COMPUTATIONAL ANALYSIS OF RSA BASED ATTACKS

Mustafa KOCAKULAK

Master of Science Thesis

Department of Mechatronics Engineering

Assoc. Prof. Dr. Turgay TEMEL

2015

T.R

BURSA TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTATIONAL ANALYSIS OF RSA BASED ATTACKS

MASTER OF SCIENCE THESIS

Mustafa KOCAKULAK

Department of Mechatronics Engineering

BURSA

June 2015

MASTER OF SCIENCE THESIS EXAMINATION RESULT FORM

The thesis entitled "COMPUTATIONAL ANALYSIS OF RSA BASED ATTACKS" completed by "Mustafa KOCAKULAK" under supervision of "Assoc. Prof. Turgay TEMEL" has been reviewed in terms of scope and quality and approved as a thesis for the degree of Master of Science.

Jury Members

Assoc. Prof. Turgay TEMEL ……..

(Bursa Technical University, Department of Mechatronics Engineering)

Prof. Nurettin ACIR ……..

(Bursa Technical University, Department of Electrical-Electronic Engineering)

Asst. Prof. İsmail BÜTÜN ……..

(Bursa Technical University, Department of Mechatronics Engineering)

Date of Examination: 11/06/2015

Director of Graduate School of Natural and Applied Sciences

Prof. Nurettin ACIR …/06/2015 ……..

DECLARATION ON PLAGIARISM

I hereby declare that this thesis is the result of my own independent scholarly work, and that in all cases material from the work of others is acknowledged, and quotations and paraphrases are clearly indicated. No material other than listed has been used. I am aware of what constitutes an act of plagiarism and understand that my thesis will be rejected if found to contain any instance of plagiarism.

Student Name and Surname: Mustafa KOCAKULAK

Signature:

ACKNOWLEDGEMENTS

This thesis has been conducted, under the supervision of Assoc. Prof. Turgay TEMEL, for the master program: Mechatronics Engineering.

The main reason for choosing the topic of “Computational Analysis of RSA Based Attacks” is because RSA have been important and indisputable for cryptology and web since it was proposed in 1977. Although the RSA public-key algorithm was proposed and developed in 1977, it has been widely used in a variety of computer security applications. It is still contemporary and it has wide usage. But its widely usage does not make us evaluate the RSA algorithm as a perfect algorithm for public-key cryptography. Throughout the research, I have learned more about the public-key algorithms, the RSA algorithm, attacks against RSA’s security and RSA applications in Java.

The completion of that master thesis would not have been feasible without the help of my supervisor. For that reason, I want to express my deepest gratitude to my supervisor Assoc. Prof. Turgay TEMEL who has been very helpful and patient in guiding me throughout the research process. His guidance and experience played an important role in the improvement of my thesis. I would like to thank my thesis committee members for all of their guidance through this process.

Lastly, I would like to thank my parents Fatma KOCAKULAK and İbrahim KOCAKULAK and also my sisters for their support and deepest love. Their encouragement and love enabled me to overcome difficulties that I faced.

Mustafa KOCAKULAK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ACRONYMS

Symbols	Definition
C	cipher-text
d	private exponent of RSA
e	public exponent of RSA
gcd	greatest common divisor
K	the encryption or the decryption key in terminology
M	message
N	component of private-key or public-key of RSA
$\mathbb{N}$	set of natural numbers
p	one of large primes of RSA
P	plain-text
$Phi(N)$	Euler's Phi function
q	one of large primes of RSA
r	remainder
x	plain-text in terminology
y	cipher-text in terminology
$\mathbb{Z}$	set of integers
$\mathbb{Z}_n$	set of positive integers from 0 to n-1
$\Phi(N)$	Euler's Totient function

Acronyms	Definition
API	American Standard Code for Information Interchange
ASCII	American Standard Code for Information Interchange
JCA	Java Cryptography Architecture
JCE	Java Cryptographic Extension
RNG	Random Number Generator
RSA	Rivest-Shamir-Adleman Algorithm

ÖZET

RSA TABANLI ATAKLARIN HESAPLAMALI ANALİZİ

Mustafa KOCAKULAK

Bursa Teknik Üniversitesi

Fen Bilimleri Enstitüsü

Mekatronik Mühendisliği Ana Bilim Dalı

Yüksek Lisans Tezi

Doç. Dr. Turgay TEMEL

Haziran 2015, 45 Sayfa

İki veya daha fazla asal sayının çarpımından meydana gelen büyük bir sayının, asal bileşenlerine ayrıştırılabilmesinin zorluğu esasına dayanan RSA sistemi, sağlamış olduğu güvenlik seviyesi ve anahtar paylaşımında getirdiği yeniliklerle kriptoloji alanında tartışmasız bir öneme sahiptir. Mevcut algoritmanın beraberinde getirdiği uzun anahtar boyutları, gerektirdiği geniş hafıza alanı ve anahtar paylaşımının dayandığı 'asal bileşenlere ayrıştırmanın zorluğu' esasının güvenlik açısından aşılabilir olması, bu alanda mevcut RSA algoritmasında değişiklikler yapmayı ya da RSA'yı maksimum güvenlikle korumayı sağlayan önlemleri uygulama esnasında almayı gerekli kılmaktadır. Bu çalışmada RSA algoritması birçok yönüyle ele alınacak, uygulanan bazı kriptanaliz yöntemlerine karşı RSA'ya maksimum güvenlik sağlayacak tedbirler gösterilecektir.

Anahtar Sözcükler: RSA, Çarpanlara Ayırma, Anahtar Boyutu, Asimetrik Anahtar

ABSTRACT

COMPUTATIONAL ANALYSIS OF RSA BASED ATTACKS

Mustafa KOCAKULAK

Bursa Technical University

Graduate School of Natural and Applied Science

Department of Mechatronics Engineering Program

Master of Science Thesis

Assoc. Prof. Dr. Turgay TEMEL

June 2015, 45 Pages

The RSA Algorithm has an indisputable importance in cryptology. RSA's security depends on the difficulty of factoring big composite number. This number is the multiplication of two or more prime numbers and its factorization is nearly infeasible. Since the existing RSA Algorithm needs long key sizes, requires big memory spaces and also has the deficiency depending on a feasible but long factorization principle, it seems necessary to make some changes or taking some necessary precautions during the implementation of RSA in order to provide maximum security level. In this thesis, RSA algorithm will be examined and evaluated in detail. Moreover, some security precautions, that support RSA against some applied cryptanalysis methods, will be indicated.

Key Words: RSA, Factorization, Key Size, Asymmetric-Key

1. INTRODUCTION

Cryptology is the art of secret writing [1]. It is a set of cryptography and cryptanalysis. Cryptography is used to write hidden messages and cryptanalysis is used to analyze hidden messages and recover the original messages. More specifically, cryptology is a way providing security and secrecy to any communication between three main characters in any unsecure channel. These characters are sender, receiver, and eavesdropper. In this unsecure (open to possible attacks) communication channel, sender and receiver want to exchange information securely and secretly. Up to this point, there is no need for cryptology but when eavesdropper starts to listen to this channel to reach the secret information between sender and receiver, cryptology's importance for communication with the intended receivers is realized.

For thousands of years, people always want to send messages which can only be read by the intended receivers. Until 1980s, the message between the sender and the receiver was carried by using symmetric-key cryptosystems [2]. With the increase in the usage of computers and mobile devices, the usage area of cryptology has enlarged dramatically. The increase in computer usage, have made the interconnections via networks between computers very important. When people started to have their private computers and use them to send and to receive information, their awareness and needs to protect their data and resources have increased profoundly. At this moment, the modern cryptology has arisen. After 1980s, with Diffie-Hellman Algorithm and the Rivest-Shamir-Adleman Algorithm, RSA, people have met asymmetric-key cryptosystems [3].

RSA Algorithm has an indisputable importance in cryptology. It is one of the most widely known and used asymmetric-key cryptosystem [4]. Since the existing RSA Algorithm needs long key sizes and has the deficiency depending on a feasible but long factorization principle, it seems necessary to make some changes or taking some necessary precautions during the implementation of RSA to provide maximum security level. The objectives of this thesis are emphasizing the mathematical logic behind RSA Algorithm, implementation of RSA Algorithm up to on 2048-bit with and without padding, realizing the effect of bit length on the success of given some well-known factorization algorithms.

2. MATHEMATICAL CRYPTOGRAPHY

In order to understand the mathematical ideas underlying symmetric-key cryptography and asymmetric-key cryptography, it is necessary to explain the number theory and emphasize the importance of algebra on algorithms because most of the public-key algorithms are based on modular arithmetic [5].

2.1. Number Theory and Algebra

The set of integers, denoted by $\mathbb{Z}$, contains all numbers with no fraction from negative infinity to positive infinity. These integers are used in binary operations in cryptography. The binary operations used in cryptography are addition, multiplication and subtraction. Each of these operations takes two inputs and creates one output. Since division gets two inputs (dividend and divisor) and creates two outputs (remainder and quotient) instead of one output, division operation is not categorized as binary operation. Although division algorithm is not classified as binary operation, it is used in cryptology and it has a vital role in asymmetric-key cryptosystems.

2.1.1. Divisibility and Greatest Common Divisor

Division algorithm implies that when an integer is divided by a positive integer, there is a quotient and a remainder. Suppose that a is an integer and d is a positive integer, then there are unique integers, q and r, where $0 \leq r < d$, $a = d * q + r$. a is called the dividend, d is called the divisor, q is called the quotient and r is called the remainder. Now suppose that x and y are integers and n is a positive integer, then x is congruent to y modulo n if $n|(x - y)$. The notation $x \equiv y \ (mod\ n)$ implies that x is congruent to y modulo n. Instead of equality operator, $=$, congruence operator, $\equiv$, is always used to indicate congruence. The difference between these two operators is that while equality operator is one-to-one, the congruence operator is many-to-one.

Division operator's usage in cryptography is various and not limited. For example, primality testing and finding greatest common divisor processes require division operator. A positive integer greater than 1 is prime number if its only divisors are 1 and itself. The greatest common divisor of two positive integers, a and b, is indicated by $\gcd(a, b)$ and it equals to the largest integer that can divide both integers.

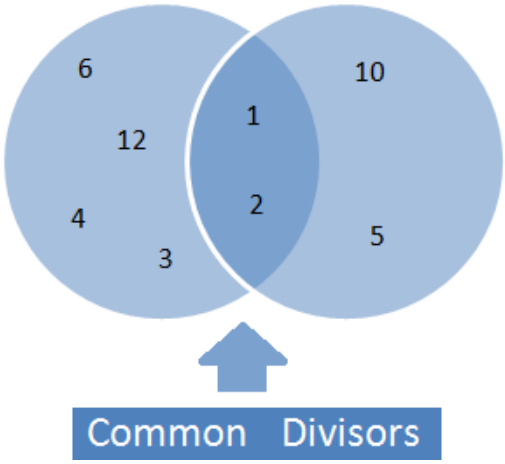

Figure 2.1 The common divisors of 12 and 10

As seen in Figure 2.1, the common divisors of 12 and 10 are 1 and 2. The greatest common divisor for 12 and 10, gcd (12,10) is 2 since $2 > 1$.

The integers a and b are relatively prime which is known also as co-prime if and only if $\gcd(a, b) = 1$. Owing to finding out that whether two numbers have common divisor or not, some of asymmetric-key cryptography algorithms can be applied correctly such as RSA Algorithm.

2.1.2. Primality and Testing

A Primality Test is an algorithm that is used to find out whether or not given number is prime ($p \in \mathbb{N}$ and $p \geq 2$). An integer n is a prime if and only if n has two divisors 1 and itself, n. There are infinite numbers of prime numbers in a set $\mathbb{P}$ where $\mathbb{P} = \{2,3,5,7,11,13, \dots\}$. In order to find primes, the wrong way is generate random numbers and then try to factor them. It can be too time consuming. The correct way can be generating random numbers and test their primality. If any number passes some primality tests, then it is likely to be prime number. But if it fails one of primality test, it is definitely composite number [6]. Trial Division Test, Fermat's Test and Miller Rabin Test are widely known and used testing methods [7].

2.1.3. Euler's Phi Function

Euler Function is known as Euler's Phi Function or Euler's Totient Function [8]. This function is showed by $\Phi(n)$ and defined as the number of positive integers less

than n and relatively prime to n. The set of integers contains all integer numbers with no fraction from negative infinity t positive infinity is denoted by $\mathbb{Z}$, where $\mathbb{Z}_n$ is defined as the set of integers $\{0,1,2,\dots,n-1\}$, Euler Function makes us know that how many numbers in this set are relatively prime to n.

Let $n = 5$, then the associated set is $\mathbb{Z}_5 = \{0,1,2,3,4\}$. Firstly, $\gcd(i,n)$ must be calculated for every integer i from 0 to $n-1$ must be calculated.

$$\gcd(0,5) = 5$$

$$\gcd(1,5) = 1 \text{ (1 and 5 are co-prime)}$$

$$\gcd(2,5) = 1 \text{ (2 and 5 are co-prime)}$$

$$\gcd(3,5) = 1 \text{ (2 and 5 are co-prime)}$$

$$\gcd(4,5) = 1 \text{ (2 and 5 are co-prime)}$$

Therefore, since $n = 5$, $\mathbb{Z}_5 = \{0,1,2,3,4\}$ and for $i = \{1,2,3,4\}, \gcd(i,n) = 1$, $\Phi(5) = 4$. Euler Theorem may be used easily to reduce large powers modulo n. Since $\Phi(n)$ equals to the number of positive integers less than n that are co-prime with n, it states that if n is a positive integer and a co-prime with n, then

$$a^{\Phi(n)} \equiv 1 \pmod{n} \tag{2.1}$$

For Euler's Totient Function, $\Phi(n)$, there are also some other useful cases that makes its calculation faster:

- For all $\Phi(n)$, if n is prime then,
$$\Phi(n) = n - 1 \tag{2.2}$$
- If $\gcd(n,m) = 1$ then,
$$\Phi(m * n) = \Phi(m) * \Phi(n) \tag{2.3}$$
- If p is prime and $n \neq 1$, then
$$\Phi(p^n) = p^n - p^{n-1} \tag{2.4}$$

In Figure 2.2, the red points indicate the value of Euler Phi Function for the varying values from 0 to 30. The nearest red points to blue points indicate the validity of Equation 2.2.

