1024000	228	480013
2048	2048000	232	960025
3072	3072000	235	1440037
4096	4096000	235	1920049
5120	5120000	235	2400061
6144	6144000	236	2880073
7168	7168000	236	3360084
8192	8192000	237	3840096
9216	9216000	237	4320108
10240	10240000	243	4800120

Table B.1, shows the size of header file and binary file for the ten tested files. each header file contains signature as *#YamenCryptosystem, Huffman counts file for fileName, #Fri Jan 23 19:27:45 IST 2015*. Date and time change based on the time and date the file compress.

Table B.2 Execution Time of RSA Encryption From PC1

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	2.513	2.607	2.399	2.5
File 2	2	4.804	4.643	4.671	4.7
File 3	3	6.861	7.026	6.729	6.9
File 4	4	8.956	9.07	8.886	9
File 5	5	11.051	11.249	11.32	11.2
File 6	6	13.68	13.369	13.54	13.5
File 7	7	15.462	15.668	15.25	15.5
File 8	8	17.96	17.558	17.953	17.8
File 9	9	21.717	21.401	20.687	21.3
File 10	10	22.831	22.872	22.647	22.8

Table B.2, shows the average execution time for encrypting ten files by using the basic *RSA* algorithm. Each file is encrypted three times, so we compute the average of these reading times.

Table B.3 Execution Time of RSA Decryption From PC1

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	134.27	133.13	133.579	133.66
File 2	2	264.424	268.911	265.699	266.34
File 3	3	400.65	397.664	401.489	399.93
File 4	4	530.272	530.585	530.096	530.32
File 5	5	669.692	663.263	669.417	667.46
File 6	6	822.441	804.51	809.934	812.3
File 7	7	948.129	938.223	944.748	943.7
File 8	8	1075.878	1086.319	1078.683	1080.29
File 9	9	1221.505	1210.76	1239.224	1223.83
File 10	10	1349.759	1350.657	1348.766	1349.73

Table B.3, shows the average execution time for decrypting ten files using basic *RSA* algorithm. We compute the average of three reading for each file as in the encryption process.

Table B.4 Execution Time of Yamen Encryption From PC1

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	2.905	2.804	2.751	2.82
File 2	2	3.498	3.42	3.378	3.43
File 3	3	4.29	4.653	4.468	4.47
File 4	4	5.61	5.692	5.857	5.72
File 5	5	6.736	6.448	6.675	6.62
File 6	6	7.297	7.279	7.902	7.49
File 7	7	8.612	8.907	8.071	8.53
File 8	8	9.072	9.231	9.065	9.12
File 9	9	10.546	10.293	10.512	10.45
File 10	10	11.683	11.418	11.786	11.63

Table B.4, shows the average execution time for encrypting ten files by using *Yamen* algorithm. Each file is encrypted three times, so we compute the average of these reading times.

Table B.5 Execution Time of *Yamen* Decryption From PC1

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	2.296	1.804	2.252	2.12
File 2	2	2.116	2.282	2.506	2.3
File 3	3	2.483	2.718	3.144	2.78
File 4	4	3.326	3.586	3.684	3.53
File 5	5	4.463	4.948	4.848	4.75
File 6	6	5.29	5.199	5.458	5.32
File 7	7	6.028	6.077	6.373	6.16
File 8	8	6.495	6.877	6.979	6.78
File 9	9	8.182	7.497	7.37	7.68
File 10	10	8.427	8.458	7.662	8.18

Table B.5, shows the average execution time for decrypting ten files using *Yamen* algorithm. We compute the average of three reading for each file as in the encryption process.

