RSA Analysis

MINOR PROJECT REPORT

Submitted in partial fulfillment of the Requirements

For the award of the degree of

Bachelor of Technology

In

Information Technology

Submitted By

Aarushi Singhal (00176803113)

Divya Batra (00476803113)

Simran Kaur (03376803113)

Jasmeet Kaur (03576803113)

Under the guidance of

Mr. Pradeep Gulati

GURU TEGH BAHADUR INSTITUTE OF TECHNOLOGY

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

DWARKA, NEW DELHI

YEAR 2016-2017

DECLARATION

This is to certify that project report entitled "**RSA Analysis"** which is submitted by Aarushi Singhal (001), Divya Batra (004), Simran Kaur (033), Jasmeet Kaur (035) in partial fulfillment of the requirement for the award of degree Bachelor of Technology, in Information Technology, Guru Tegh Bahadur Institute Of Technology, Rajouri Garden, New Delhi is an authentic record of our own work carried out under the guidance of Mr. **Pradeep Gulati.**

Aarushi Singhal (00176803113)

Divya Batra (00476803113)

Simran Kaur (03376803113)

Jasmeet Kaur (03576803113)

CERTIFICATE

This is to certify that this project report on RSA Analysis is submitted by Aarushi Singhal (001), Divya Batra (004), Simran Kaur (033) and Jasmeet Kaur (035), who have carried out the project work under my supervision.

I approved this project for submission of the Bachelor of Engineering in the department of Information Technology , faculty of Engineering affiliated to Guru Gobind Singh Indraprastha University, Delhi.

Ms. Gurpreet Kaur
H.O.D. (IT)

Mr. Pradeep Gulati
Project Mentor

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the Report of the B.Tech project undertaken during B.Tech 4th Year. We owe special thanks to **Mr. Pradeep Gulati** Department of Information Technology, Guru Tegh Bahadur Institute of Technology for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for all of us. We also take the opportunity to acknowledge him for his full support and assistance during the development of the project.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

ABSTRACT

RSA is the algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. The other key must be kept private. It is based on the fact that finding the factors of an integer is hard (the factoring problem). RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it in 1978. A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.

LIST OF FIGURES

LIST OF TABLES

Index

CHAPTER 1
INTRODUCTION

1.1 Overview

Security has, is and will be the most important issue, whether it's the security of important documents or safe sending of messages. As security is the major concern implementation of the security measures also has been constantly ameliorated per changing concerns. Earlier people used Cesar technique for sending messages then they come to data compression using hash functions, then on symmetric key exchange and finally now the rest on asymmetric means of security. By asymmetric we, can involvement of two keys. One is public key and the other is private key, this idea was also brought into use due to disadvantages came from symmetric encryption of the data and key exchange technique. The use of asymmetric encryption has sought out the problem of exchanging keys and thus has come out the most prominent technique used in security. One of such asymmetric key cryptography is "RSA".

Three scientist named as Ron Rivest, Adi Shamir and Len Adleman at MIT led to the invention of an algorithm named as RSA(last names of the scientist who invented it) in 1977and from then onward it has become the most widely accepted and implemented general purpose approach to public key encryption.

RSA is a block cipher in which plain text and cyphertext is between 0-n-1 for some value of n. A typical size for n is 1024 bits. RSA makes use of an expression with exponentials. A user of RSA creates and the publishes a public key based on 2 large prime numbers (p and q), along with auxiliary value. The prime numbers must be kept secret. RSA algorithm then includes key generation, key distribution. Encryption and decryption. Once p and q are chosen b algorithm then n is calculated.

N= p*q

RSA involves a public key and a private key. The public key can be known by everyone or they are sometimes provided by user on their website so that person can get access to their public key and use them to encrypt data to send to the user, but the intension is that

messages encrypted with public key can only be encrypted in reasonable amount of time using private key. Plaintext (text to be sent) is encrypted in blocks with each block having a binary value less than n, that is block size must be less than or equal to log(n)
C is the cipher text which is produced at sender's place using the public key of the person to whom the data is to be sent.

C= (M)^e mod(n) where , M is the plain text and e is the parameter constant at sender's side, it is chosen in such a way that 1<=e<= phi(n) , Where phi(n)is Euler's totient and is calculated by :

Phi(n)= (p-1)*(q-1), Where e is co-prime and phi(n) share no other factor as 1.

Gcd(e,phi(n))b, 'e' as generated at sender's place so, is public key.

At receiver's end, M= C^d mod(n) putting value of 'C' from '1'.'d' is the parameter calculated at receiver end.

M=M^ed mod(n), both sender and receiver knows the value of 'n' and senders knows value of 'e' and receiver knows the value of 'd' respectively.

Public key KV={e,n}

Private key KR={d,n}

The basic principle behind RSA is the observation that is practical to find very large positive number e,d and n such that with modular exponentiation for all 'm'.

m^ed=n mod(n)

Despite of knowing e or n or even m it can be extremely difficult to find 'd'.

m^ed=m mod(n)

m^ed= m^phi(n)+1=m^(p-1)*(q-1)+1

ed=phi(n)+1

ed=1mod(n)

d=e^-1 mod(n)

(d is the modular multiplicative inverse of e)

Example:-

1. p=17 ,q=11

2. n=17*11=187
3. phi(n)=(p-1)*(q-1)=(17-1)*(11-1)=160
4. Now generation of public key and private key.
5. Select e,such that relatively prime to phi(n)=160 and less than phi(n),we choose e=7.
6. Determine ‘d’ such that

   de=1mod (160)and d<160

   so, d*7=1mod(160)

   d=23*7 =161

   so value of d=23

   1=161mod(160)=1

   L.H.S=R.H.S

   Public key KV ={7,187}

   Private key KR={23,187}

   C=88^7 mod (187)

   =(88)^4 mod(187)*(88)^2 mod(187)*(88)^1mod(187)

   =11

   Similarly,

   M=(11)^23 mod(187)=88

1.2 Public Key Cryptography

• Public-key refers to a cryptographic mechanism. It has been named public-key to differentiate it from the traditional and more intuitive cryptographic mechanism known as: symmetric-key, shared secret, secret-key and also called private-key.

• It introduces a concept of involving key pairs: one for encrypting, the other for decrypting. This concept, is very clever and attractive, and provides a great deal of advantages over symmetric-key:

- Simplified key distribution
- Digital Signature
- Long-term encryption

• Encryption and decryption are carried out using two different keys. The two keys in such a key pair are referred to as the public key and the private key.

• Party A, if wanting to communicate confidentially with party B, can encrypt a message using B's publicly available key. Such a communication would only be decipherable by B as only B would have access to the corresponding private key.