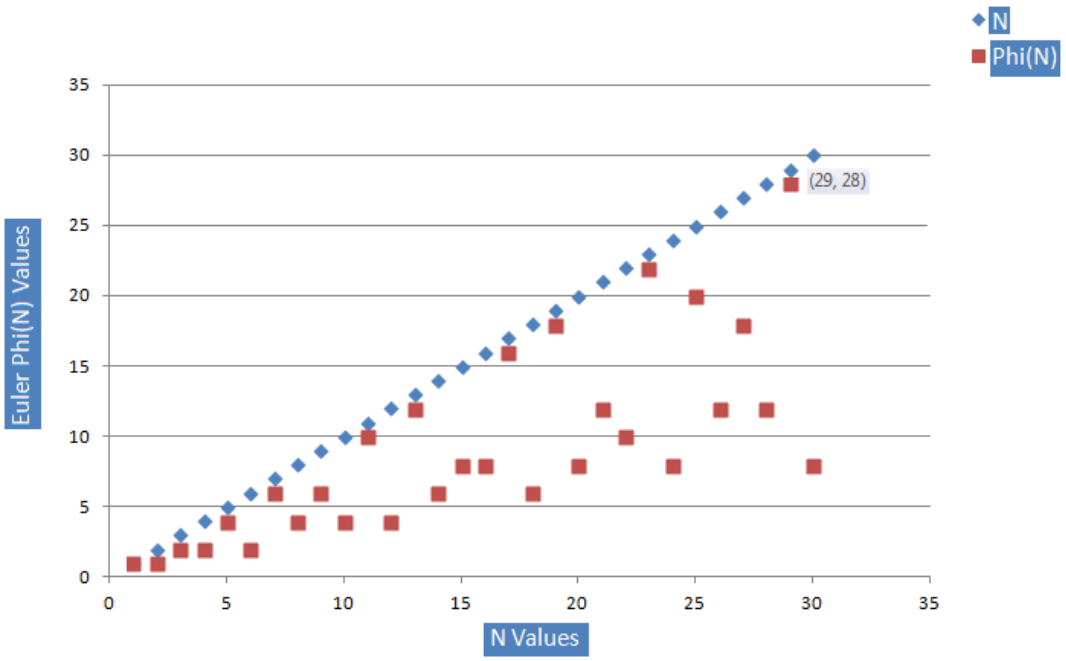

Figure 2.2 The distribution of Euler Phi Function from 0 to 30.

Euler's Phi Function is very crucial for asymmetric-key cryptography, especially for RSA [9]. In order to calculate Euler's Phi Function quickly, the factorization of n must be known. It is one of the most important points for RSA because if the factorization of a number n is known then, it is possible to compute Euler's Phi Function and decrypt the cipher-text [10]. If the factorization is not known, then Phi Function cannot be computed hence the cipher-text cannot be decrypted.

2.1.4. Fermat's Little Theorem

Fermat's Little Theorem is also known that this theorem is a special case of Euler's Theorem. Fermat's Theorem states that if a is an integer and p is a prime, then:

$$a^{p-1} \equiv 1 \ (mod\ p) \tag{2.5}$$

Fermat's Little Theorem is useful for primality testing and in many aspects of asymmetric-key cryptography [11]. Fermat's Little Theorem is useful in computing the remainders modulo p large powers of integers. To visualize the importance of Fermat's Little Theorem for computing the remainders modulo p large powers of integers, let's solve $7^{221}\ modulo\ 11$.

By using Fermat's Little Theorem, we know that $7^{11-1} = 7^{10} \equiv 1\ (mod\ 11)$. Since $(7^{10})^k \equiv 1(mod\ 11)$ for every positive integer, k, $7^{221} \equiv (7^{10})^{22} * 7^1 \equiv 1^{22} * 7^1(mod\ 11) \equiv 7(mod\ 11) \equiv 7$. As seen by this example, Fermat's Little Theorem is useful in doing exponentiations modulo an integer is needed. Alternative form of Fermat's Little Theorem is

$$a^p \equiv a\ (mod\ p) \tag{2.6}$$

where for every integer a, p must be prime. The difference between two forms of Fermat's Theorem is that $a^p \equiv a(mod\ p)$ form does not require that a relatively prime to p while $a^{p-1} \equiv 1\ (mod\ p)$ requires.

2.2. Cryptography and Cryptanalysis

Cryptology is a set of cryptography and cryptanalysis as seen in Figure 2.3.

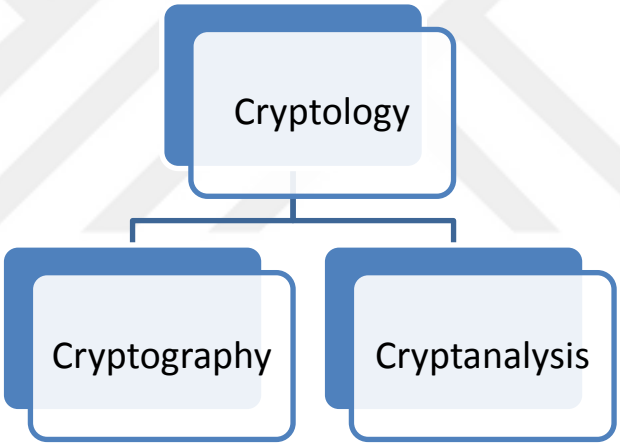

Figure 2.3 Classification of cryptology

In cryptology, Alice, Bob and Eve are three characters used in any information exchange scenario. Alice is the person who needs to send secure data. Bob is the recipient of the data. Eve is the person who somehow disturbs the communication between Alice and Bob by intercepting messages. The original message before being encrypted is known as plain-text.

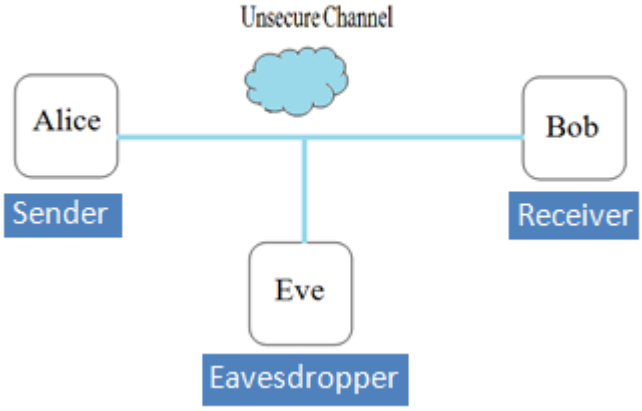

Figure 2.4 Communication over an unsecure channel

The original message after being encrypted is known as cipher-text. Encryption is known as a process of changing original text into cipher-text. Decryption is known as a process of changing cipher-text into original text. Algorithms used in decryption or encryption are known as cipher. A key is defined as a number that the cipher (encryption or decryption) operates on [12]. In every encryption process while encrypting any message, the following materials are necessary: an encryption algorithm, a plain-text and an encryption key. In every decryption process while decrypting any message, a decryption algorithm, a cipher-text and a decryption key are required. In cryptography terminology, the plain-text is denoted by x, the cipher-text is denoted by y, the encryption key and decryption key are denoted by K [13].

2.2.1. Cryptography and Keys

In cryptography, there are three types of keys that we deal with: the secret-key, the public-key and the private-key. The secret-key is the shared-key used in symmetric-key cryptography. The other keys, the private-key and the public-key are used in asymmetric-key cryptography.

2.2.2. Symmetric-Key Cryptography

Symmetric-key cryptography is sometimes referred as secret-key cryptography. It is more historical form of cryptography in which a single key is used to decrypt and encrypt a message. In symmetric-key cryptography, the same key is used by both

parties that is why it called symmetric-key. The logic behind the symmetric-key cryptography relies on the shared key agreement between the sender and the receiver. The sender and the receiver agree on a key and use it for encryption and decryption of the message. Nobody else other than them knows this shared key. Shortly, the key is shared and single. As seen in Figure 2.5, *K* is used as a single key for encryption and decryption by both sides.

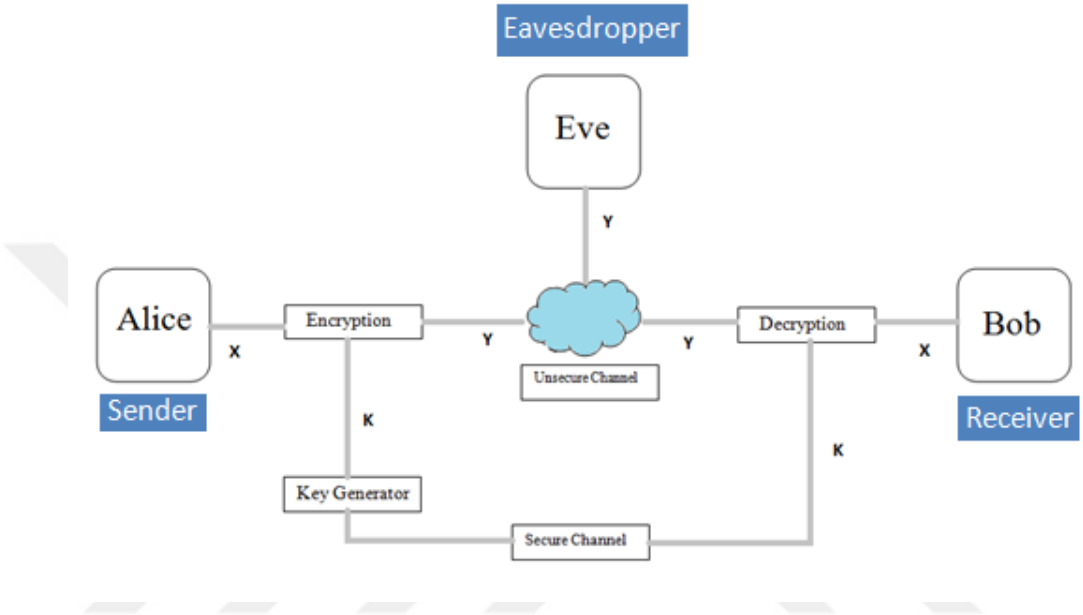

Figure 2.5 Symmetric-key cryptosystem

2.2.3. Asymmetric-Key Cryptography

In asymmetric-key cryptography, there are two keys: a private-key and a public-key. The private-key is kept by the receiver. The public-key is announced to the public. In asymmetric-key encryption / decryption, the public-key that is used for encryption is different from the private-key that is used for decryption. That is why it is known as asymmetric-key cryptography. The public-key is available to the public; the private-key is available only to an individual. The private-key is always linked mathematically to the public-key. Therefore, it is always possible to attack a public-key system by deriving the private-key from the public-key.

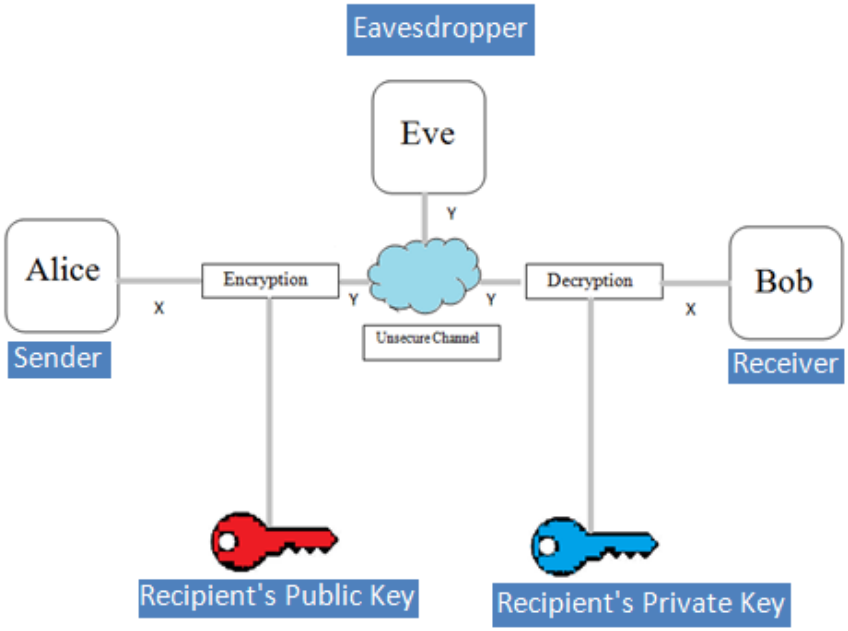

Figure 2.6 Asymmetric-key cryptosystem

2.2.4. Comparison of Symmetric-Key and Asymmetric-Key Cryptography

The main advantage of symmetric-key cryptography is its speed [14]. It is faster than asymmetric-key cryptography. The main disadvantage of symmetric-key cryptosystems is key management and distribution [15]. Since the number of the required keys increases with the number of network population, key management and distribution becomes problematic issue for symmetric-key algorithms. Key management is necessary to use keys securely. For asymmetric-key, there is no need for exchanging keys, thus there is not any key distribution problem. The main disadvantage is its speed. Since long key sizes needed, the usage of asymmetric-key in any system makes process slower [16]. For example, for RSA Algorithm, the main disadvantage is that it requires long key sizes in order to provide good security. Therefore, RSA is not suitable for encrypting the large texts and it is usually used for key exchange.

2.2.5. Key Management for Symmetric-Key Cryptography

Key distribution is the function that delivers a key to a sender and a receiver to make them able to exchange encrypted data securely. The security of distribution of keys

depends on some protocols and mechanisms. For symmetric encryption, a sender and a receiver must share the same key and this key must be protected from third parties and also distributed keys must be changed and destroyed in frequent periods. The frequent key changes and key destruction have crucial importance to decrease the amount of data obtained by attacker. For symmetric encryption, the number of keys depends on the number of people in a communication network. The number of required keys is

$$\frac{N*(N-1)}{2} \tag{2.7}$$

where N equals to the number of people in the network [17]. As shown in Figure 2.7, if there are 2 people in the network, the required key is $\frac{N*(N-1)}{2} = \frac{2*1}{2} = 1$. When there are 3 people in the network, $\frac{N*(N-1)}{2} = \frac{3*2}{2} = 3$ and when 4 people in the network, $\frac{N*(N-1)}{2} = \frac{4*3}{2} = 6$ keys are required. When the network population increases, as it is seen, the number of required keys increases and key management becomes more difficult.

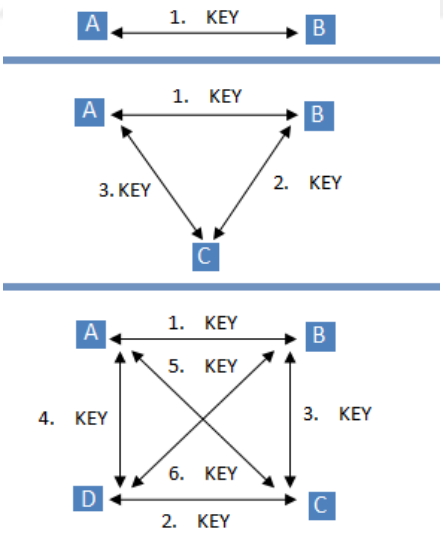

Figure 2.7 Key distribution for symmetric-key cryptosystem

For asymmetric-key cryptography, key management is easier. Since in asymmetric-key cryptography, everybody has only one public-key and one private-key, the

number of required keys is regardless of the population of the network and it is fixed. Therefore, key management for asymmetric-key cryptography is not as hard as key management for symmetric-key cryptography.

2.2.6. Key Exchange and Diffie-Hellman Algorithm

Diffie-Hellman Algorithm was the first public-key algorithm found in 1976. The idea behind Diffie-Hellman Algorithm is to generate a private-key that can later be used for communication [18]. Diffie-Hellman is known as a simple asymmetric-key algorithm for key exchange. This protocol enables two users to establish a public-key scheme based on discrete logarithm.

Two people, let's say the sender and the receiver, can use this algorithm to generate a secret key for key distribution. First, sender and receiver agree on large prime numbers n and g such that g is primitive mod n. They perform the following steps:

1. The sender chooses a large integer x and sends the receiver, $a = g^x \ mod \ n$.
2. Similarly, the receiver chooses a large integer y and sends the sender, $b = g^y mod \ n$
3. The sender computes k from b that the receiver sent where $k = b^x \ mod \ n$
4. Similarly the receiver computes $k' = a^y \ mod \ n$.