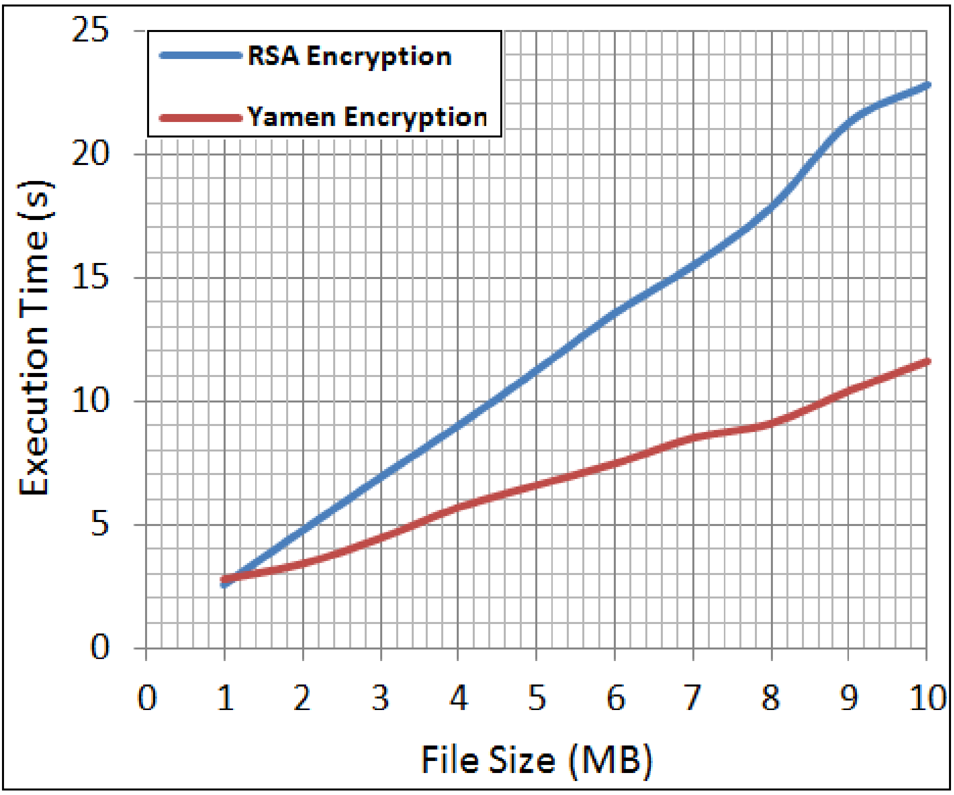


Figure B.1: Execution Time for RSA and *Yamen* Encryption Process (PC1)

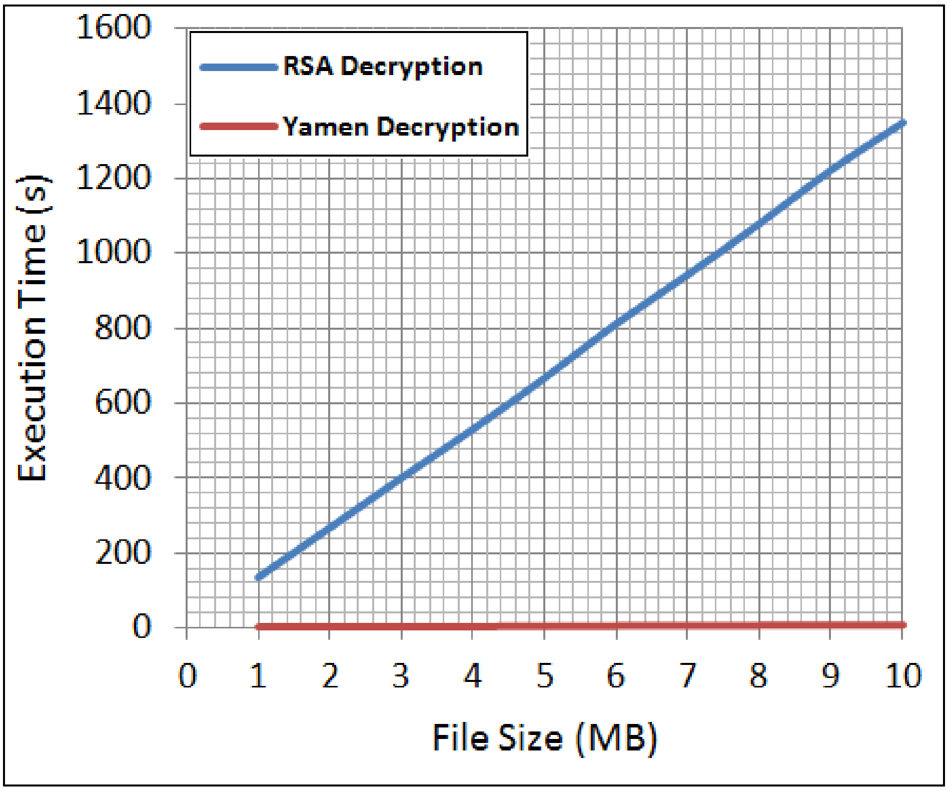


Figure B.2: Execution Time for RSA and Yamen Decryption Process (PC1)

The following tables and graphs, are another two experiments running in two different PC's with different characteristics to show that *Yamen* cryptosystem is faster than RSA.