• Party A, if wanting to send an authenticated message to party B, would encrypt the message with A's own private key. Since this message would only be decipherable with A's public key, that would establish the authenticity of the message — meaning that A was indeed the source of the message.

• Therefore, Public-key encryption can be used to provide both confidentiality and authentication at the same time. Note again that confidentiality means that we want to protect a message from eavesdroppers and authentication means that the recipient needs a guarantee as to the identity of the sender.

CHAPTER 2
RSA

2.1 RSA Algorithm

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. The RSA cryptosystem is the most widely-used public key cryptography algorithm in the world. It can be used to encrypt a message without the need to exchange a secret key separately.

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Party A can send an encrypted message to party B without any prior exchange of secret keys. A just uses B's public key to encrypt the message and B decrypts it using the private key, which only he knows. RSA can also be used to sign a message, so A can sign a message using their private key and B can verify it using A's public key.

A user wishing to exchange encrypted messages using a public-key cryptosystem would place their public encryption procedure, E, in a public file. The user's corresponding decryption procedure, D, is kept confidential. Rivest, Shamir, and Adleman provide four properties that the encryption and decryption procedures have:

- Deciphering the enciphered form of a message M yields M. That is, D(E(M)) = M
- E and D are easy to compute.
- Publicly revealing E does not reveal an easy way to compute D. As such, only the user can decrypt messages which were encrypted with E. Likewise, only the user can compute D efficiently.
- Deciphering a message M and then enciphering it results in M. That is, E(D(M)) = M

As Rivest, Shamir, and Adleman point out, if a procedure satisfying property (3) is used, it is extremely impractical for another user to try to decipher the message by trying all possible messages until they find one such that E(M) = C.

A function satisfying properties (1) - (3) is called a "trap-door one-way function". It is called "one-way" because it is easy to compute in one direction but not the other. It is called "trap-door" because the inverse functions are easy to compute once certain private, "trap-door" information is known.

2.2 RSA Procedure

With this under our belt, we are ready to describe the implementation of the RSA public-key cryptosystem.

To create his public and secret keys, a person uses the following procedure:

- Select two large prime numbers, p and q , typically of 100 digits each.
- Compute n = pq .
- Select a small odd integer e that is relatively prime to m = (p-1)(q-1) .
- Compute d, the multiplicative inverse of e modulo m .
- Publish the pair P(e,n) as his RSA public key.
- Keep secret the pair S(d,n) as his RSA secret key.

For this scheme, the domain D is Zn = {0,1,...n-1}. That is, the message M must be such that M belongs to Zn .The transformation of a message M associated with a public key P(e,n).

CHAPTER 3

RSA APPLICATIONS

3.1 APPLICATIONS

3.1.1 Often used to encrypt a symmetric key.

1. Browser requests a page.
2. Server sends its public key with the certificate.
3. Browser checks the certificate , generate random key k1 . Encrypt URL and https data with k1. encrypt k1 with public key and sends to the server.
4. Server decrypt k1 with private key and uses k1 to decrypt URL and https data.
5. Server sends back requested http data encrypted with k1.
6. Browser decrypts and displays.

3.1.2 Digital Signatures

Digital signatures are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature, but not all electronic signatures use digital signatures. In some countries, including the United States, India, Brazil, Indonesia, Saudi Arabia, Switzerland and the countries of the European Union, electronic signatures have legal significance.

Digital signatures employ asymmetric cryptography. In many instances they provide a layer of validation and security to messages sent through a nonsecure channel: Properly implemented, a digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital seals and signatures are equivalent to handwritten signatures and stamped seals. Digital signatures are equivalent to traditional handwritten signatures in many respects, but properly implemented digital signatures are more difficult to forge than the handwritten type. Digital signature schemes, in the sense used here, are cryptographically based, and must be implemented properly to be effective. Digital signatures can also provide non-repudiation, meaning that the signer

cannot successfully claim they did not sign a message, while also claiming their private key remains secret; further, some non-repudiation schemes offer a time stamp for the digital signature, so that even if the private key is exposed, the signature is valid. Digitally signed messages may be anything representable as a bitstring: examples include electronic mail, contracts, or a message sent via some other cryptographic protocol.

What is a **Digital Signature**

Digital signature scheme typically consists of three algorithms:

- A *key generation* algorithm that selects a *private key* uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding *public key*.
- A *signing* algorithm that, given a message and a private key, produces a signature.
- A *signature verifying* algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

Two main properties are required. First, the authenticity of a signature generated from a fixed message and fixed private key can be verified by using the corresponding public key. Secondly, it should be computationally infeasible to generate a valid signature for a party without knowing that party's private key. A digital signature is an authentication mechanism that enables the creator of the message to attach a code that acts as a signature.

In the following discussion, 1^n refers to a unary number.

Formally, a digital signature scheme is a triple of probabilistic polynomial time algorithms, (G, S, V), satisfying:

- *G* (key-generator) generates a public key, *pk*, and a corresponding private key, *sk*, on input 1^n, where *n* is the security parameter.
- *S* (signing) returns a tag, *t*, on the inputs: the private key, *sk*, and a string, *x*.
- *V* (verifying) outputs *accepted* or *rejected* on the inputs: the public key, *pk*, a string, *x*, and a tag, *t*.

For correctness, *S* and *V* must satisfy

$\Pr[\ (pk, sk) \leftarrow G(1^n), V(\ pk, x, S(sk, x)\ ) = accepted\ ] = 1.$[8]

A digital signature scheme is secure if for every non-uniform probabilistic polynomial time adversary, *A*

$\Pr[\ (pk, sk) \leftarrow G(1^n), (x, t) \leftarrow A^{S(sk, \cdot)}(pk, 1^n), x \notin Q, V(pk, x, t) = accepted] < \underline{\text{negl}}(n)$,where $A^{S(sk, \cdot)}$ denotes that *A* has access to the oracle, $S(sk, \cdot)$, and *Q* denotes the set of the queries on *S* made by *A*, which knows the public key, *pk*, and the security parameter, *n*. Note that we require any adversary cannot directly query the string, *x*, on *S*.

3.1.3 Hash Functions

Hash functions are extremely useful and appear in almost all information security applications.

A hash function is a mathematical function that converts a numerical input value into another compressed numerical value. The input to the hash function is of arbitrary length but output is always of fixed length.