Both k and k' are equal to $g^{xy} \ mod \ n$. Any person listening to the conversation would only know n, g, a and b. They cannot recover x and y because of the Discrete Logarithm Problem. The security lies on choosing large values of n and g.

3. RSA (Rivest-Shamir-Adleman) ALGORITHM

In 1977, Ron Rivest, Adi Shamir and Leonard Adleman introduced a cryptographic algorithm, RSA, which is named for the first letter in each of its inventors' last name [19]. RSA's motivation is Diffie-Hellman Algorithm which describes the idea of such an algorithm that enables public-key cryptosystem. Here are the steps of RSA Algorithm:

- The first step of RSA Algorithm is to select two different prime numbers, p and q.
- The second step is the calculation of n where $N = p * q$.
- The calculation of $\Phi(N) = (p-1) * (q-1)$ is the third step.
- As a fourth step, an integer e is selected as a public-key which is co-prime with $\Phi(N)$.
- Finally, the inverse of e modulus $\Phi(N)$ is taken to produce d, the private-key. By using e and $d\ modulus\ N$, the encryption and decryption are done.

In the RSA Algorithm, the public-key includes two numbers N and e, while the private-key is N together with a different number d. Given a numerical message M is encrypted by

$$M \rightarrow M^e (mod\ N) = C \tag{3.1}$$

Similarly an encrypted message C is decrypted by the message

$$C \rightarrow C^d (mod\ N) = M \tag{3.2}$$

For the implementation of RSA, the number N is a product of two large prime numbers p and q. If you know p and q you can obtain d from e. As N is a part of the public-key and it is the multiplication of p and q, then in principle it is possible to factorize N to find p and q.

3.1. Simple RSA Example

Here are the steps of example RSA:

- Find two primes p and q.
- We will choose $p = 11$ and $q = 5$.

- We define $N = p * q = 55$ and $k = (p-1) * (q-1) = 40$.
- Find a random integer e which is co-prime with k where $k = 2^3 * 5$
- We can choose $e = 7$, since $\gcd(7,40) = 1$.
- Take your message M, encode it as whole number in the range $0 \leq M < N$. If your message is too long, then break the message into blocks in this range.
- The message M is encrypted into the C in the range $0 \leq C < n$ under the rule

$$C = M^e \ (mod\ N)$$

- Compute the unique whole number d such that $de = 1(mod\ k)$ and d is in the range $1 \leq d < k$ because e and k are co-prime, we can make sense of $d = e^{-1}$. In our example, $e = 7$ and $k = 40$, we can do this by trial and error and we find $d = 23$

$$7 * 23 = 161 = 1(mod\ k)$$

- Given the encrypted message C, this can be decrypted back into the message M by taking

$$M = C^d \ (mod\ N)$$

- Take the previous value of $N = 55, k = 40, e = 7$ and $d = 23$.
- Suppose our message assigned as $M = 13$.
- In order to encode it, we must calculate the value of C:

$$C = M^e (mod\ n) = 13^7 (mod\ 55)$$

$$13^2 = 169 = 3 * 55 + 4 = 4\ (mod\ 55)$$

$$C = 13^2 * 13^2 * 13^2 * 13 = 4 * 4 * 4 * 13 = 64 * 13 = 7\ (mod\ 55)$$

- Decoding $C = 7$, we should obviously get $M = 13$. In this example M is given by

$$M = C^d (mod\ n) = 7^{23}\ (mod\ 55)$$

$$\text{Since } 7^2 = 49 = -6\ (mod\ 55),$$

$$7^6 = (7^2)^3 = (-6)^3 = -216 = 4\ (mos55)$$

$$= 7^6 * 7^6 * 7^6 * 7^2 * 7^2 * 7\ (mod\ 55)$$

$$= 4 * 4 * 4 * (-6) * (-6) * 7$$

$$= 64 * 6 * 6 * 7$$

$$= 54 * 42$$

$$= -1 * 42$$

$$= -42 \equiv 13\ (mod\ 55)$$

3.1.1. The Relation between e, d and Euler's Phi Function

Modular arithmetic is known as clock arithmetic. All we know clock starts from 12 and ends with 12. The space of clock is 0 to $N-1$=11 where $N=12$. $\{0,1,2,3,4,5,6,7,8,9,10,11\}$ is the space of clock and 0 is congruent with 12. Suppose the time 12 o'clock right now. 3 hours later, it will be 3. However, 15 hours later, it will be 3, too. This equality or congruence is the result of working in modulus 12 where 15≡3 mod 12. Suppose your clock starts from 12 and it is in modulus 12 but it does not increase by 1 each time. It increases by 2 each time. Then, its space gets narrow and becomes {0, 2, 4, 6, 8, 10} where 0 is congruent with 12. If it increases by 3 each time, then its space gets narrow and becomes {0, 3, 6, 9} where 0 is congruent with 12. If it increases by 5 each time, then its space does not change. But the order of the space becomes {0, 5, 10, 3, 8, 1, 6, 11, 4, 9, 2, 7}. If it increases by 7 each time, then its space length does not change again. But the order the space becomes {0, 7, 2, 9, 4, 11, 6, 1, 8, 3, 10, 5}.

As seen above, if the incremental value is co-prime with modulus, N, 12, the result space does not get narrow and the space does not change. When it is co-prime with N, then you will use every number exactly once before returning to starting value 12. However, when the incremental value is not co-prime with N, 12, the result space gets narrow.

For RSA, the relation between e, d and Euler's Phi Function can be explained by using clock arithmetic. Since $M^e = C(mod\ N)$ and $C^d = M(mod\ N)$, for every distinct message value, M, each message value has exactly one and unique corresponding cipher value, C in mod N owing to being co-prime with phi(N).

Message	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Cipher	1	8	27	31	26	18	13	17	3	10
Message	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Cipher	11	12	19	5	9	4	29	24	48	14
Message	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Cipher	21	22	23	30	16	20	15	7	2	6
Message	**31**	**32**								
Cipher	25	32								

Figure 3.1 The one to one matching of cipher-text values and plain-text values

For example, for e = 3, N = 33, q = 3, p = 11, M = [(1) to (32)], Phi(N) = 20, for every distinct message value of M, there is exactly one and unique corresponding cipher-text value as seen in Figure 3.1. As shown, for all different message values from 1 to N-1, the corresponding cipher-texts are unique and different. There is no repetition.

3.1.2. Prime Numbers and Prime Number Generation

An integer n is a prime if and only if n has two divisors 1 and itself, n. There are infinitely many prime numbers in a set $\mathbb{P}$ where $\mathbb{P} = \{2,3,5,7,11,13, \dots\}$.

2	3	5	7	11	13	17
19	23	29	31	37	41	43
47	53	59	61	67	71	73
79	83	89	97	101	103	107
109	113	127				

Figure 3.2 The generated prime numbers with key size=7 bits

As it is seen in Figure 3.2, prime numbers does not follow any linear pattern. There is not any function that gives all prime numbers from 2 to positive infinity. In order to find prime numbers, the wrong way is generate random numbers and then try to factor them. It can be time consuming. The correct way can be generating random numbers and test their primality. Although finding a formula or pattern to list all prime numbers from 0 to positive infinity is not feasible yet, classifying prime numbers after generating them by using primality tests is possible. Prime numbers can be categorized as Sophie-Germain Primes, Safe Primes and Strong Primes. Sophie-Germain Prime is a prime number p such that $2 * p + 1$ is also prime number. Safe Prime is a prime number q such that $2 * p + 1 = q$ where p is also prime. Strong Prime is a prime number q such that $q = \frac{p-1}{2}$ is prime, $\frac{p+1}{2}$ is prime, $\frac{q-1}{2}$ is prime and p is also prime.

3.1.3. Attacks on RSA

In cryptography, encryption aims at providing security and secrecy to any communication between the sender and the receiver. While providing security,

encryption or encryption-like transformations of information often uses some mechanisms like enciphering, digital signature and access control against the cryptographic attacks. These attacks are classified as passive or active attacks [20]. If attacks are used to access and to obtain information for eavesdropping on or monitoring, these attacks are passive attacks. Passive attacks are generally very difficult to detect because these attacks do not involve any alteration of information. If attacks involve some modification of information, these attacks are classified as active attacks and it is difficult to prevent active attacks.

Some of popular attacks on RSA can be listed as below:

- **Cyclic Attack**

  If the sender and the receiver do not take care in determining their keys, the attacker can intercept the encrypted message sent by the sender to the receiver. Suppose RSA variables of our system are $p = 3$, $q = 17$, $N = 51$, $\Phi(N) = 32$, $e = 3$, $d = 11$, $M = 5$, and $C = 23$. The cipher-text starts with 23. By attempting to crack it by using cyclic attack, the attacker can find d, p and q values.

$$Encrypt(23) = 23^3 \ (mod\ 51) = 29$$

$$Encrypt(29) = 29^3 \ (mod\ 51) = 11$$

$$Encrypt(11) = 11^3 \ (mod\ 51) \ = 5 = M$$

$$(This\ is\ the\ original\ message\ encrypted\ by\ the\ sender)$$

$$Encrypt(5) = 5^3 \ (mod\ 51) = 23 = C$$

$$(This\ is\ the\ original\ cipher - text\ decrypted\ by\ the\ receiver)$$

  As it is shown, the attacker reached the values of p, q, and d by the cycling re-encryption.

- **Low-Encryption Exponents Attack**

  For small values of e and for small values of the plain-text, by taking the e^{th} root of the cipher-text over the integers, the cipher-text can be easily decrypted. In order to improve the system performance, e can be chosen as small as possible but it leads to some security leaks [21]. The smallest possible value for e is 3 but to avoid some certain attacks the value $2^{16} + 1 = 65537$ is recommended.

- **Forward Search Attack**

  For this attack, if message space is not wide and also if the attacker can predict the message, the attacker can decrypt the cipher-text by encrypting all possible messages until he obtains a match with the cipher-text [22].

- **Common Modulus Attack**

  For this attack, allowing two different receivers to share the same modulus N for RSA can lead to Common Modulus Attack. Thus, for RSA, modulus N should not be used by more than one communication [23].

- **Low Decryption Exponents Attack**

  To improve the performance of RSA decryption, the d value can be chosen as small rather than a large number. Indeed, small d value improves the system performance dramatically but the system become vulnerable to attacks [24].

- **Chosen Cipher-Text Attack**

  A Chosen Cipher-Text Attack is an attack where the cryptanalyst chooses a cipher-text and let it decrypt. From some pairs of cipher-text and decrypted cipher-texts, he obtains information about key or about the message. Since $m1^e * m2^e = (m1 * m2)^e$ is mathematically valid, the Chosen Cipher-Text Attack can be possible where $m1$ is chosen as small number and sent by the attacker. Since $m1$ is small, it can be analyzed easily.

- **Brute-Force Attack**

  A Brute-Force Attack involves trying all possible public-key and private-keys until finding the right ones. The attacker tries all possible public-key and private-key combinations. RSA with short key size is vulnerable against this attack. However, Brute-Force Attack is useless for long key sizes.

- **Cipher-Text Only Attack**

  For this attack, the attacker knows only the cipher-text and he tries to recover the plain-text by using cipher-text by trial and error.

- **Factorization Attack**

  The security of RSA is based on the idea that the modulus is so large that is not possible to factor it in a short time. The sender selects p and q and he calculates $N = p * q$ where N is public. Although N is public, p and q are private. Since e is also public and $e * d = 1\ (mod\ \Phi(N))$, any attacker who knows p and q values, can calculate d value which is used to decrypt any encrypted message. In this thesis, some of Factorization Attacks like Brute-Force Attack, Fermat Attack, Pollard-Rho Attack and KNJ Attack will be examined in detail.

3.1.4. RSA Implementation Hints

Here are the recommended hints that increase the security level of RSA implementation:

- p and q should not be very small. Since $N = p * q,\ \ N$ can be easily factorized if p and q values are both small.
- Primes p and q should not be too far from each other. If one of them is too small, it makes factoring of n easier.
- p and q primes can be selected from "Strong Primes" where $(p-1)$ and $(q-1)$ have large prime factors. This can make factorization process of N more complex.
- e and d should not be very small. Let's describe a simplified version RSA encryption. As emphasized before, in any encryption operation, the main objective of the attacker is to recover the message from the cipher-text. In order to reduce the load of exponentiation in RSA, one may prefer to use a small value of private-key or public-key values rather than a random value. For small private-key values, RSA system becomes vulnerable to attacks.
- e and d values should not be equal, where $e \neq d$. Since decryption process is the inverse of encryption process, e and d should not be equal. If they are equal, encrypted value of plain-text can be recovered easily.
- The most frequently selected e values are $e = 3, e = 17, e = 2^{16} + 1$ [25]. These 3 different e values have a common feature. They have only 2 times 1 bits in their binary representation. Therefore, total number of multiplications needed to perform exponentiation is minimized.

$$e = 0000\ 0011 = 3$$

$$e = 0001\ 0001 = 17$$

$$e = 1000\ 0000\ 0000\ 0001 = 2^{16} + 1$$

- For RSA system, the public-key is (N, e). The private-key is (N, d). These integers need to be very large to avoid easy factorization of N. Since the maximum integer type value generated in Java is 31 bits, it corresponds to $2^{31} - 1 = 2147483647$. In order to generate numbers bigger than 2147483647, Java uses BigInteger instead of using Integer class.

3.2. Cryptology and Java

A cryptographic Application Programming Interface (API) enables a programmer to implement cryptographic techniques. There are few cryptography APIs like Java Cryptography Architecture (JCA), Microsoft Cryptography API, Bouncy Castle Crypto API and Linux Kernel Cryptographic API [26]. These API's scopes are very wide. Some API's are written only for one language and others are written for more than languages. Java is a popular and multi-platform language that wide-spreads the usage of cryptography to almost every systems. A Java framework that provides a cryptographic API is built by Java Cryptography Architecture (JCA) and the Java Cryptographic Extension (JCE) [27]. These architecture and extensions enable users to use any cryptographic concept without to worry about the underlying details. Although there are a lot of providers out there already, I implemented my own cryptographic service provider by using Sun's JCA and JCE. I specifically used 3 different packages javax.crypto, javax.crypto.interfaces and javax.crypto.spec.

3.2.1. BigInteger Class

The security of RSA depends on N, p, and q. If these values are sufficiently large, the RSA system is sufficiently secure. For commercial applications, N is typically chosen to be usually 1024 bits and for more critical applications, it is chosen as 2048 bits [28]. Since $N = p * q$ and p and q values are generally chosen as equally long such as B bits long, the length of N is 2B bits. Therefore, integer or long types become useless for creating such a long keys. Therefore, programmers choose to use big number libraries. Java uses BigInteger class. BigInteger provides analogues to all

of Java's primitive integer operators and some methods. Moreover, BigInteger provides some pre-defined methods like primality testing and greatest common divisor. These methods make easy to implement cryptographic operations in Java.