Table B.6 Execution Time of RSA Encryption From PC2

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	2.855	2.906	3.285	3.02
File 2	2	5.111	5.732	5.569	5.47
File 3	3	7.75	7.806	8.099	7.89
File 4	4	10.625	10.757	10.384	10.59
File 5	5	12.139	12.572	12.954	12.56
File 6	6	15.443	14.744	15.268	15.15
File 7	7	17.413	17.863	17.112	17.46
File 8	8	20.101	20.352	21.136	20.53
File 9	9	23.363	24.081	23.092	23.51
File10	10	24.842	24.392	24.813	24.68

Table B.7 Execution Time of RSA Decryption From PC2

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	150.547	152.07	151.132	151.25
File 2	2	297.689	297.969	298.747	298.14
File 3	3	444.772	448.098	445.102	445.99
File 4	4	597.925	593.324	597.109	596.12
File 5	5	746.158	743.055	746.916	745.38
File 6	6	892.343	889.634	893.173	891.72
File 7	7	1046.371	1055.765	1059.693	1053.94
File 8	8	1207.236	1213.067	1213.18	1211.16
File 9	9	1369.996	1369.951	1367.53	1369.16
File 10	10	1515.706	1512.852	1512.798	1513.79

Table B.8 Execution Time of Yamen Encryption From PC2

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	2.419	2.935	2.597	2.65
File 2	2	3.888	3.898	3.705	3.83
File 3	3	4.776	4.608	4.488	4.62
File 4	4	5.484	5.581	6.27	5.78
File 5	5	7.223	6.835	6.447	6.84
File 6	6	7.867	7.732	7.96	7.85
File 7	7	9.363	9.406	9.003	9.26
File 8	8	10.159	10.804	10.009	10.32
File 9	9	11.89	11.237	11.888	11.67
File 10	10	12.608	12.063	11.892	12.19

Table B.9 Execution Time of Yamen Decryption From PC2

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	1.874	1.234	1.265	1.46
File 2	2	2.332	2.227	2.093	2.22
File 3	3	3.387	3.153	3.044	3.19
File 4	4	3.512	3.605	3.544	3.55
File 5	5	4.793	4.652	4.792	4.75
File 6	6	5.259	5.572	5.696	5.51
File 7	7	6.258	6.678	5.789	6.24
File 8	8	6.211	7.709	7.459	7.13
File 9	9	8.683	8.223	7.069	7.99
File 10	10	8.507	9.284	7.99	8.59

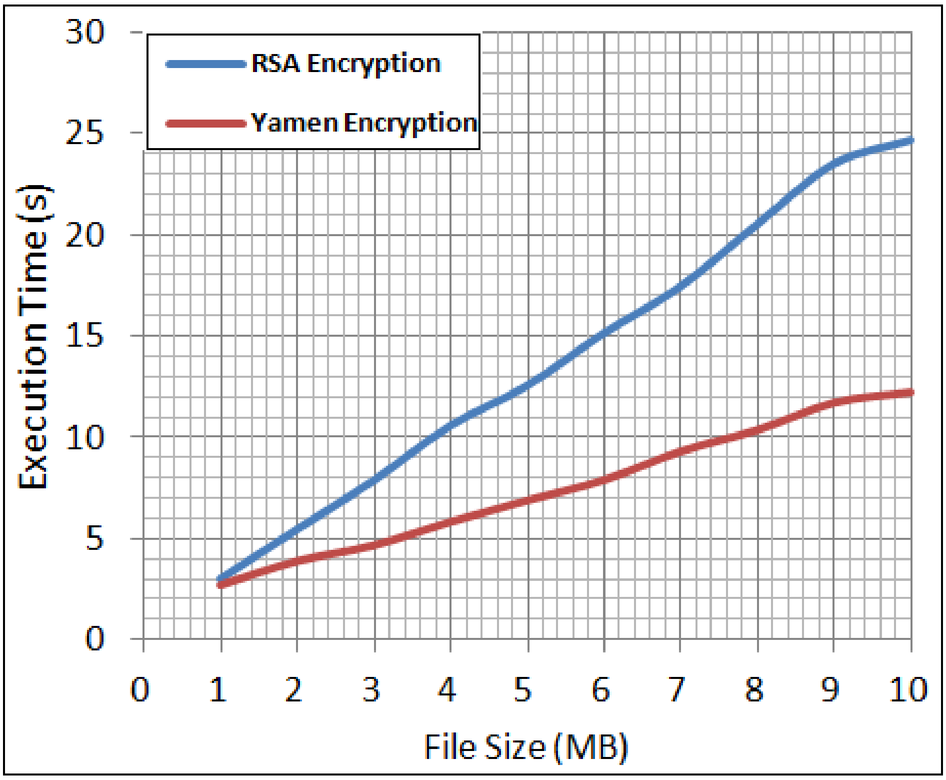


FIGURE B.3: Execution Time for RSA and Yamen Encryption Process (PC2)