Values returned by a hash function are called **message digest** or simply **hash values**. The following picture illustrated hash function –

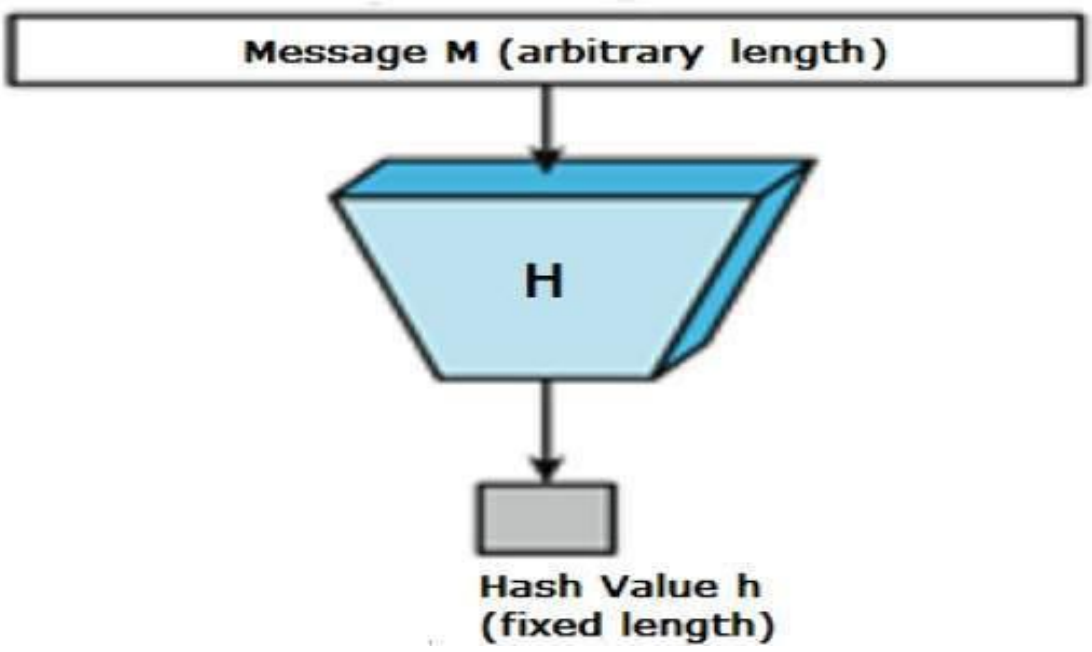


Fig 3.1 Hash Diagram

The typical features of hash functions are : −

- **Fixed Length Output (Hash Value)**
    - Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as **hashing the data**.
    - In general, the hash is much smaller than the input data, hence hash functions are sometimes called **compression functions**.
    - Since a hash is a smaller representation of a larger data, it is also referred to as a **digest**.
    - Hash function with n bit output is referred to as an **n-bit hash function**. Popular hash functions generate values between 160 and 512 bits.

- **Efficiency of Operation**
    - Generally for any hash function h with input x, computation of h(x) is a fast operation.
    - Computationally hash functions are much faster than a symmetric encryption.

Properties of Hash Functions

In order to be an effective cryptographic tool, the hash function is desired to possess following properties −

- **Pre-Image Resistance**
    - This property means that it should be computationally hard to reverse a hash function.
    - In other words, if a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
    - This property protects against an attacker who only has a hash value and is trying to find the input.
- **Second Pre-Image Resistance**
    - This property means given an input and its hash, it should be hard to find a different input with the same hash.
    - In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
    - This property of hash function protects against an attacker who has an input value and its hash, and wants to substitute different value as legitimate value in place of original input value.

- **Collision Resistance**

    - This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function.

    - In other words, for a hash function h, it is hard to find any two different inputs x and y such that h(x) = h(y).

    - Since, hash function is compressing function with fixed hash length, it is impossible for a hash function not to have collisions. This property of collision free only confirms that these collisions should be hard to find.

    - This property makes it very difficult for an attacker to find two input values with the same hash.

    - Also, if a hash function is collision-resistant **then it is second pre-image resistant.**

3.1.4 Design of Hashing Algorithms

At the heart of a hashing is a mathematical function that operates on two fixed-size blocks of data to create a hash code. This hash function forms the part of the hashing algorithm.

The size of each data block varies depending on the algorithm. Typically the block sizes are from 128 bits to 512 bits. The following illustration demonstrates hash function -

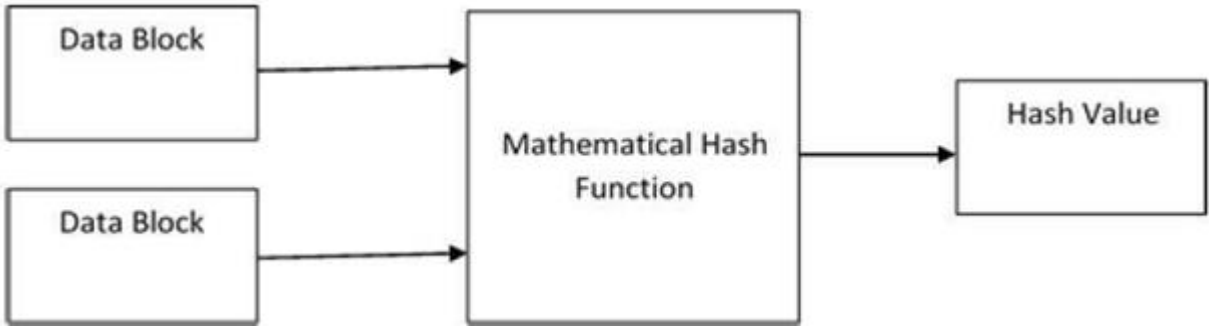


Fig 3.2 Mathematical Hash Function

Twelve Hashing algorithm involves rounds of above hash function like a block cipher. Each round takes an input of a fixed size, typically a combination of the most recent message block and the output of the last round.

This process is repeated for as many rounds as are required to hash the entire message. Schematic of hashing algorithm is depicted in the following illustration −

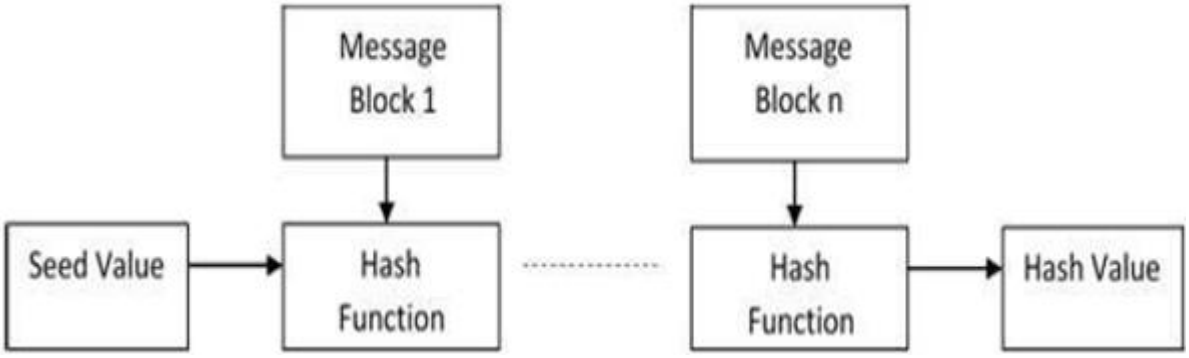


Fig 3.3 Hash Procedure

Since, the hash value of first message block becomes an input to the second hash operation, output of which alters the result of the third operation, and so on. This effect, known as an **avalanche** effect of hashing. Avalanche effect results in substantially different hash values for two messages that differ by even a single bit of data. Understand the difference between hash function and algorithm correctly. The hash function generates a hash code by operating on two blocks of fixed-length binary data. Hashing algorithm is a process for using the hash function, specifying how the

message will be broken up and how the results from previous message blocks are chained together.