3.2.2. Random Class and SecureRandom Class

The security of many cryptographic systems depends on the generation of unpredictable variables where these variables must be of sufficient size and random. The randomness means that the probability of any particular value being selected for any variable must be sufficiently small and equally probable [29]. Random numbers are extremely useful in cryptography in RSA Algorithm implementation especially while deciding on p, q and d values for encryption and decryption.

For RSA, picking a number in predetermined bit length seems simple. However, if you need truly random numbers which are equally probable, the task is quite complex. Up to now, some robust techniques to generate random numbers are developed. Since computer's all actions are predictable and deterministic, generating random numbers truly is not possible. But generating pseudo random numbers and getting close to generate random numbers is possible.

Java's Random class provides some methods to generate pseudo random numbers. This class uses a 48 bit seed [30]. This seed is very vital to generate the key. The seed is to generate the random algorithm to avoid being deterministic. If any pattern in the seed is caught, this pattern can / will propagate itself in the key. In the design of random generator, the most important feature to consider and not to neglect is that the random bit generator should not be observed and manipulated. The natural source of randomness of random bit generators is subject to affect by external factors and malfunction. Since Random class uses the system clock to generate the seed, the generated numbers can be reproduced easily by the attacker if he knows the time when the seed was generated. However, SecureRandom class of Java uses the random data from operating system and uses that data to generate the seed. Since generated seed uses a random data in SecureRandom class rather than a pseudo random data, this feature makes the usage of SecureRandom class in cryptography more appropriate than Random class because SecureRandom class produces non deterministic output instead of deterministic output.

In order to indicate the effect of random number generator (RNG) in the implementation of RSA Algorithm in Java, during the creation of RSA key pairs, the following experiment was done. The RSA Algorithm was implemented in 2 different Java projects. The first project uses Random class and the second project uses SecureRandom class as a random number generator for the creation of p and q. Every line of code, except the line of random number generator class type, used in these 2 projects are identical. The histograms of the RSA Algorithm's variables p, q and N indicate the effect of RNG on the generated key pairs.

Table 3.1 Generated key pair values by using Java's Random class

p	q	N	e	d
19	17	323	73	217
17	19	323	215	71
31	23	713	323	47
19	31	589	517	493
23	29	667	45	397
23	31	713	277	193
31	19	589	401	101
23	29	667	213	509
31	19	589	127	523
31	17	527	101	461

In this experiment, the distribution of corresponding p, q, and N values for these two classes were saved and examined. Random class and SecureRandom class were set to generate 5 bit long p and q values. Therefore, the calculated N value can be utmost 10 bit long. With the analysis of the experiment, the behavior of used random generator class were estimated and experienced.

Table 3.2 Generated key pair values by using Java's SecureRandom class

p	q	N	e	d
19	23	437	145	325
19	17	323	205	229
29	17	493	303	207
29	19	501	73	145
17	31	527	397	133
31	17	527	149	29
31	19	589	413	17
23	29	667	327	535
17	31	527	311	71
31	29	899	289	529

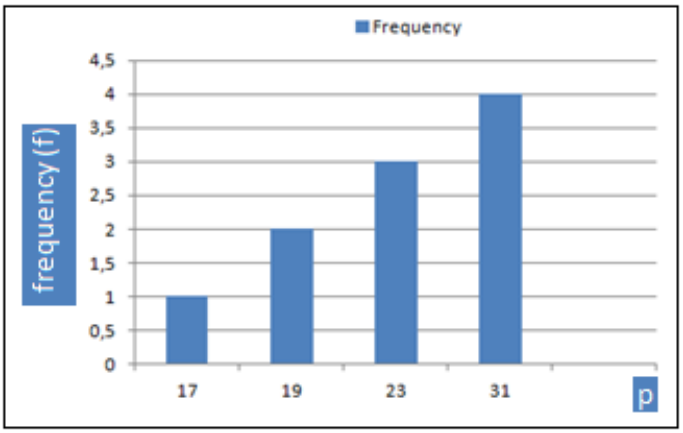

Figure 3.3 The histogram of p values generated by Random Class

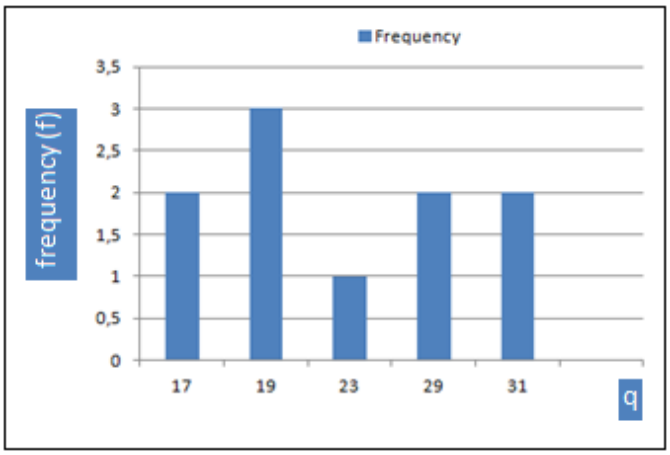

Figure 3.4 The histogram of q values generated by Random Class

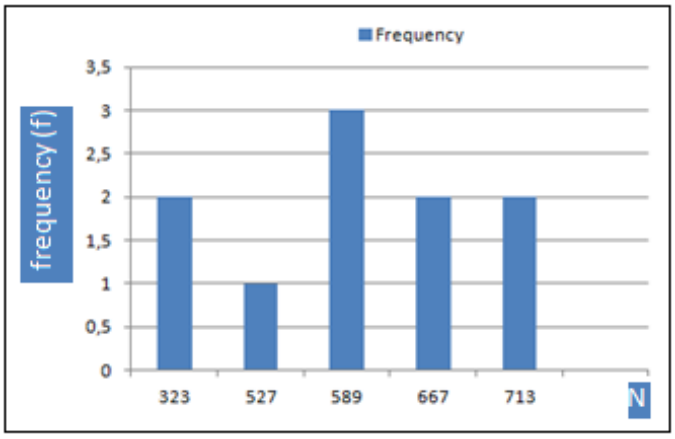

Figure 3.5 The histogram of N values generated by Random Class

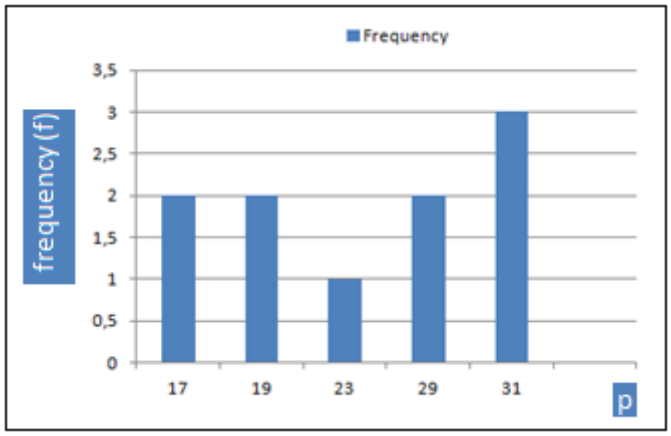

Figure 3.6 The histogram of p values generated by SecureRandom Class

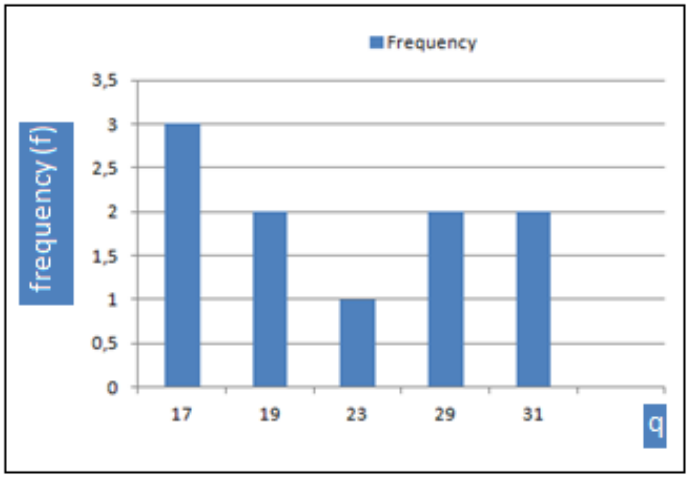

Figure 3.7 The histogram of q values generated by SecureRandom Class

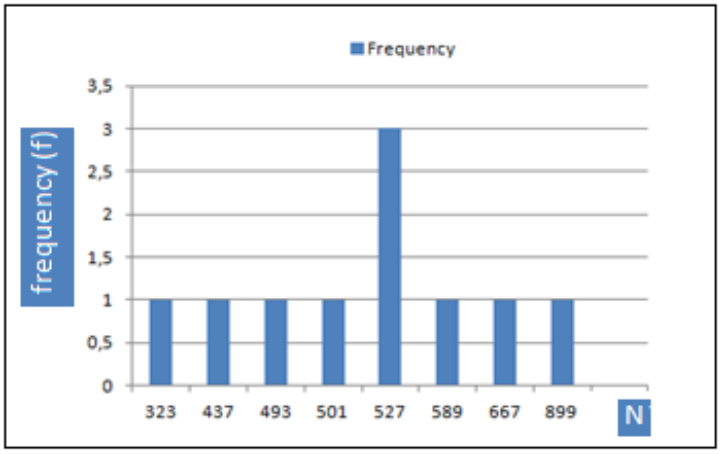

Figure 3.8 The histogram of N values generated by SecureRandom Class

When Random class of the Java used instead of SecureRandom class for number generation, the N values' distribution does not seem equally likely. However, SecureRandom can be labeled as "True Random Generator" owing the distribution of N values of it shown in Figure 3.8. Since N values created by Random class are not equally probable, the Random class cannot be classified as truly random as it is expected. Beyond being truly random generator, SecureRandom class provides more various N values to be used in encryption and decryption processes. In Figure 3.5, there are 5 different N values generated by Random class. However, in Figure 3.8, there are 8 different N values generated by SecureRandom class. The difference between the numbers of generated different N values, indicates the positive effect of SecureRandom class for random number generation in Java.

3.2.3. Symmetric Cases of Plain-text and Cipher-text for RSA

The principle of encryption and decryption for RSA depends on the public-key pair (e, N) and private-key pair (d, N). The cipher-text is found by the formula $C = M^e \ mod \ N$ and the plain-text is found by the formula $M = C^d \ mod \ N$. While implementing RSA encryption, for some M values from 1 to $N - 1$, the plain-text can be equal to the cipher-text. Although the case where $C = M$ is against the spirit of encryption (hiding information), it is still possible to meet such cases. In order to visualize these kinds of cases in RSA, Figure 3.9 can be examined.

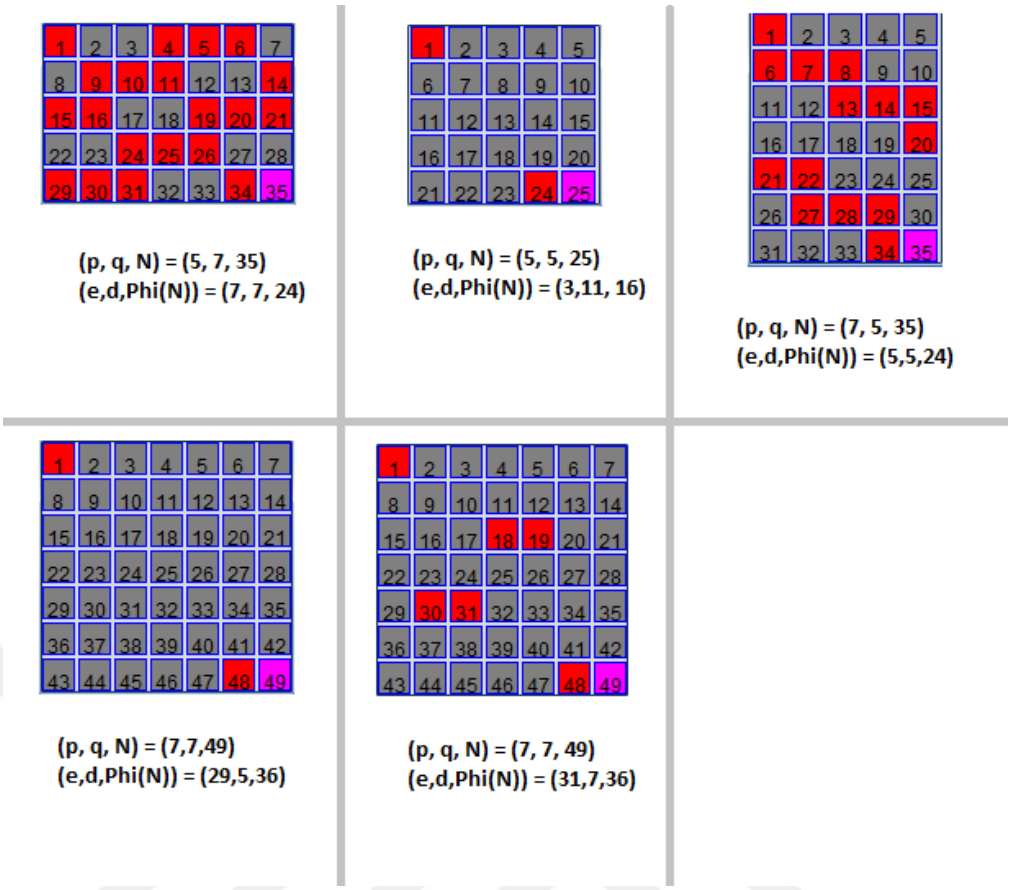

Figure 3.9 The distribution of the cases where plain-text equals to cipher-text

For $p = 5\ or\ p = 7$ and $q = 5\ or\ q = 7$, related private-key and public-key pairs were created and messages from 1 to N were encrypted according to the principles of RSA with the indicated public-key and private-key pairs. The cases where plain-text equal to cipher-text indicated by red boxes in Figure 3.9. During this experiment, the symmetry between positions of message values of red boxes was observed. If you examine one of these 5 different cases, the distribution of red boxes from 1 to N and the distribution of red boxes from N to 1 are symmetric. Moreover, the summation of the value of symmetric red boxes is equal to N. Due to the fact that with the increase in the value of p and q values, the number of red boxes, where the plain-text equal to cipher –text, decreases, I emphasized only limited number of p, q, N pairs.

4. IMPLEMENTATION of RSA in JAVA

RSA Algorithm for cryptography consists of three main stages: Key Generation Stage, Encryption Stage and Decryption Stage. Therefore, RSA was implemented in Java by considering these 3 stages. Key Generation Stage is the process of generating keys for cryptography. Keys, generated in this stage, are used to encrypt the plain-text in Encryption Stage and used to decrypt the cipher-text in Decryption Stage. Encryption Stage is the process of encoding messages in such a way that only authorized people can understand it. By encryption, the message is converted into cipher-text. Decryption Stage is the process of decoding the cipher-text to get the original message.

4.1. Cryptography for RSA

There are four different screens for the users who want to implement RSA in Java. These screens are “Generate RSA Keys”, “Generate Manual Keys”, “Encrypt Message” and “Decrypt Message” screens. The first two screens are for generating RSA’s public-key and RSA’s private-key. The third screen is for encrypting an inputted message and the fourth screen is for decrypting a resulting cipher-text.