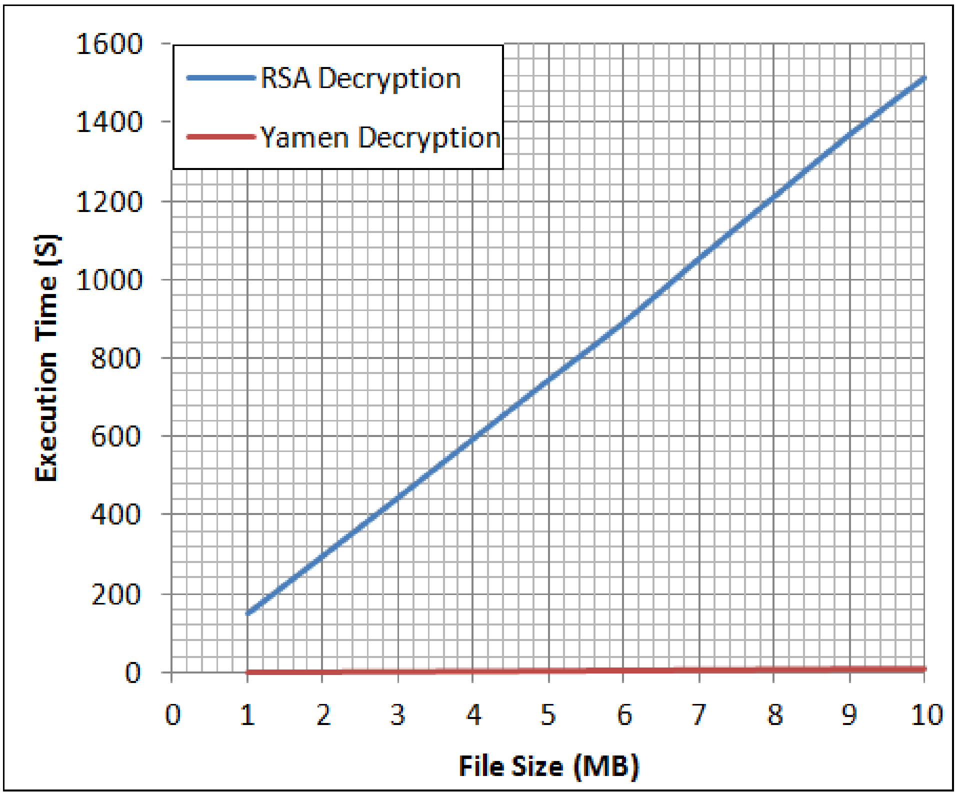


FIGURE B.4: Execution Time for RSA and Yamen Decryption Process (PC2)

Table B.10 Execution Time of RSA Encryption From PC3

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	3.005	3.176	3.143	3.11
File 2	2	6.021	5.717	5.727	5.82
File 3	3	8.421	8.421	8.373	8.41
File 4	4	10.78	11.004	11.06	10.95
File 5	5	13.49	13.742	13.855	13.7
File 6	6	16.467	16.177	16.21	16.28
File 7	7	18.999	18.942	18.668	18.87
File 8	8	21.43	21.574	21.256	21.42
File 9	9	24.509	24.36	24.266	24.38
File10	10	26.477	26.815	26.73	26.67

Table B.11 Execution Time of RSA Decryption From PC3

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	165.04	164.922	165.472	165.14
File 2	2	331.522	332.941	335.182	333.22
File 3	3	498.477	496.855	495.879	497.07
File 4	4	670.622	672.98	685.55	676.38
File 5	5	841.374	842.529	819.171	834.36
File 6	6	1012.094	1013.664	995.613	1007.12
File 7	7	1153.341	1145.127	1156.84	1151.77
File 8	8	1342.598	1341.429	1324.035	1336.02
File 9	9	1478.807	1474.442	1472.8	1475.35
File 10	10	1687.022	1626.527	1622.741	1645.43

Table B.12 Execution Time of Yamen Encryption From PC3

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	3.653	2.977	3.161	3.26
File 2	2	4.214	4.382	4.229	4.28
File 3	3	5.119	5.235	5.531	5.3
File 4	4	6.747	6.563	6.391	6.57
File 5	5	7.706	7.63	8.083	7.81
File 6	6	8.664	8.889	8.956	8.84
File 7	7	10	10.002	9.659	9.89
File 8	8	10.471	10.361	10.993	10.61
File 9	9	12.636	11.438	12.583	12.22
File 10	10	12.92	13.278	13.301	13.17