3.1.5 Popular Hash Functions

Let us briefly see some popular hash functions −

1. Message Digest (MD)

MD5 was most popular and widely used hash function for quite some years.

- The MD family comprises of hash functions MD2, MD4, MD5 and MD6. It was adopted as Internet Standard RFC 1321. It is a 128-bit hash function.

- MD5 digests have been widely used in the software world to provide assurance about integrity of transferred file. For example, file servers often provide a pre-computed MD5 checksum for the files, so that a user can compare the checksum of the downloaded file to it.

- In 2004, collisions were found in MD5. An analytical attack was reported to be successful only in an hour by using computer cluster. This collision attack resulted in compromised MD5 and hence it is no longer recommended for use.

2. Secure Hash Function (SHA)

Family of SHA comprise of four SHA algorithms; SHA-0, SHA-1, SHA-2, and SHA-3. Though from same family, there are structurally different.

- The original version is SHA-0, a 160-bit hash function, was published by the National Institute of Standards and Technology (NIST) in 1993. It had few weaknesses and did not become very popular. Later in 1995, SHA-1 was designed to correct alleged weaknesses of SHA-0.

- SHA-1 is the most widely used of the existing SHA hash functions. It is employed in several widely used applications and protocols including Secure Socket Layer (SSL) security.

- In 2005, a method was found for uncovering collisions for SHA-1 within practical time frame making long-term employability of SHA-1 doubtful.

- SHA-2 family has four further SHA variants, SHA-224, SHA-256, SHA-384, and SHA-512 depending up on number of bits in their hash value. No successful attacks have yet been reported on SHA-2 hash function.

- Though SHA-2 is a strong hash function. Though significantly different, its basic design is still follows design of SHA-1. Hence, NIST called for new competitive hash function designs.

- In October 2012, the NIST chose the Keccak algorithm as the new SHA-3 standard. Keccak offers many benefits, such as efficient performance and good resistance for attacks.

CHAPTER 4
ATTACKS

4.1 Attacks on RSA

1. Brute Force Attack

In cryptography, a **brute-force attack** consists of an attacker trying many passwords or passphrases with the hope of eventually guessing correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found. Alternatively, the attacker can attempt to guess the key which is typically created from the password using a key derivation function. This is known as an **exhaustive key search**.

A brute-force attack is a cryptanalytic attack that can, in theory, be used to attempt to decrypt any encrypted data (except for data encrypted in an information-theoretically secure manner). Such an attack might be used when it is not possible to take advantage of other weaknesses in an encryption system (if any exist) that would make the task easier.

When password guessing, this method is very fast when used to check all short passwords, but for longer passwords other methods such as the dictionary attack are used because a brute-force search takes too long. Longer passwords, passphrases and keys have more possible values, making them exponentially more difficult to crack than shorter ones.

Brute-force attacks can be made less effective by obfuscating the data to be encoded making it more difficult for an attacker to recognize when the code has been cracked or by making the attacker do more work to test each guess. One of the measures of the strength of an encryption system is how long it would theoretically take an attacker to mount a successful brute-force attack against it.

Brute-force attacks are an application of brute-force search, the general problem-solving technique of enumerating all candidates and checking each one.

4.2 Basic concept

Brute force attacks work by calculating every possible combination that could make up a password and testing it to see if it is the correct password. As the password's length increases, the amount of time, on average, to find the correct password increases exponentially. This means short passwords can usually be discovered quite quickly, but longer passwords may take decades.

4.3 Timing attack

In cryptography, a **timing attack** is a side channel attack in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms. Every logical operation in a computer takes time to execute, and the time can differ based on the input; with precise measurements of the time for each operation, an attacker can work backwards to the input.

Information can leak from a system through measurement of the time it takes to respond to certain queries. How much such information can help an attacker depends on many variables: crypto system design, the CPU running the system, the algorithms used, assorted implementation details, timing attack countermeasures, the accuracy of the timing measurements, etc.

Timing attacks are often overlooked in the design phase because they are so dependent on the implementation

A timing attack is an example of an attack that exploits the data-dependent behavioral characteristics of the implementation of an algorithm rather than the mathematical properties of the algorithm itself.

Many cryptographic algorithms can be implemented (or masked by a proxy) in a way

that reduces or eliminates data dependent timing information: consider an implementation in which every call to a subroutine always returns in exactly x seconds, where x is the maximum time it ever takes to execute that routine on every possible

authorized input. In such an implementation, the timing of the algorithm leaks no information about the data supplied to that invocation. The downside of this approach is that the time to execute many invocations increases from the average performance of the function to the worst-case performance of the function.

Timing attacks are practical in many cases:

- Timing attacks can be applied to any algorithm that has data-dependent timing variation. Software run on a CPU with a data cache will exhibit data-dependent timing variations as a result of memory looks into the cache. Some operations, such as multiplication, may have varied execution time depending on the inputs. Removing timing-dependencies is difficult in some algorithms that use low-level operations that frequently exhibit varied execution time.
- Finding secrets through timing information may be significantly easier than using cryptanalysis of known plaintext, ciphertext pairs. Sometimes timing information is combined with cryptanalysis to improve the rate of information leakage.

Example

The execution time for the square-and-multiply algorithm used in modular exponentiation depends linearly on the number of '1' bits in the key. While the number of '1' bits alone is not nearly enough information to make finding the key trivially easy, repeated executions with the same key and different inputs can be used to perform statistical correlation analysis of timing information to recover the key completely, even by a passive attacker. Observed timing measurements often include noise (from such sources as network latency, or disk drive access differences from access to access, and the error correction techniques used to recover from transmission errors). Nevertheless, timing attacks are practical against a number of encryption algorithms, including RSA, ElGamal, and the Digital Signature Algorithm.

In 2003, Boneh and Brumley demonstrated a practical network-based timing attack on SSL-enabled web servers, based on a different vulnerability having to do with the use of RSA with Chinese remainder theorem optimizations. The actual network distance was small in their experiments, but the attack successfully recovered a server private key in a matter of hours. This demonstration led to the widespread deployment and use of blinding techniques in SSL implementations. In this context, blinding is intended to remove correlations between key and encryption time.