4.1.1. Implementation of RSA without Padding

In the key generation screen as shown in Figure 4.1. $p,\ q, N, \Phi(N), e$ and d values are generated or calculated according to the selected key size. The key size can be the values from 4 to 2048 in bits. The key values can be generated randomly or can be generated manually by using “Create Manual Keys” option. In Figure 4.1, 32 bits is chosen as a bit length of p and q values. Firstly, p and q values are generated. With the multiplication of these values, N value is calculated as 64 bits long. Required data values, $\Phi(N), e,$ and d values, are created according to the scenario of RSA Algorithm. After completing key generation step, user becomes able to encrypt any plain-text and decrypt any cipher-text by using generated or calculated RSA data values.

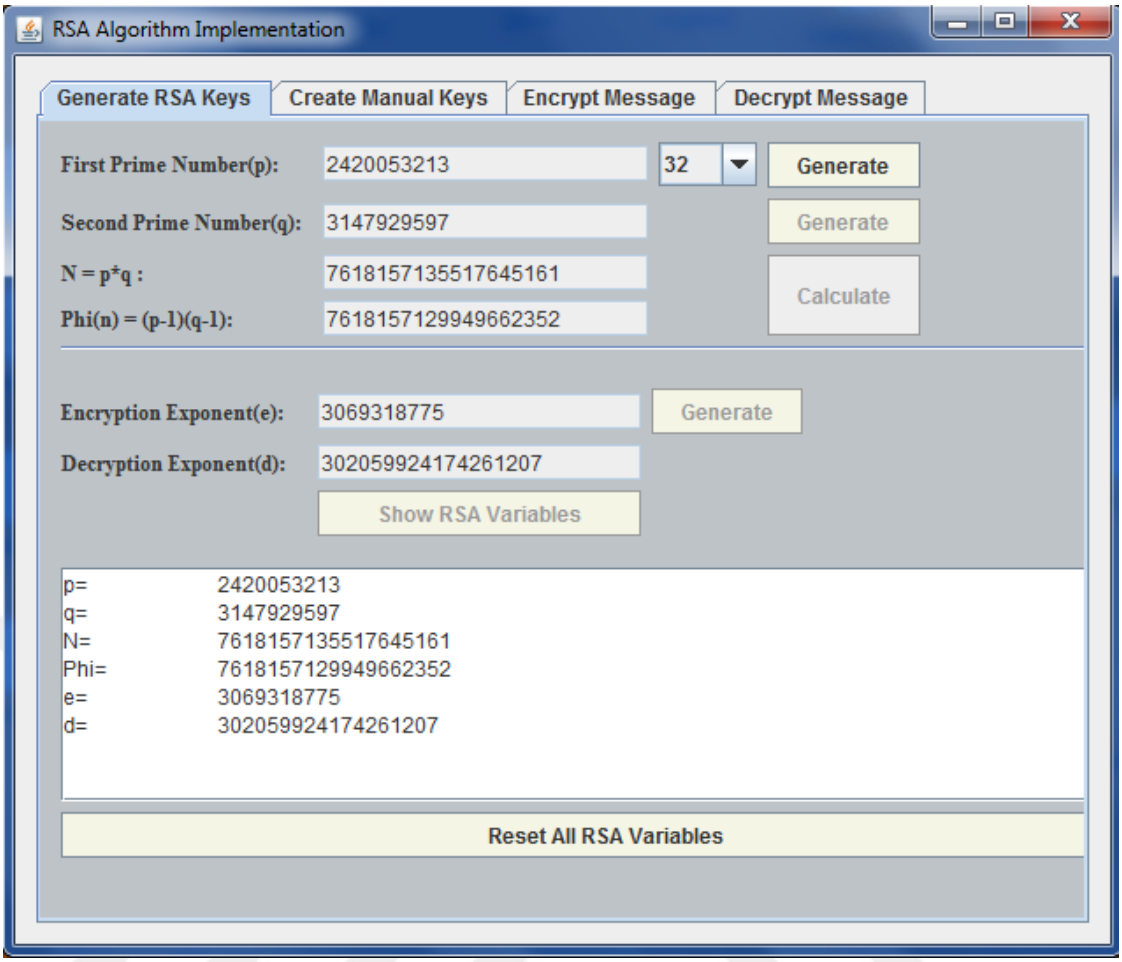

Figure 4.1 The key generation screen for RSA without padding

In the encryption screen, the input plain-text is converted into corresponding numerical ASCII value where ASCII is the abbreviation of American Standard Code for Information Interchange. ASCII allows any text to be represented numerically. In order to generate cipher-text, each character of plain-text, whether letter, number, punctuation mark or space character, is converted into their numerical ASCII values by one by. These numeric values are processed according to RSA implementation steps. Resulting values for each character are concatenated to compose cipher-text. As seen in Figure 4.2, the plain-text is "Mustafa KOCAKULAK". 32 bits long p and q values and 64 bits long N value were generated and calculated in Figure 4.1. These key values were used to create cipher-text in Figure 4.2.

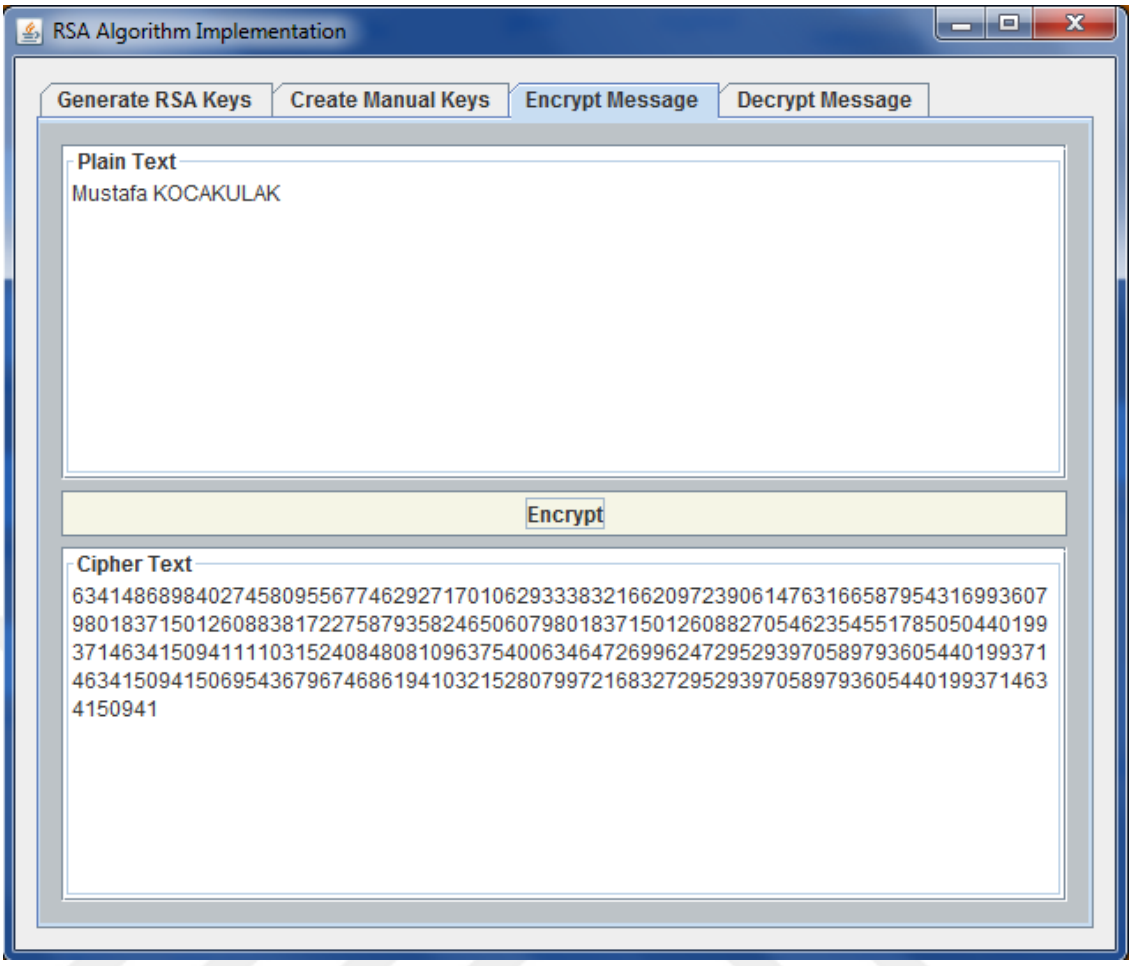

Figure 4.2 The encryption screen for RSA without padding

In the decryption screen, the encrypted values of each character of the plain-text were converted into decrypted numeric values. These numeric values were converted into their ASCII values. After completion of decryption process, the original plain-text was recovered as seen in Figure 4.3.

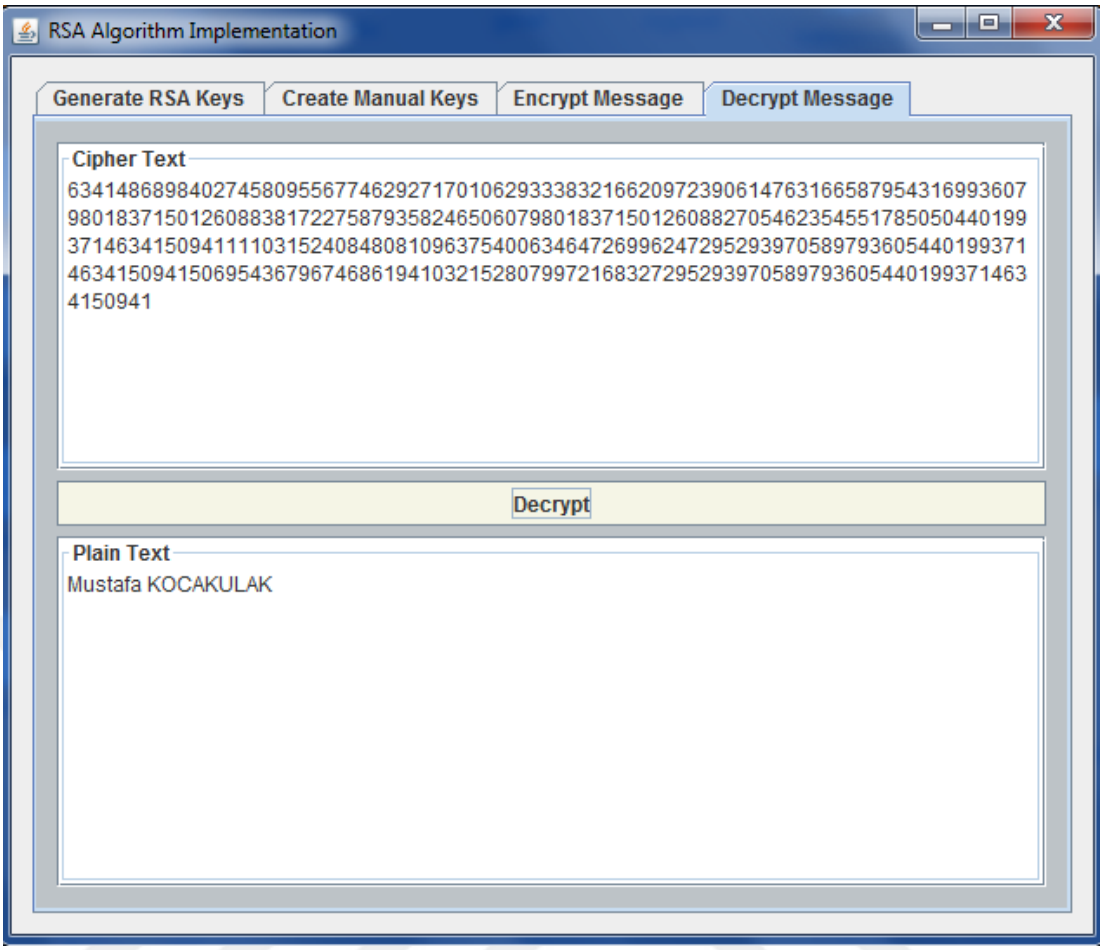

Figure 4.3 The decryption screen for RSA without padding

In decryption screen, since the attacker knows only the cipher-text, he can try to recover the plain-text by Cipher-Text-Only Attack. The attacker looks for finding a pattern in the cipher-text. Since any pattern in cipher-text can / will propagate itself in the key, he decrypts the cipher-text. In Figure 4.4, plain-text is "Mustafa KOCAKULAK" and it contains repetitive 'K' letter. Due to not using padding feature, cipher-text contains propagating encrypted 'K' values which corresponds to '4401993714634150941'. The frequent propagation of this numeric value makes the attacker able to decrypt the cipher-text.

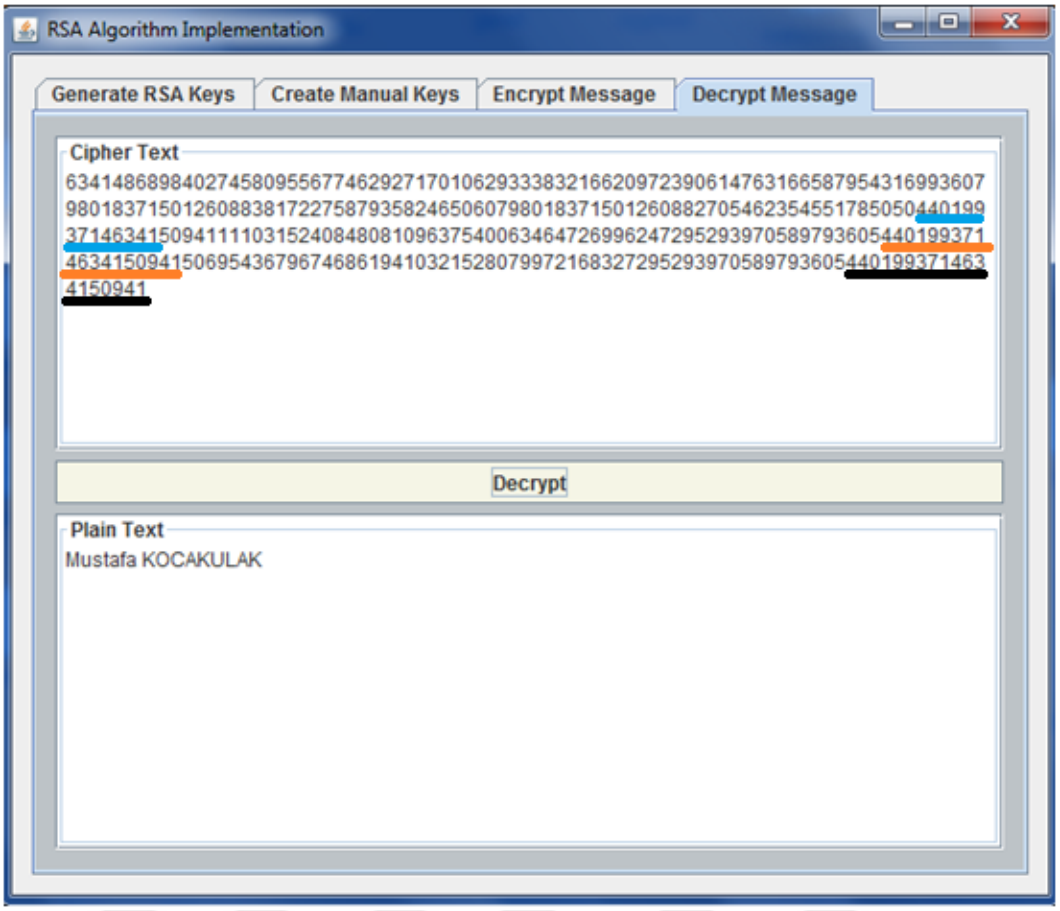

Figure 4.4 The vulnerability of RSA to Cipher-Text-Only Attack

The recovery from cipher-text to plain-text is the sign of vulnerability of the RSA system to the Cipher-Text-Only Attacks. In order not to scarify the system security, padding must be applied to the RSA algorithm implementation. Thus, recovery from cipher-text to plain-text is avoided.