Table B.13 Execution Time of Yamen Decryption From PC3

File Information		Execution Time/Second			
File Name	File Size/MB	First Reading	Second Reading	Third Reading	Average
File 1	1	2.587	2.232	2.42	2.41
File 2	2	2.967	2.296	2.592	2.62
File 3	3	3.919	3.899	3.95	3.92
File 4	4	4.902	4.136	4.308	4.45
File 5	5	5.002	4.909	4.76	4.89
File 6	6	5.665	5.915	5.961	5.85
File 7	7	6.773	6.648	6.81	6.74
File 8	8	7.515	7.492	7.304	7.44
File 9	9	7.865	8.323	8.302	8.16
File 10	10	9.108	8.676	8.901	8.9

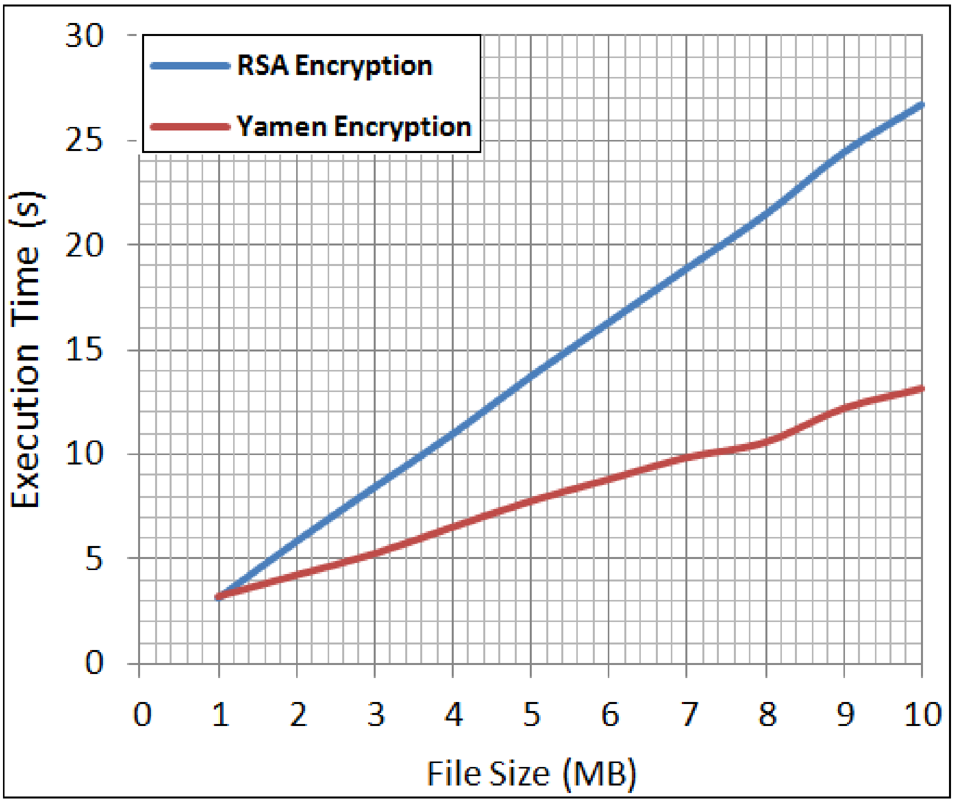


Figure B.5: Execution Time for RSA and Yamen Encryption Process (PC3)