Some versions of Unix use a relatively expensive implementation of the *crypt* library function for hashing an 8-character password into an 11-character string. On older hardware, this computation took a deliberately and measurably long time: as much as two or three seconds in some cases. The *login* program in early versions of Unix executed the crypt function only when the login name was recognized by the system. This leaked information through timing about the validity of the login name, even when the password was incorrect. An attacker could exploit such leaks by first applying brute-force to produce a list of login names known to be valid, then attempt to gain access by combining only these names with a large set of passwords known to be frequently used. Without any information on the validity of login names the time needed to execute such an approach would increase by orders of magnitude, effectively rendering it useless. Later versions of Unix have fixed this leak by always executing the crypt function, regardless of login name validity.

Two otherwise securely isolated processes running on a single system with either cache memory or virtual memory can communicate by deliberately causing page faults and/or cache misses in one process, then monitoring the resulting changes in access times from the other. Likewise, if an application is trusted, but its paging/caching is affected by

branching logic, it may be possible for a second application to determine the values of the data compared to the branch condition by monitoring access time changes; in extreme examples, this can allow recovery of cryptographic key bits.

It is an attack in which attacker gets access to the private information of the user without actually getting into the code.

It was first discovered by Kocher in 1996 which amazed even the developers of RSA. He was able to decode the text which was having small bit keys and created threat to RSA security.

To encounter this effect CRT was introduced CRT stands for Chinese remainder theorem which tries to minimize the affect of Kochers security threat. According to this theorem the text was divided into multiple block of cipher text with each of them having different RSA implementation. But this was also encountered, and still the RSA is improving.

CHAPTER 5
SECURITY SERVICES OF RSA

5.1 SECURITY SERVICES OF CRYPTOGRAPHY

The primary objective of using cryptography is to provide the following four fundamental information security services. Let us now see the possible goals intended to be fulfilled by cryptography.

1. Confidentiality

Confidentiality is the fundamental security service provided by cryptography. It is a security service that keeps the information from an unauthorized person. It is sometimes referred to as **privacy** or **secrecy**.

Confidentiality can be achieved through numerous means starting from physical securing to the use of mathematical algorithms for data encryption.

2. Data Integrity

It is security service that deals with identifying any alteration to the data. The data may get modified by an unauthorized entity intentionally or accidently. Integrity service confirms that whether data is intact or not since it was last created, transmitted, or stored by an authorized user.

Data integrity cannot prevent the alteration of data, but provides a means for detecting whether data has been manipulated in an unauthorized manner.

3. Authentication

Authentication provides the identification of the originator. It confirms to the receiver that the data received has been sent only by an identified and verified sender.

Authentication service has two variants −

- **Message authentication** identifies the originator of the message without any

regard router or system that has sent the message.

- **Entity authentication** is assurance that data has been received from a specific entity, say a particular website.

Apart from the originator, authentication may also provide assurance about other parameters related to data such as the date and time of creation/transmission.

4. Non-repudiation

It is a security service that ensures that an entity cannot refuse the ownership of a previous commitment or an action. It is an assurance that the original creator of the data cannot deny the creation or transmission of the said data to a recipient or third party.

Non-repudiation is a property that is most desirable in situations where there are chances of a dispute over the exchange of data. For example, once an order is placed electronically, a purchaser cannot deny the purchase order, if non-repudiation service was enabled in this transaction.

5.2 LIMITATIONS

1. Using small prime numbers.

   Small values of p, q and n makes the algorithm less secure and easy to crack with the help of Brute force algorithm. Guessing the values of p and q becomes easy when the length of key is short. Therefore to make a system secure, it should at least consist of keys with length 1024 bits.

2. Using prime numbers that are very close.

   Taking prime numbers that are near each other makes the guessing easy and ultimately cracking the private key.

3. Public key system depends on the use of some sort of invertible mathematical functions, for e.g. :- invertible functions like trapdoor functions.

4. Key size is kept large, but it degrades the performance.

   Two people using the same N, receiving the same message.

5. Suppose Alice and Bob use the same N and so have public keys of the form (e1, N) and (e2, N). If e1 and e2 are co-prime then Jane is able to read any message m that is sent to both of them. By number theory, e1, e2 co-prime guarantees the existence of integers a, b such that ae1 + be2 = 1. Jane knows the values of e1, e2 so can use Euclids algorithm to find such a, b.

6. A primary application is for choosing the key length of the RSA public-key encryption scheme. Progress in this challenge should give an insight into which key sizes are still safe and for how long.

CHAPTER 6
CONCLUSIONS

6.1 CONCLUSIONS

Brute force attack:

Brute force attack is inspired from number of successful hit and trials for breaking encryption code and thus includes calculation of all possible number combination of keys that could help in deciphering code thus, RSA this research paper concentrates on one of the limitation of RSA that is selection of prime numbers which are near to each other.

For RSA to be successful it is very important that the prime numbers should be chosen that even if processing speed is also enhances then also determination of prime numbers become very difficult and thus here choosing function

BigInteger.probablePrime() has given enhanced security of RSA code as it is randomly choosing prime number from the sample of very large number and thus the product of two large prime numbers is also very large which becomes very difficult for an attacker to break in polynomial solvable time, hence security is enhanced.

Increasing the key length has increased number of possible combinations of key estimation and thus possible estimation of value of key is decreased, hence security is enhanced.

With randomly choosing of very large prime numbers makes product of prime numbers (n) also very large, which is very hard to decipher even if attacker knows the value of ‘n’.

6.2 Scope for Future Work

The recent developments in the domain of wireless communications and pervasive computing incite the different business operators to deploy multicast and broadcast based applications combining video, voice and text (such as video conferencing, interactive group games, news and stock quotes feeding, etc.,) over wireless devices equipped with more and more powerful processors (PDAs, laptops, cell phones, etc.,) and in environments without infrastructures (Ad-hoc Networks). Besides, the recent advances in the Micro-Electro-Mechanical technologies (MEMS) allowed the development of micro-components that combine sensor capacities and wireless communication facilities into the same circuit, with reduced dimensions and reasonable cost. However, the large scale deployment of this kind of applications cannot be achieved without securing the multicast model over this type of wireless networks. Many key problems inherent to the distributed nature of multicasting and to the resource constraints in sensor equipments and in wireless devices, in general, make securing group communications over this type of networks more difficult.

Among other problems one can cite: Efficient security mechanisms for resource constrained devices: Most of the proposed solutions in the literature for securing. group communications require high computation and storage capacities. However, the devices used in ad hoc and sensor networks, such as PDA s, cell phones, etc., have relatively limited resources in terms of computation and storage. Appropriate mechanisms have to be developed to assure some security levels with lower resource requirements.