4.1.2. Implementation of RSA with Padding

Up to this section, encryption and decryption process of RSA were examined. However, encryption or decryption process of RSA is not as vital as padding process. Padding is an armoring process of plain-text during the encryption and it is not an optional process for RSA. Padding is necessary for RSA against certain attacks and to enable the plain-text to be reconstructed after encryption [31]. Since the basic principle of any cipher is to confuse hackers and never to establish a pattern that can be broken, padding has an indisputable importance for RSA implementation for

secure communication supply [32]. Due to the fact that attackers look for finding any propagation in cipher-text to recover plain-text from cipher-text, padding is one of the most used precautions to avoid such a security leak.

Padding ciphers work on fixed-sized output. As an example of RSA Implementation with padding, 512 bits long p and q were used. In encryption process, plain-texts are encrypted usually in groups of bits. These groups of bits are blocks. If a plain-text is less than block size, then it must be padded with additional data. A constant byte is added to the end of the message to make its length equal to the block size. Therefore, the message length must be known before applying required padding.

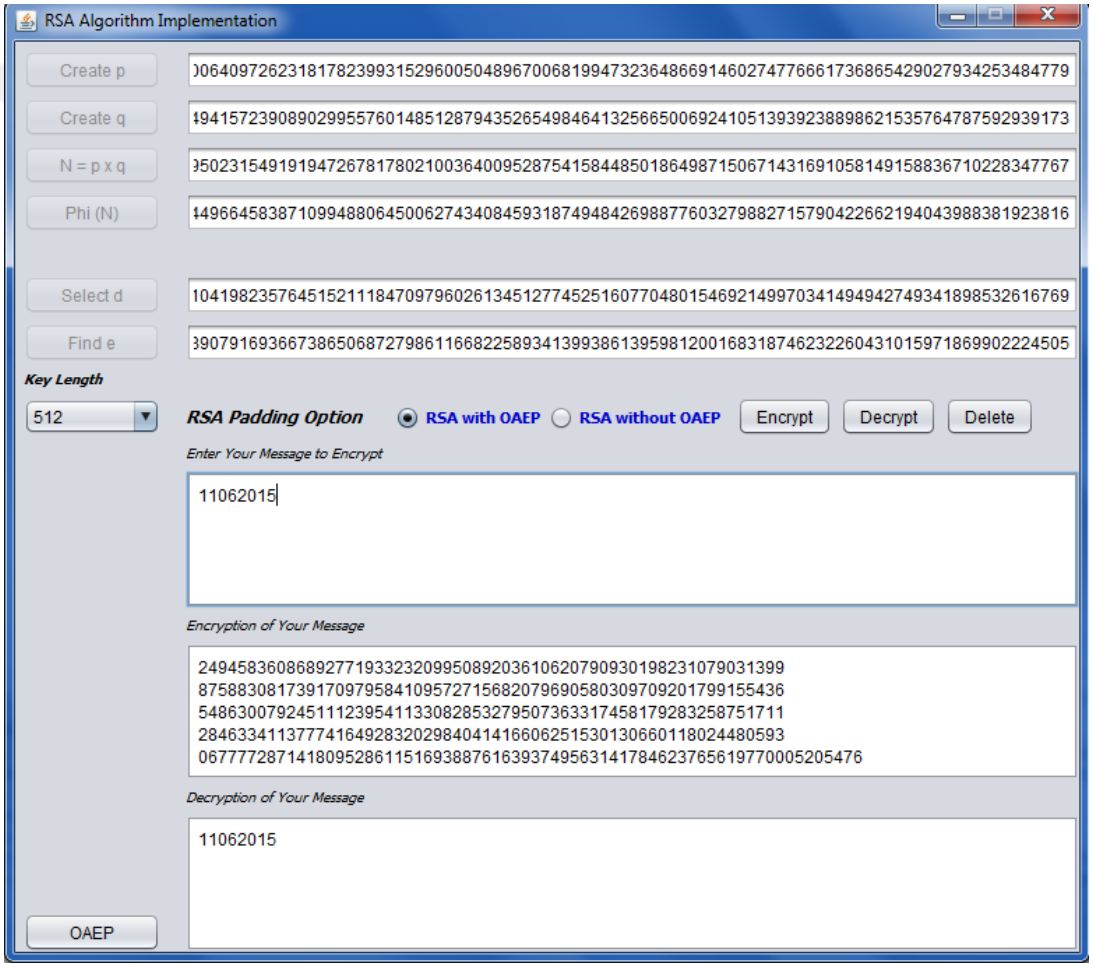

Figure 4.5 The implementation of RSA with padding

Since 512 bits is commonly used key lengths for RSA, the ASCII value of plain-text was padded and RSA encryption was done by using 512 bits long key pairs. As seen in Figure 4.5, the plain-text is “11062015”. Not containing any repetitive sequence in

the cipher-text can be evaluated as a complexity or positive effect of padding on encryption. Therefore, the corresponding cipher-text in Figure 4.5 is more complex and more secure if it is compared with cipher-text created by RSA without padding. The attackers cannot find any repetitive sequence in the cipher-text owing to padding effect. However, Implementing RSA without padding sacrifices the security due to the possibility of propagating some sequences in the cipher-text.

4.2. Cryptanalysis for RSA with Factorization Methods

Generally, cryptanalytic attacks rely on the basis of the algorithm and general features of plain-text. For RSA, the security of algorithm depends upon N which is the multiple of p and q. Finding the value of p and q is known as factorization which is the inverse process of multiplication. Factorization of N is easy for small N values but for great numbers, it is very problematic. Here are four different factorization methods for cryptanalysis of N value of RSA as: Brute-Force Factorization, Pollard-Rho Factorization, Fermat Factorization and KNJ- Factorization. These methods are good at factorization of N up to for limited number of bits due to the limitation of processors of computers.

4.2.1. Brute-Force Factorization Attack

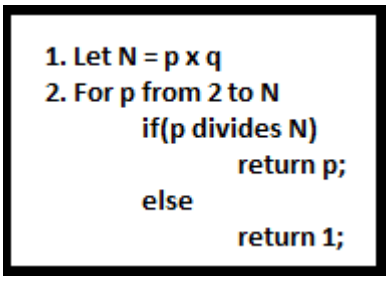

Figure 4.6 Pseudo-code for Brute-Force factorization

Brute-Force Attack is not analyzing the cryptographic algorithm, but trying many permutations of keys until some information is recovered from the plaintext [33]. Trying to divide N by every number between 1 to $\sqrt{N}$ is the simplest and the longest way of obtaining p and q values. It is known as Brute-Force Factorization and shown in Figure 4.6.

4.2.2. Pollard-Rho Factorization Attack

```
1. Let N = p x q
2. Let A=2,  B=2
3. do
        A = A x A
        A = A + 1
        B= B x B
        B = B + 1
        B= B x B
        B = B + 1
  while(gcd(N, A-B)==1)

4. return gcd(N, A-B);
```

Figure 4.7 Pseudo-code for Pollard-Rho factorization [34]

Pollard-Rho Factorization, factorize the N by using 2 different variables. By using a loop and these 2 variables, algorithm checks the divisibility of N. After completion of each iteration, these variables and their changing value makes the algorithm reach the factor of N as shown in Figure 4.7.

4.2.3. Fermat Factorization Attack

```
1. Let N = p x q
2. Compute A = Math.ceil(Math.sqrt(N))
3. Compute A x A
4. Compute B = A x A -N
5.  while(B % (Math.sqrt(B))!=0)
        A = A+1;
        B=A x A -N;
6. Return A - (Math.sqrt(B)).
```

Figure 4.8 Pseudo-code for Fermat factorization [35]

Fermat Factorization is a good technique if p and q have equal distance from $\sqrt{N}$. Fermat Factorization depends on the difference of two squares as seen in Figure 4.8.

4.2.4. KNJ Factorization Attack

```
1. Let N =p x q
2. Compute A = Math.floor(Math.sqrt(N))
3. Check if (A % 2==0)
        4. A=A+1
        5. Check if A's primality (If it is prime)
                6. Compute B = N / A
                        7. Check if B is integer
                                8. return A
                        9. B is not integer
                                A = A -2
        10. A = A-2, A-4, A-6, ..., 7, 5, 3.
11. Repeat step 5- to step 10 until B is integer
```

Figure 4.9 Pseudo-code for KNJ factorization [36]

KNJ-Factorization works well if p or q is close to $\sqrt{N}$. Algorithm starts checking the numbers from $\sqrt{N}$ to 1 instead of starting checking from 1 to N as seen in Figure 4.9.

4.2.5. Comparison and Result

For varying N values, the iteration numbers for 4 different factorization methods: Brute-Force Factorization, Pollard-Rho Factorization, Fermat Factorization and KNJ Factorization are shown in figures from Figure 4.10 to Figure 4.11. These figures were drawn by using JFreeChart library. In these figures, it is clearly seen that iteration number for Brute-Force Factorization in all cases, is much greater than other factorization methods' iteration number. As N increases, the efficiency of Brute-Force Attack decreases. As p and q values are close to $\sqrt{N}$ in these figures, the efficiency of other 3 methods increases and iteration numbers for these methods decrease. For example, for $N = 35$, $p = 5$ and $q = 7$. As seen, p and q values are close to $\sqrt{35} \cong 6$. Except Brute-Force, iteration numbers for all 3 factorization are low owing to close position of p and q values. Although iteration numbers are close each other for the methods other than Brute-Force, the most effective one is Fermat since p and q have nearly equal distance to $\sqrt{N}$ as seen in Figure 4.10 and 4.11. For greater values of N, the difference between the iteration numbers of factorization methods increases.

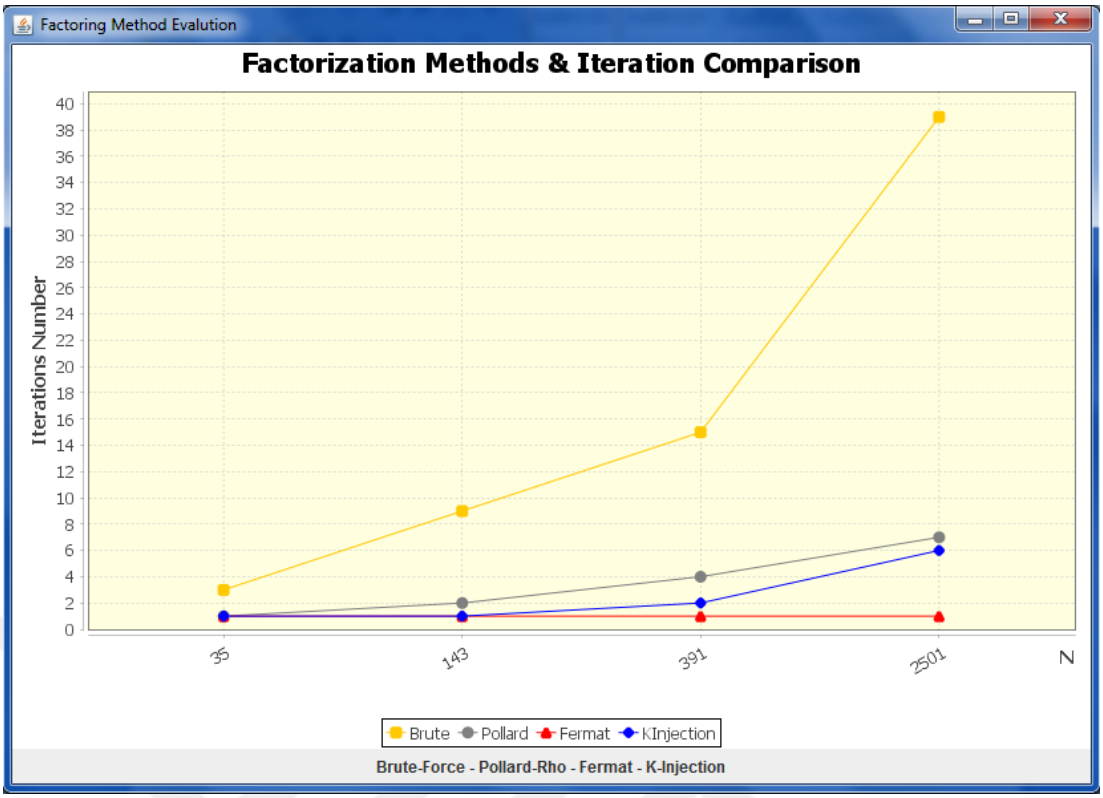

Figure 4.10 Iterations of factorization methods for varying small N values

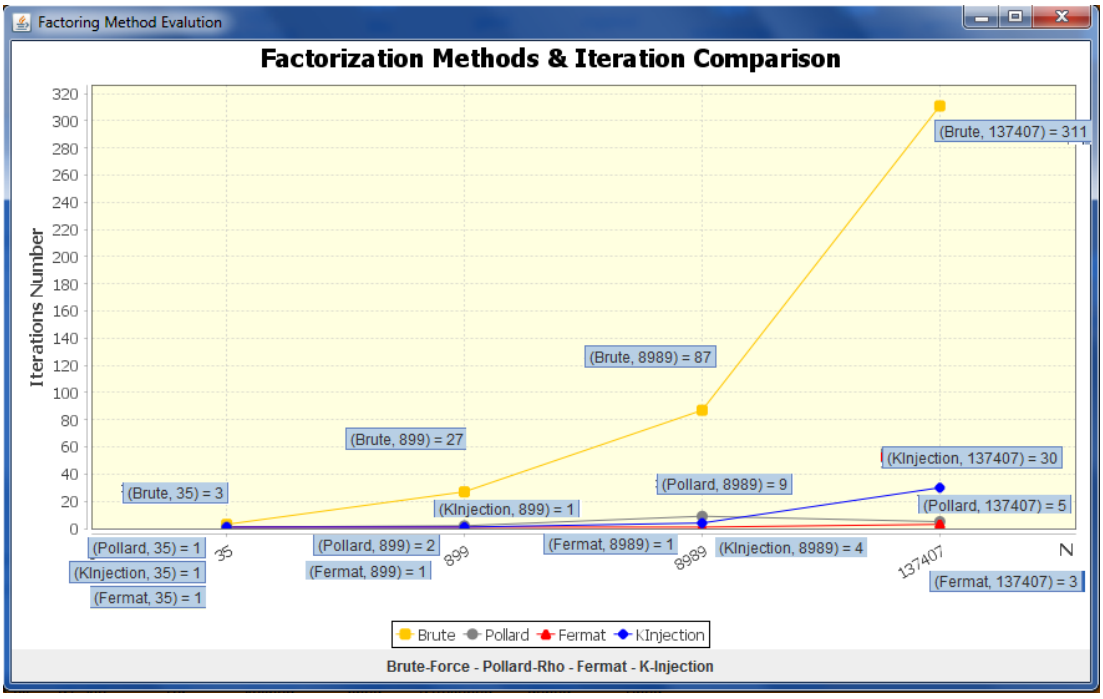

Figure 4.11 Iterations of factorization methods for varying bigger N values

After comparison of iteration numbers of 4 factorization methods, comparison of elapsed time values of 4 factorization methods was done. In order to compare the elapsed time values for the factorization of varying N values, 15 specific N values with different bit length, were created as shown in Table 4.1.