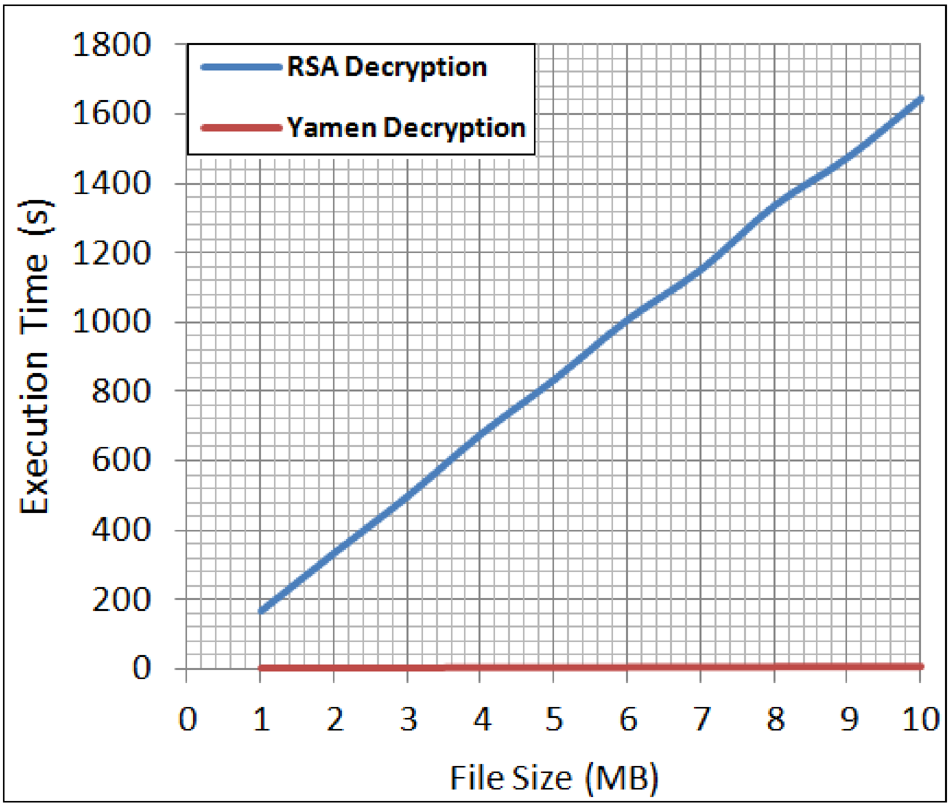


Figure B.6: Execution Time for RSA and Yamen Decryption Process (PC3)

Our testing results from the three experiments tell us, *Yamen* cryptosystem is faster than basic RSA by 45% in encryption process and 99% in decryption process.

Bibliography

[1] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[2] Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0, August 2011. RFC6101.

[3] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2, August 2008. RFC5246.

[4] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) transport layer protocol, January 2006. RFC4253.

[5] Asecurity dynamics company RSA Data Security. Security protocols overview an RSA data security brief. ftp://ftp.rsa.com/pub/pdfs/protocols.pdf. Accessed March 19,2014.

[6] EMC Corporation. Rsa authentication agents for microsoft window. http://brazil.emc.com/collateral/data-sheet/h9060-agnwin-ds.pdf, . Accessed December 2, 2014.

[7] Simson Garfinkel, Gene Spafford, and Alan Schwartz. *Practical UNIX and Internet security.* " O'Reilly Media, Inc.", 2003.

[8] Oracle Corporation. Oracle security server concepts. https://docs.oracle.com/cd/A58617_01/network.804/a54088/conc1.htm, . Accessed December 2, 2014.

[9] Oracle Corporation. Enterprise encryption function descriptions. http://dev.mysql.com/doc/refman/5.6/en/enterprise-encryption-functions.html, . Accessed December 2, 2014.

[10] Taher Elgamal. Electronic commerce using a secure courier system, September 23 1997. US Patent 5,671,279.

[11] Vipul Gupta and Sumit Gupta. Securing the wireless internet. *Communications Magazine, IEEE*, 39(12):68–74, 2001.

[12] Malek Jakob Kakish. Enhancing the security of the RSA cryptosystem. *International Journal of Research & Reviews in Applied Sciences*, 8(2), 2011.

[13] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[14] Gustavus J Simmons. A weak privacy protocol using the RSA crypto algorithm. *Cryptologia*, 7(2):180–182, 1983.

[15] John M DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8(3):253–259, 1984.

[16] Song Y Yan. *Cryptanalytic attacks on RSA*. Springer, 2007.

[17] Hagai Bar-El. Introduction to side channel attacks. *Discretix Technologies Ltd*, 43, 2003.

[18] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology–CRYPTO'96*, pages 104–113. Springer, 1996.

[19] Chia-Long Wu, Der-Chyuan Lou, Jui-Chang Lai, and Te-Jen Chang. Fast parallel exponentiation algorithm for RSA public-key cryptosystem. *Informatica*, 17(3): 445–462, 2006.

[20] Swapna B Sasi, Dila Dixon, and Jesmy Wilson. A general comparison of symmetric and asymmetric cryptosystems for wsns and an overview of location based encryption technique for improving security. *IOSR Journal of Engineering*, 4(3):1, 2014.

[21] William Freeman and Ethan Miller. An experimental analysis of cryptographic overhead in performance-critical systems. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999. Proceedings. 7th International Symposium on*, pages 348–357. IEEE, 1999.

[22] Whitfield Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

[23] Stefan Katzenbeisser. *Recent advances in RSA cryptography*, volume 3. Springer, 2001.