Security context and mobility impact: Contrary to static environments, mobility in ad hoc networks can have important impacts on 130 security efficiency and performance. Moreover, mobile nodes establish a certain security context that they want to maintain after their movements without having to negotiate it again. Tests have to be conducted to tackle mobility issues relating to group communication security.

Absence of third trust party: Existing solutions suppose in general, the existence of third trust entities responsible for authentication, authorization and access control.

However, ad hoc and sensor networks are without infrastructures. Therefore, it is difficult to assume the existence of such entities. Thus, it is necessary to develop new trust models suitable for networks without infrastructure in hostile environments such as battles and devasted areas following natural disasters (earthquake, Tsunami, etc.)

Masking heterogeneity: Next generation networks consist not only collaborative autonomous systems, but also heterogeneous technologies that aim to provide the same service for users connected using different sorts of terminals. The challenge is then, how to design portable security mechanisms for these different technologies in order to guarantee safety and dependability for end-users in a seamless way. As application avenues are kept wide open, there is a lot of scope for improving the performance of the multicast MANET. As of today there is no single security enhancing algorithm, that is best suited for all applications which optimizes all the parameters viz: computation complexity, storage capacity, computation overhead etc.

Future scope should be in RSA cryptography to comparison of another cryptography algorithm such symmetric and asymmetric. Develop another algorithm merge two algorithm which provide more security. The security architecture and design domain dissected several important areas: establishing isolation management within shared technologies; designing architectures for meeting customer demands for service and availability; and certifying and accrediting systems before use, while leveraging federal solutions. The application security domain addressed exploitation and countermeasures to protect insecure interfaces. It provided methods on increasing security for PaaS, SaaS, and IaaS in the realm of message communication, information handling, key management, SDLC, tools and services, metrics, economics, and inter-host communication. This paper presents the implementation of RSA through an encryption and decryption procedures, which are readily available for commercial use.

Experiments were conducted on different text sizes. The results obtained in encryption and decryptions of RSA were given in seconds.

CHAPTER 7
SOURCE CODE, OBERVATIONS AND DESCRIPTIONS

SOURCE CODE (in JAVA)

```
package rsacopy;
import java.math.BigInteger;
import java.util.Random;
import java.util.Scanner;
public class Rsacopy {
    public static void main(String[] args) {
        BigInteger p;
    BigInteger q;
    BigInteger k;
    BigInteger phin, d;
    BigInteger e;
    BigInteger cip;
    System.out.println("Enter your msg to be encrypted!");
    Scanner in =new Scanner(System.in);
    BigInteger msg=in.nextBigInteger();
    long stime=System.nanoTime();
    Random rnd = new Random();
    p=BigInteger.probablePrime(1024, rnd);
    q=BigInteger.probablePrime(1024, rnd);
    k=p.multiply(q);
    phin=(p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
    e=BigInteger.valueOf(65537);
    d =
e.modInverse((p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE)));
    cip=msg.modPow(e, k);
    long etime=System.nanoTime()-stime;
    System.out.println(cip);
```

```
    System.out.println("Time for encryption : "+etime);
    long dstime=System.nanoTime();
    msg=cip.modPow(d, k);
    long detime=System.nanoTime()-dstime;
    System.out.println(msg);
    System.out.println("Time for decryption : "+detime);
    }
}
```

OBSERVATIONS

Table 1.

S.No.	Key Length (bits)	Encrypting Time (seconds)	Decyrpting Time (seconds)	Size of Message (kilobyte)	Size of Prime number (bits) p	Size of Prime number (bits) Q
1.	1024	1.31	0.16	1 KB	512	512
2.	1024	1.26	0.174	2 KB	512	512
3.	1024	1.23	0.174	4 KB	512	512
4.	1024	1.84	0.174	8 KB	512	512
5.	1024	1.98	0.170	16 kB	512	512

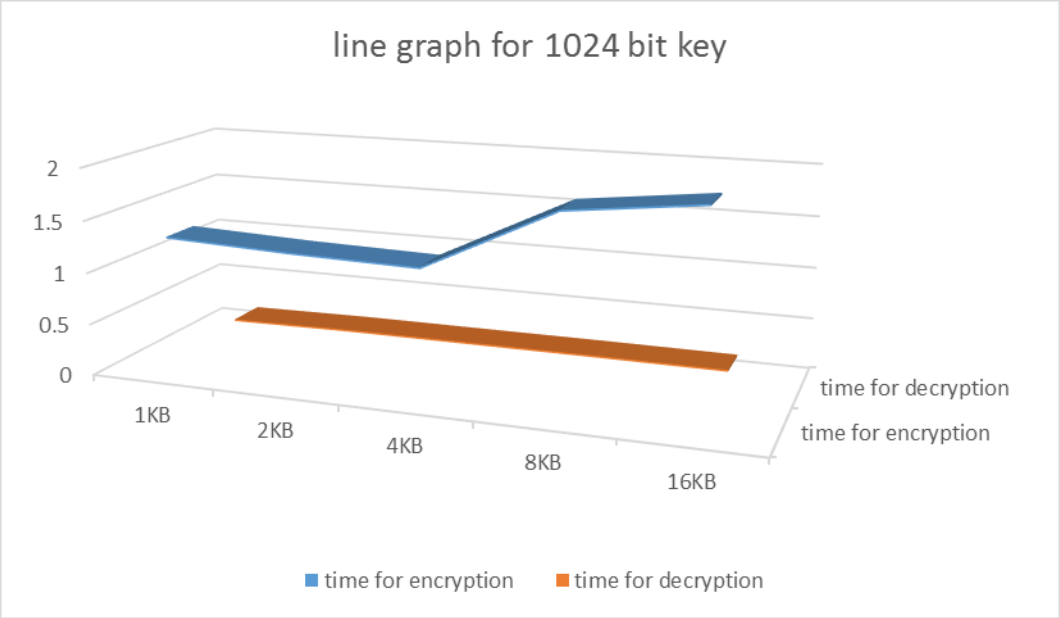
Table 7.1 Observation table of 1024 bits key

Table 2.