Table 4.1 Generated sample RSA key pairs for comparison of elapsed times

Size of N (bit)	(p, q, N)
16	(137, 193, 26441)
18	(439, 281, 123359)
20	(661, 613, 405193)
22	(1619, 1831, 2964389)
24	(2617, 2473, 6471841)
26	(4447, 7669, 34104043)
28	(11399, 8861, 101006539)
30	(30631, 20719, 634643689)
32	(36299, 36997, 1342954103)
34	(87257, 115811, 10105320427)
36	(311341, 400949, 124831862609)
38	(648191, 953917, 618320414147)
40	(1909283, 1431737, 2733591114571)
42	(2206783, 3986351, 8797011618833)
44	(5310313, 8243899, 43777684030387)

For each factorization method, 15 different N were factored 1000 times. These factorizations were done simultaneously by using parallel computing logic as shown in Figure 4.12.

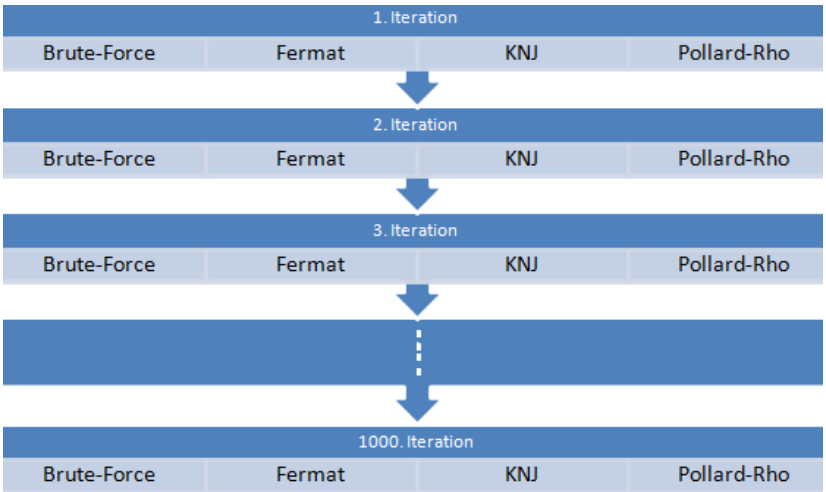

Figure 4.12 Parallel computing of elapsed times for 4 factorization methods

For 4 different factorization methods and for 15 different N values, $4\ x\ 15\ x\ 1000 = 60000$ different time values were obtained and saved in txt files. After creating datasets for 4 factorization methods, these 60000 time values were processed in Matlab. Firstly, the mean values of 1000 elapsed time of each N value, for each factorization method, were calculated. It is shown in Table 4.2. These mean values were used to generate the plots from Figure 4.13 to Figure 4.19.

Table 4.2 Mean value of elapsed times of each factorization method

N (bit) / Methods	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44
Brute-Force (msec)	0.3519	0.4888	0.3120	0.3899	0.3658	5.4706	1.3602	13.7103	0.4516	30.6785	70.7839	51.6396	278.6109	328.4548	409.5940
Fermat (msec)	0.3352	0.4616	0.3485	0.3688	0.3676	5.6095	1.4600	13.7833	0.4376	29.9377	70.0776	52.1192	274.0746	327.3919	400.0254
KNJ (msec)	0.3289	0.5101	0.3548	0.3785	0.3567	5.4332	1.2777	13.6844	0.4705	30.6712	68.7191	50.9743	273.3747	327.5894	398.7826
Pollard-Rho (msec)	0.3265	0.4131	0.3288	0.3856	0.3816	5.4527	1.3101	13.6676	0.4984	30.3033	70.7733	52.4225	274.0978	337.5512	400.0443