[24] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology–EUROCRYPT'98*, pages 308–318. Springer, 1998.

[25] wikipedia. Bitwise operation. http://en.wikipedia.org/wiki/Bitwise_operation, September 2014. Accessed October 7, 2014.

[26] Matt Bishop. What is computer security? *Security & Privacy, IEEE*, 1(1):67–69, 2003.

[27] Gurtaptish Kaur and Sheenam Malhotra. A hybrid approach for data hiding using cryptography schemes. *International Journal of Computer Trends and Technology (IJCTT)*, 4:1–7, August 2013. URL http://www.ijcttjournal.org/Volume4/issue-8/IJCTT-V4I8P196.pdf.

[28] Nicholas G. McDonald. Past, present, and future methods of cryptography and data encryption. URL http://www.eng.utah.edu/~nmcdonal/Tutorials/EncryptionResearchReview.pdf.

[29] Sushma Pradhan and Birendra Kumar Sharma. A new design to improve the security aspects of RSA cryptosystem. *International Journal of Computer Science and Business Informatics*, 3:1–11, 2013. URL http://www.ijcsbi.org/index.php/ijcsbi/article/view/82.

[30] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 6th edition, 2013. ISBN 0133354695, 9780133354690.

[31] Kenneth H Rosen. *Elementary number theory and its applications*. Reading, Mass., 1993.

[32] William Stallings. *Network security essentials applications and standards, 5th*. Pearson Education, 2013.

[33] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[34] RSA Labortories. What is public key cryptography. http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/what-is-public-key-cryptography.htm. Accessed April 19,2014.

[35] Eric Rescorla. Diffie-Hellman key agreement method. 1999. RFC 2631.

[36] Michael O Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, 1979.

[37] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[38] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177): 203–209, Jan. 1987. American Mathematical Society.

[39] Victor S Miller. Use of elliptic curves in cryptography. In *Lecture Notes in Computer Sciences; on Advances in cryptology—CRYPTO 85*, volume 218, pages 417–426. Springer-Verlag New York, Inc., 1986.

[40] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology–CRYPTO99*, pages 537–554. Springer, 1999.

[41] L Harn, W-J Hsin, and M Mehta. Authenticated Diffie–Hellman key agreement protocol using a single cryptographic assumption. *IEE Proceedings-Communications*, 152(4):404–410, 2005.

[42] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *Advances in Cryptology–ASIACRYPT 2001*, pages 566–582. Springer, 2001.

[43] R Stinson Douglas. *Cryptography theory and practice.* pub-CRC, 1995.

[44] Sonal Sharma, Prashant Sharma, and Ravi Shankar Dhakar. Rsa algorithm using modified subset sum cryptosystem. In *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, pages 457–461. IEEE, 2011.

[45] Arvinderpal S Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 324–328. IEEE, 2005.

[46] Gururaja HS, M Seetha, Anjan K Koundinya, and Prashanth CA. Comparative study and performance analysis of encryption in RSA, ECC and Goldwasser-Micali cryptosystems. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, 3(1):111–118, January 2014.

[47] Aravinthan Jegatheesan. Effectiveness of public key algorithms against quantum computers. April 2005. SE 4C03 Winter 2005.

[48] Vibhor Mehrotra and KC Joshi. A method for breaking RSA security. *International Journal*, 2(9), 2012.

[49] Arpit Kumar Srivastava and Abhinav Mathur. The Rabin cryptosystem & analysis in measure of chinese reminder theorem. *International Journal of Scientific and Research Publications*, page 493.

[50] Cunsheng Ding. *Chinese remainder theorem*. World Scientific, 1996.

[51] Neal R Wagner. The laws of cryptography with java code. *Available online at Neal Wagners home page*, 2003.

[52] wikipedia. Rabin cryptosystem. http://en.wikipedia.org/wiki/Rabin_cryptosystem. Accessed May 9, 2014.

[53] Man Young Rhee. *Cryptography and secure communications*. McGraw-Hill, Inc., 1993.

[54] Steven D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, Cambridge, New York, 2012. ISBN 978-1-10-701392-6. URL http://opac.inria.fr/record=b1133733.

[55] Naiara Escudero Sanchez. The Rabin cryptosystem. University of Paderborn. URL http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Bloemer/lehre/2011/ss/seminar/Essays/Naiara_Sanchez_-_Rabin_Cryptosystem.pdf. Accessed May 9, 2014.

[56] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology–EUROCRYPT'98*, pages 59–71. Springer, 1998.