S.No.	Key Length	Encrypting Time	Decyrpting Time	Size of Message	Size of numbers p	Prime in bits Q
1.	2048	0.57	0.12	1 KB	1024	1024
2.	2048	0.492	0.132	2 KB	1024	1024
3.	2048	2.311	1.17	4 KB	1024	1024
4.	2048	1.46	0.12	8 KB	1024	1024
5.	2048	2.46	1.2	16 kB	1024	1024

Table 7.2 Observation table of 2048 bits key

Description

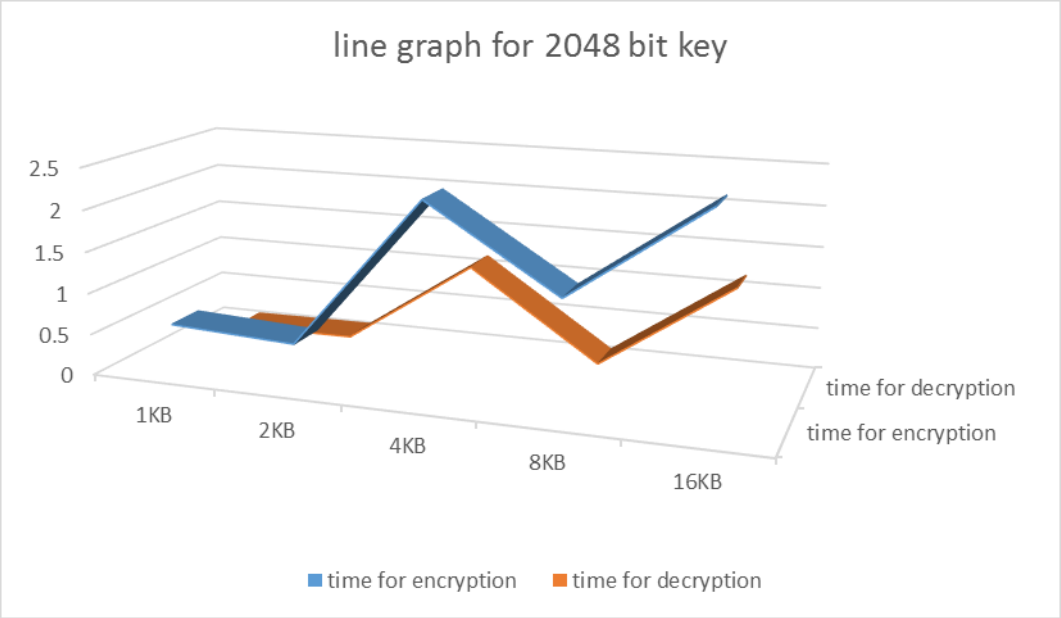


Graph (7.a) Time for encryption & decryption (Time versus Data size)

(1024-bit key length)

This is the graph which shows the description and relation between time of encryption and time for decryption of the message or file of different sizes.

With the graph we can say that with increase in file or message size time of decryption usually coming out to be constant without much variation in the values.



Graph (7.b) Time for encryption & decryption (Time versus Data size)

(2048-bit key length)

Description:

This graph shows that as the file size increases so the time of encryption increases and as the time of encryption increases, time of decryption also increases thus giving a somewhat similar linear equation as that of time of encryption.

This graph shows both the time encryption as well as decryption is directly proportional to each other.

RSA SOURCE CODE

C program to implement RSA algorithm. The given program will Encrypt and Decrypt a message using RSA Algorithm.

```
#include<stdio.h>     // used iostream.h for input and output
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

long int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
void main()
{
clrscr();
printf("\n ENTER FIRST PRIME NUMBER\n");
scanf("%d",&p);
flag=prime(p);
if(flag==0)
{
    printf("\n WRONG INPUT\n");
    getch();
    exit(1);
```

```
}

printf("\n ENTER ANOTHER PRIME NUMBER\n");
scanf("%d",&q);
flag=prime(q);
if(flag==0||p==q)
{
    printf("\n WRONG INPUT\n");
    getch();
    exit(1);
}
printf("\n ENTER MESSAGE\n");
fflush(stdin);   // used for the buffer storage
scanf("%s",msg);
for(i=0; msg[i]!=NULL;i++)
m[i]=msg[i];
n=p*q;
t=(p-1)*(q-1);
ce();
printf("\n POSSIBLE VALUES OF e AND d ARE\n");
for(i=0; i<j-1; i++)
printf("\n%ld\t%ld",e[i],d[i]);
encrypt();
decrypt();
getch();
}
int prime(long int pr)
{
int i;
J = sqrt(pr);   //using math.h header file
```

```
for(i=2; i<=j; i++)
{
    if(pr%i==0)
    return 0;
}
return 1;
}
void ce()
{
int k;
k=0;
for(i=2; i<t; i++)
{
    if(t%i==0)
    continue;
    flag=prime(i);
    if(flag==1 && i!=p && i!=q)
    {
        e[k]=i;
        flag=cd(e[k]);
        if(flag>0)
        {
            d[k] = flag;
            k++;
        }
        if(k == 99)
        break;   //used to exit loop*
    }
}
}
```

```
long int cd(long int x)
{
long int k=1;
while(1)    // it checks the parameter as 1 required to execute the loop
{
    k=k+t;   // use the shorthand ie : k + = t;
    if(k%x==0)
    return(k/x);
}
}
void encrypt()
{
long int pt,ct,key=e[0],k, len;
i=0;
len=strlen(msg);    //calculates the length of the message and returns a integer number
while(i!=len)
{
    pt=m[i];
    pt=pt-96;   // use shorthand ie : pt - = 96
    k=1;
    for(j=0; j<key; j++)
    {
        k=k*pt;      // use shorthand ie: k * = pt;
        k=k%n;    // leaves remainder   OR   use shorthand
    }
    temp[i]=k;
    ct=k+96;
    en[i]=ct;
    i++;    // use prefix in place of postfix
}
```

```
en[i]=-1;
printf("\nTHE ENCRYPTED MESSAGE IS\n");
for(i=0;en[i]!=-1;i++)
printf("%c",en[i]);
}
void decrypt()
{
long int pt,ct,key=d[0],k;
i=0;
while(en[i]!=-1)
{
    ct=temp[i];
    k=1;
    for(j=0;j<key;j++)
    {
        k=k*ct; s
        k=k%n;
    }
    pt =k+96;
    m[i]=pt;
    i++;
}
m[i]=-1;
printf("\nTHE DECRYPTED MESSAGE IS\n");
for(i=0;m[i]!=-1;i++)
printf("%c",m[i]);
}
```