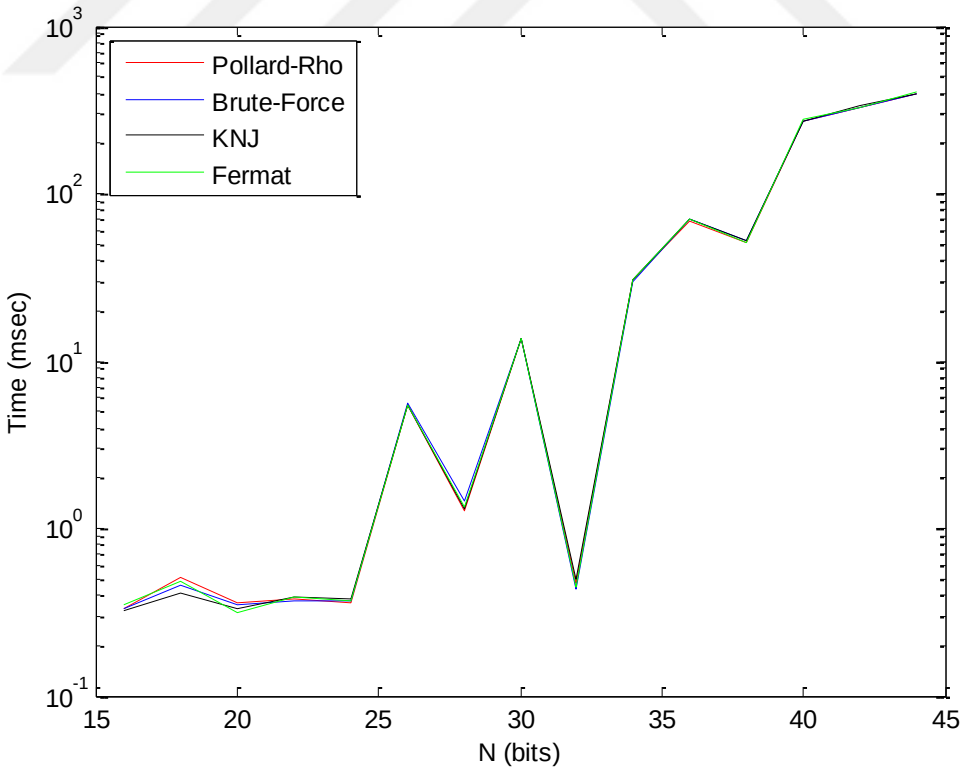

Figure 4.13 Elapsed factorization time values of 4 factorization methods

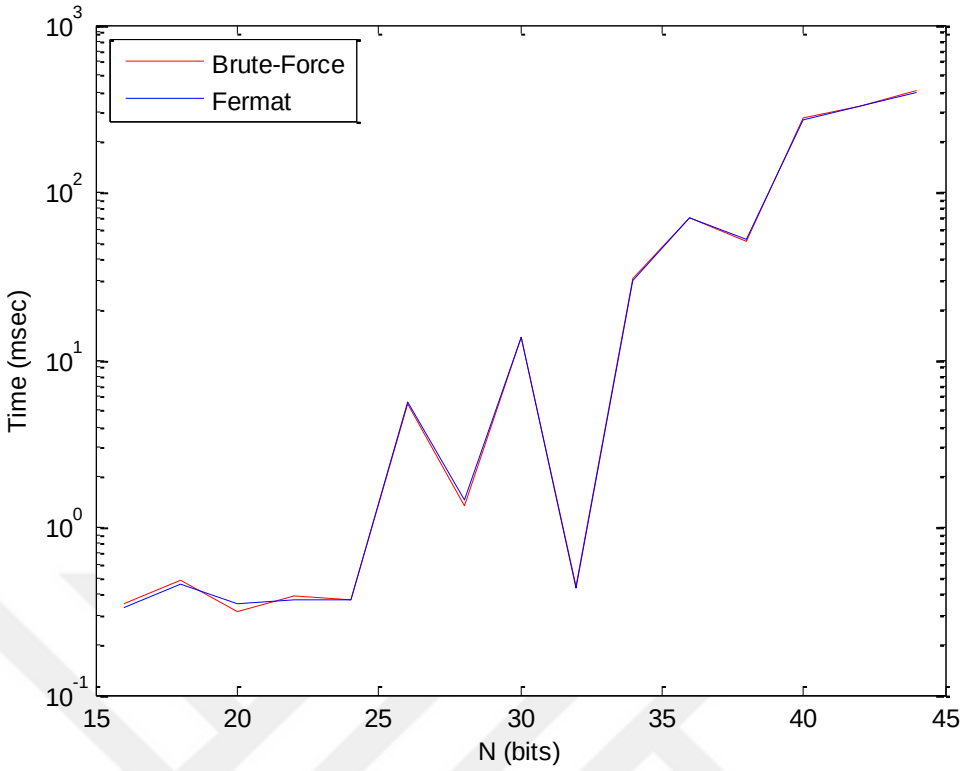

Figure 4.14 Elapsed factorization time values of Brute-Force vs Fermat

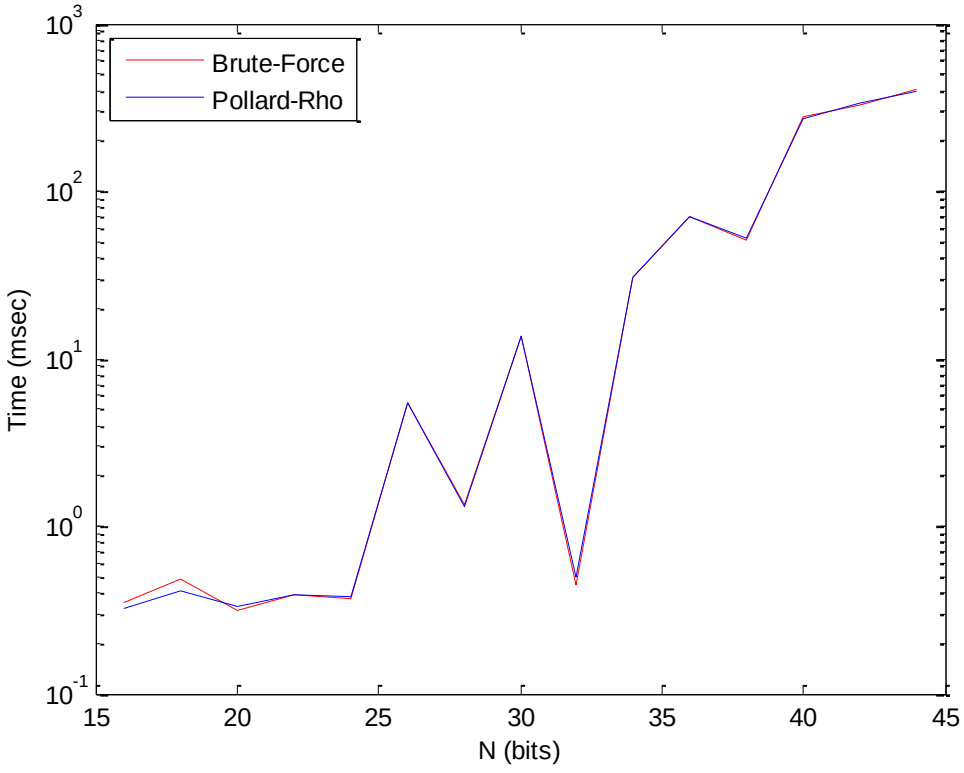

Figure 4.15 Elapsed factorization time values of Brute-Force vs Pollard-Rho

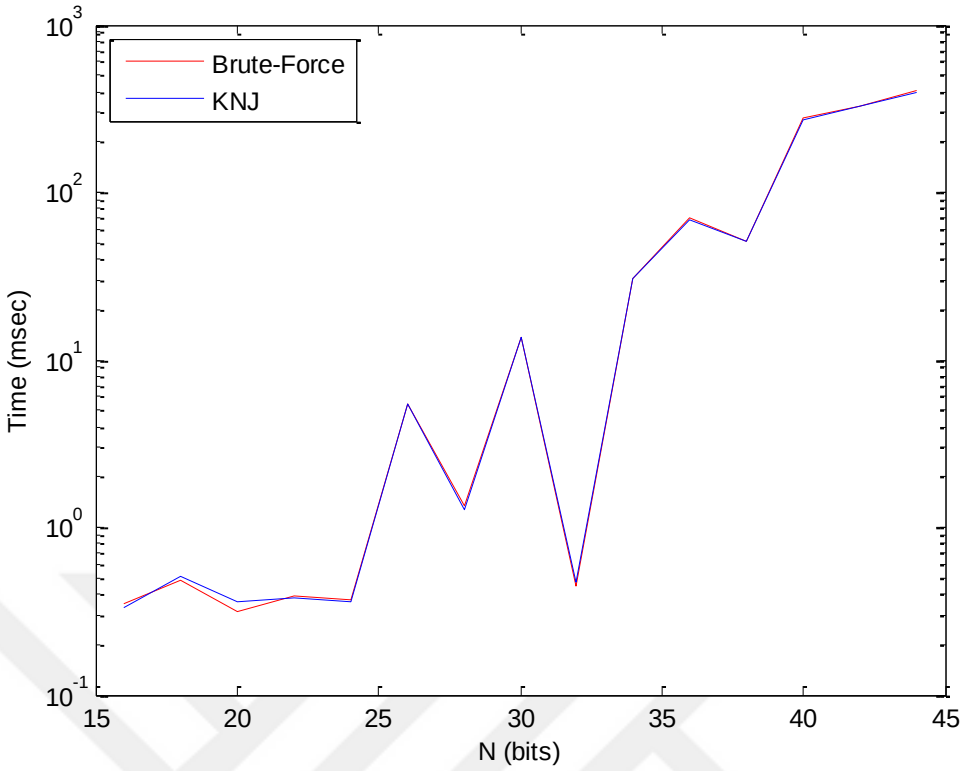

Figure 4.16 Elapsed factorization time values of Brute-Force vs KNJ

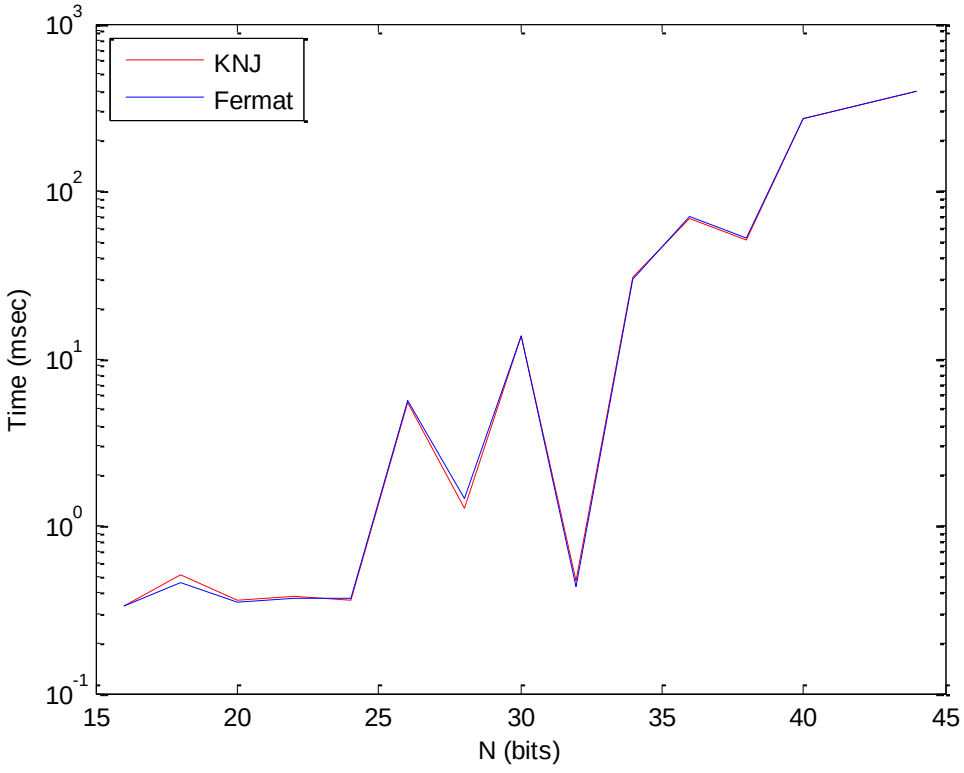

Figure 4.17 Elapsed factorization time values of KNJ vs Fermat

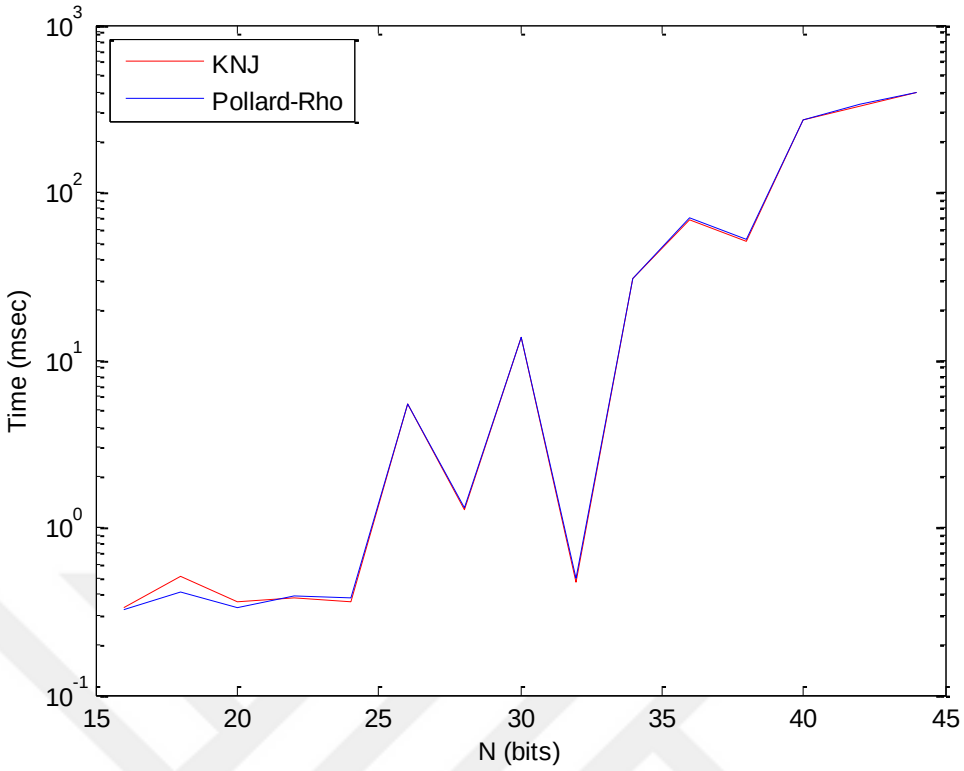

Figure 4.18 Elapsed factorization time values of KNJ vs Pollard-Rho

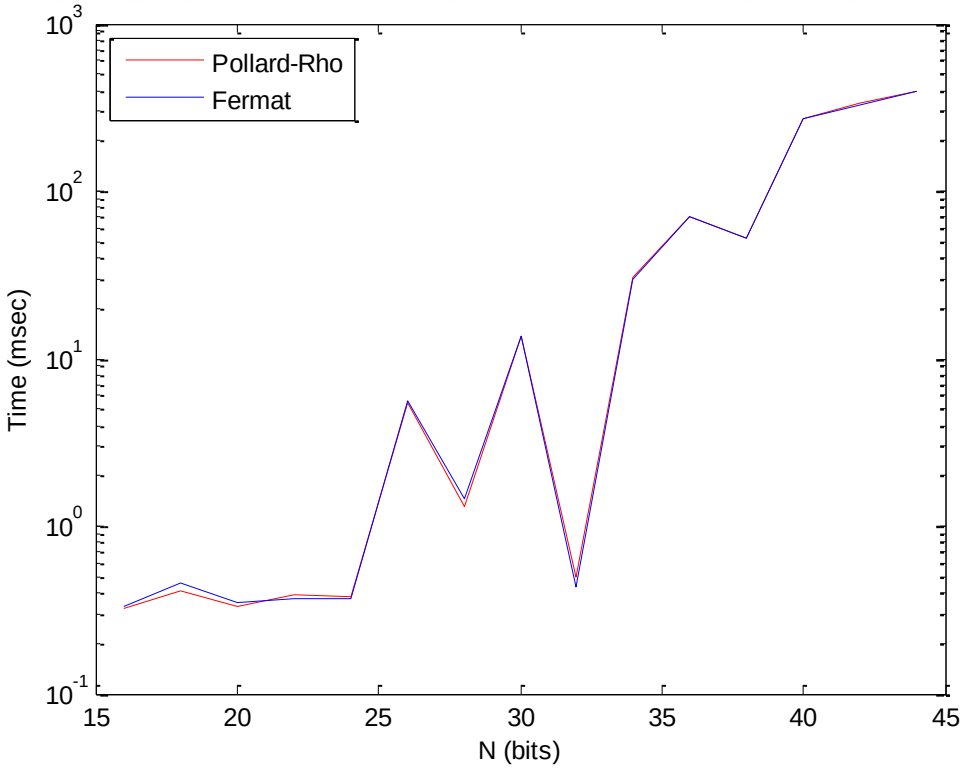

Figure 4.19 Elapsed factorization time values of Pollard-Rho vs Fermat

Although elapsed time values for Fermat is less than other factorization methods as seen in Figure 4.13, it is not easy to comment on other methods' elapsed time efficiency. Therefore, time values versus N values for varying N values were plotted in pairs from Figure 4.14 to Figure 4.19. In these figures, elapsed time values for all factorization methods are nearly identical except some small differences. The reasons for obtaining nearly identical plots for 4 different factorization methods can be listed as follows:

- In order to propose smoothness in the test architecture, hardware should not be effective parameter to compare the efficiency of factorization methods. However, in our experiment, we did not focus on this issue because of the fact that running time of any algorithm depends on more than one parameter. Thus, to compare efficiency of factorization methods, depending only on their hardware dependent elapsed time values does not seem possible.
- Since hardware problems causes to get nearly identical time plots for 4 factorization methods, comparing efficiency of factorization methods by iteration numbers, with efficiency of factorization methods by elapsed time values, does not gives exact and reliable results.
- While creating datasets for factorization methods, 1000 time values for each N, were saved. Although the number of sample time values seems sufficient, efficiency of this experiment could be increased by taking more sample time values for each method. Thus, the calculated mean values for each factorization method could be smoother.
- In this experiment, 15 different N values were factorized. The minimum N size was 16 bits and maximum N size was 44 bits. Elapsed time values were in millisecond level. Due to working with small N values and measuring elapsed times for factorization with pre-defined Java method, nanoTime(), comparing efficiency of factorization methods depending on their elapsed time values should not be sufficient.

5. SUMMARY and CONCLUSION

In this work, RSA Algorithm has been studied in detail and it is critically analyzed. This thesis presents on insight into the basics of cryptography and it explains RSA Algorithm, RSA's mathematical background and mathematical characteristics. The deficiency in key management process of symmetric-key algorithms has been examined clearly and the reason behind the necessity of invention of asymmetric-key cryptosystems was explained. With a simple numeric RSA example, the key generation, encryption and decryption processes of the algorithm are clearly indicated. The relation between Euler's Phi Function and prime numbers were simulated with a graph. The common RSA attacks and the well-known hints to avoid such RSA attacks were explained. By introducing Java classes for random number generation, the effect of random number generators on RSA applications was proved. The importance of SecureRandom class for obtaining true-random numbers was realized for Java. The special cases where plain-text cannot be hidden and remains equal to cipher-text were shown. The symmetry in distribution of such cases in a sample encryption scenario was examined carefully.

Simulation results for cryptography and cryptanalysis have been obtained using Eclipse and Java. From 4 bits to 2048 bits, RSA encryption without padding feature has been implemented. For 512 bits key length, RSA with padding feature was implemented. As a cryptanalysis, four different factorization attack algorithms which are Brute-Force Factorization, Pollard-Rho Factorization, Fermat Factorization and KNJ Factorization, were implemented in Java to indicate the importance of choosing RSA variables according to indicated RSA hints. Comparison between the iteration numbers of these attacks and elapsed time values of these attacks were made. The efficiency of these attacks in cryptanalysis of RSA was indicated.

With choosing RSA as a subject of master thesis, I learned many useful theorems like Euler's Theorem, Fermat's Little Theorem and Euclidean Theorem. I implemented them in Java. I studied RSA Algorithm in detail and I critically analyzed it. As a future work, proposing an effective factorization algorithm for RSA and implementing it in Java can be listed. Moreover, proposing a new algorithm which has less deficiency than RSA can be foreseen.

REFERENCES

[1] C. Kaufman, R. Perlman and M. Speciner, Network Security: Private Communication in a Public World, *The Second Edition*, Prentice Hall, **2002,** 41.

[2] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 3.

[3] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 3.

[4] J. Garms, D. Somerfield, Professional Java Security, *The First Edition*, Wrox Pres Ltd, **2001**, 416.

[5] C. Kaufman, R. Perlman and M. Speciner, Network Security: Private Communication in a Public World, *The Second Edition*, Prentice Hall, **2002**, 148.

[6] F. Kaderali, Foundation and Applications of Cryptology,**2007**, 59.

[7] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 188.

[8] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Crytopgraphy, *The First Edition,* CRC Press, **1996**, 65.

[9] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 157.

[10] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 166.

[11] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 166.

[12] B. A. Forouzan, Data Communications and Networking, *The Fourth Edition*, Mc Graw Hill Higher Education, **2007**, 932.

[13] D. R. Stinson, Cryptography Theory and Practice, *The Third Edition*, Chapman & Hall / CRC, **2003**, 1.

[14] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Crytopgraphy, *The First Edition,* CRC Press, **1996**, 32.

[15] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Crytopgraphy, *The First Edition,* CRC Press, **1996**, 31.

[16] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Crytopgraphy, *The First Edition,* CRC Press, **1996**, 32.

[17] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 151.

[18] R. Churchhouse, Codes and Ciphers Julius Ceaser, the Enigma and the Internet, *The First Edition,* Cambridge University Press, **2001**, 166.

[19] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 151.

[20] W. Stallings, Codes and Ciphers Julius Ceaser, the Enigma and the Internet, *The Fifth Edition,* Prentice Hall, **2011**, 15.

[21] B. A. Forouzan, Cryptography & Network Security, *The Special Indian Edition*, Mc Graw Hill Higher Education, **2007**, 306.

[22] B. A. Forouzan, Cryptography & Network Security, *The Special Indian Edition*, Mc Graw Hill Higher Education, **2007**, 308.

[23] B. A. Forouzan, Cryptography & Network Security, *The Special Indian Edition*, Mc Graw Hill Higher Education, **2007**, 309.

[24] B. A. Forouzan, Cryptography & Network Security, *The Special Indian Edition*, Mc Graw Hill Higher Education, **2007**, 308.

[25] C. Kaufman, R. Perlman and M. Speciner, Network Security: Private Communication in a Public World, *The Second Edition*, Prentice Hall, **2002,** 159.

[26] J. Garms, D. Somerfield, Professional Java Security, *The First Edition*, Wrox Pres Ltd, **2001**, 413.

[27] J. Garms, D. Somerfield, Professional Java Security, *The First Edition*, Wrox Pres Ltd, **2001**, 52.

[28] Butun I., Demirer M., A Blind Digital Signature Scheme Using Elliptic Curve Digital Signature Algorithm, *Turkish Journal of Electrical Engineering & Computer Sciences*, **2013**, 21, 945-956.

[29] F. Kaderali, Foundation and Applications of Cryptology,**2007**, 72.

[30] Oracle, http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

[31] J. Garms, D. Somerfield, Professional Java Security, *The First Edition*, Wrox Pres Ltd, **2001**, 430.

[32] C. Paar, J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, *The First Edition,* Springer, **2010**, 192.

[33] R. Helton, J. Helton, Java Security Solutions, *The First Edition,* Wiley, **2002**, 257.

[34] B. A. Forouzan, Cryptography & Network Security, *The Special Indian Edition*, Mc Graw Hill Higher Education, **2007**, 271.

[35] B. A. Forouzan, Cryptography & Network Security, *The Special Indian Edition*, Mc Graw Hill Higher Education, **2007**, 269.

[36] N. Lal., A. P. Singh., S. Kumar, Modified Trial Divison Algorithm Using KNJ-Factorization Method to Factorize RSA Public Key Encryption, *Wireless Communication and Computing Indian Institute of Information Technology.*

CURRICULUM VITAE

PERSONAL INFORMATION

Name and Surname : Mustafa KOCAKULAK

Birth Date and Place : 30.01.1989 - BURSA

Foreign Language : Advanced English, Basic German

E-mail : kocakulakmustafa@gmail.com

EDUCATIONAL STATUS

Degree	Department	University Name	Graduation
B.S.	Electrical & Electronics Engineering	Bilkent University	2012

WORK EXPERIENCE

Year	Firm/Corporation	Enrollment
2015-February	Bursa Technical University	Research Assistant
2014	Litera Bilişim Grup LTD.ŞTİ	Software Engineer
2012	E.R.P Yazılım Danışmanlık LTD.ŞTİ	Project Engineer
2011	A Bilgi Teknolojileri LTD.ŞTİ	Intern Engineering
2010	MAKO Elektrik A.Ş	Intern Engineering

AWARDS

1. Graduated with 1# degree, Namık Kemal İlköğretim Okulu (2002-2003)
2. Semi-Finalist of Massachusetts Institute of Technology MIT Enterprise Forum Turkey Innovation Competition (2012)
3. Finalist of " TÜSİAD Bu Gençlikte İş Var" Innovation Competition (2013)