[57] Larry Hardesty. Beefing up public-key encryption. http://newsoffice.mit.edu/2013/beefing-up-public-key-encryption-0215, February 2013. Accessed April 19, 2014.

[58] Andreas de Vries. The ray attack, an inefficient trial to break RSA cryptosystems. http://arxiv.org/abs/cs/0307029, July 2003. Accessed April 19, 2014.

[59] Imad Khaled Salah, Abdullah Darwish, and Saleh Oqeili. Mathematical attacks on RSA cryptosystem. *Journal of Computer science*, 2(8), 2006.

[60] Chey Cobb. *Cryptography for dummies*. John Wiley & Sons, 2004.

[61] Wing H Wong. Timing attacks on RSA: revealing your secrets through the fourth dimension. *Crossroads*, 11(3):5–5, 2005.

[62] Dan Boneh et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[63] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, DTIC Document, 2001.

[64] Roll dice online. http://www.roll-dice-online.com/. Accessed May 3, 2014.

[65] The hungry mouse. http://www.kidsgen.com/short_stories/the_hungry_mouse.htm. Accessed May 2, 2014.

[66] Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier. Universal padding schemes for RSA. In *Advances in Cryptology–CRYPTO 2002*, pages 226–241. Springer, 2002.

[67] David Pointcheval. New public key cryptosystems based on the dependent-RSA problems. In *Advances in Cryptology–EUROCRYPT'99*, pages 239–254. Springer, 1999.

[68] Pallavi Jindal and Vikas Gupta. Article: Modifications in RSA through positive justification of random components. *International Journal of Computer Applications*, 75(12):12–16, August 2013. Full text available.

[69] Burt Kaliski. The mathematics of the RSA public-key cryptosystem. *RSA Laboratories*, 2006.

[70] T Shahana. An enhanced security technique for steganography using DCT and RSA. *International Journal*, 3(7), 2013.

[71] Gajendra Singh Chandel and Pragna Patel. A review: Image encryption with RSA and RGB randomized histograms. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 3, May 2014.

[72] Richard E Blahut. *Cryptography and Secure Communication*. Cambridge University Press, 2014.

[73] Indiver Purohit and Raj Kumar Somani. Hybrid cryptography algorithm based on prime factorization. *International Journal of Recent Development in Engineering and Technology*, 2, February 2014.

[74] Gustavus J Simmons. *Contemporary cryptology: the science of information integrity*. IEEE press, 1994.

[75] Claude E Shannon. Communication theory of secrecy systems*. *Bell system technical journal*, 28(4):656–715, 1949.

[76] David Salomon. *A concise introduction to data compression*. Springer, 2007.

[77] Wikipedia. Huffman coding. http://en.wikipedia.org/wiki/Huffman_coding/. Accessed April 18, 2014.

[78] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*, volume 63. SIAM, 1992.

[79] David W Gillman, Mojdeh Mohtashemi, and Ronald L Rivest. On breaking a huffman code. *IEEE Transactions on Information theory*, 42(3):972–976, 1996.

[80] Guicheng Shen, Bingwu Liu, Xuefeng Zheng, et al. Research on fast implementation of RSA with java. *Nanchang, PR China*, pages 186–189, 2009.

[81] Dan Boneh and Hovav Shacham. Fast variants of RSA. *CryptoBytes*, 5(1):1–9, 2002.

[82] microsoft. Asymmetric keys. http://msdn.microsoft.com/en-us/library/windows/desktop/aa387460(v=vs.85).aspx. Accessed october 7,2014.

[83] Filipe da Costa Boucinha. A survey of cryptanalytic attacks on RSA. Master's thesis, Instituto Superior Técnico, 2011.

[84] Hazarathaiah Malepati. *Digital media processing: DSP algorithms using C*. Newnes, 2010.

[85] IA Dhotre VS Bagad. *Cryptography And Network Security*. Technical Publications, 2008.

[86] D Caliskan. An application of RSA in data transfer. In *Application of Information and Communication Technologies (AICT), 2011 5th International Conference on*, pages 1–4. IEEE, 2011.

[87] Hilarie Orman and Paul Hoffman. Determining strengths for public keys used for exchanging symmetric keys, April 2004. RFC3766.

[88] Shannons theory of secrecy. http://www.eit.lth.se/fileadmin/eit/courses/edi051/lecture_notes/LN3.pdf. Accessed May 2, 2014.

[89] Colin Boyd. Enhancing secrecy by data compression: theoretical and practical aspects. In *Advances in Cryptology–EUROCRYPT'91*, pages 266–280. Springer, 1991.