Table 3

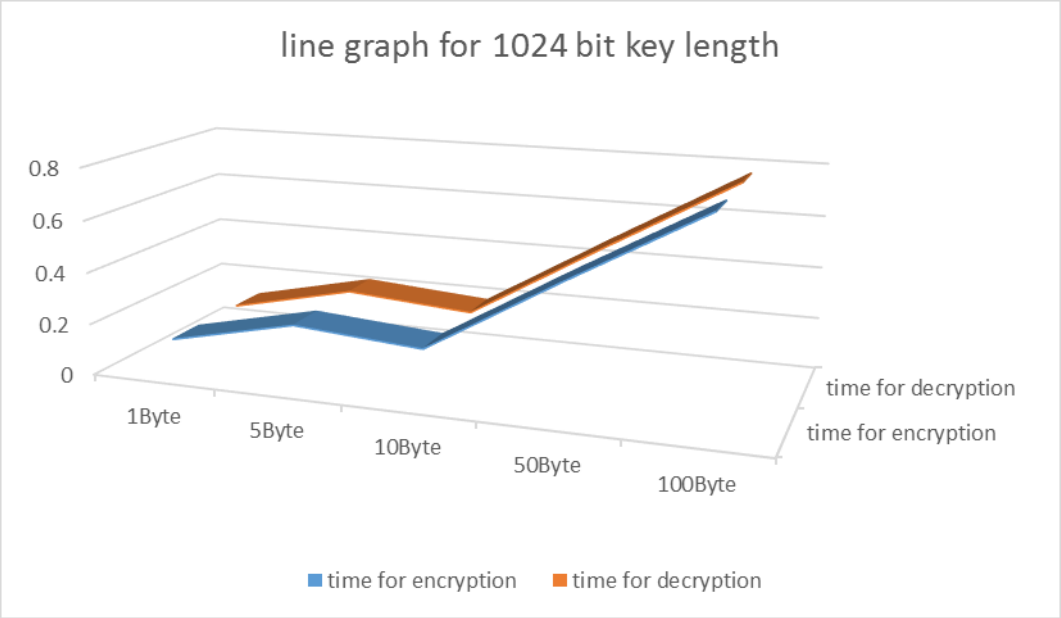
S.NO.	Key Size (bits)	Encrypting Time (seconds)	Decrypting Time (seconds)	Size of Message (bytes)	Size of prime no. p	Size of prime no. q
1.	1024	0.123	0.123	1	512	512
2.	1024	0.230	0.231	5	512	512
3.	1024	0.192	0.193	10	512	512
4.	1024	0.487	0.488	50	512	512
5.	1024	0.770	0.775	100	512	512

Table 7.3 Observation table of 1024 bits key

Table 4

S.NO.	Key Size (bits)	Encrypting Time (seconds)	Decrypting Time (seconds)	Size of Message (bytes)	Size of prime no. p	Size of prime no. q
1.	1024	0.144	0.147	1 byte	512	512
2.	1024	0.236	0.248	5 bytes	512	512
3.	1024	0.237	0.269	10 bytes	512	512
4.	1024	0.403	0.566	50 bytes	512	512
5.	1024	0.711	0.952	100 bytes	512	512

Table 7.4 Observation table 2 of 1024 bits key



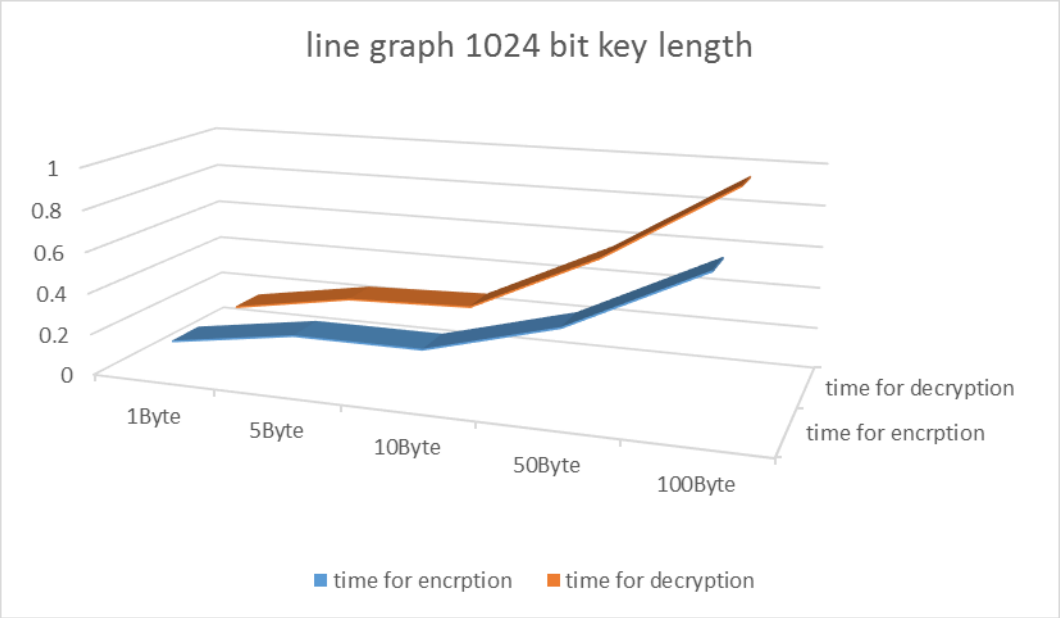
Graph (7.c) Time for encryption & decryption (Time versus Data size)

(1024-bit key length)

Description:

This graph shows that as the file size increases so the time of encryption increases and as the time of encryption increases, time of decryption also increases thus giving a somewhat similar linear equation as that of time of encryption.

This graph shows that both the time for encryption as well as decryption is directly proportional to each other. It is clearly seen that the with small prime numbers the time for encryption and decryption is approximately same, even if the length of message increases.



Graph (7.d) Time for encryption & decryption (Time versus Data size)

(1024-bit key length)

Description:

This graph shows that as the file size increases so the time of encryption increases and as the time of encryption increases, time of decryption also increases. However, it is clearly understood that the time for encryption is much different from the time of decryption. Also, it can be seen from the graph that higher range of prime numbers produce much variation in the time of encryption and decryption. Moreover, the equation obtained may not be a linear equation but is much complex.

CONCLUSION

It is quite evident from the two graphs that as the range of the prime numbers increases, the more the time of encryption and decryption takes and hence, makes it more secure against brute force attacks. Thus, making it much more secure and entry of the third party is strictly prohibited.

CONCLUSIONS

Brute force attack:

Brute force attack is inspired from number of successful hit and trials for breaking encryption code and thus includes calculation of all possible number combination of keys that could help in deciphering code thus, RSA this research paper concentrates on one of the limitation of RSA that is selection of prime numbers which are near to each other.

For RSA to be successful it is very important that the prime numbers should be chosen that even if processing speed is also enhances then also determination of prime numbers become very difficult and thus here choosing function

BigInteger.probablePrime() has given enhanced security of RSA code as it is randomly choosing prime number from the sample of very large number and thus the product of two large prime numbers is also very large which becomes very difficult for an attacker to break in polynomial solvable time, hence security is enhanced.

Increasing the key length has increased number of possible combinations of key estimation and thus possible estimation of value of key is decreased, hence security is enhanced.

With randomly choosing of very large prime numbers makes product of prime numbers (n) also very large, which is very hard to decipher even if attacker knows the value of ‘n’.