Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 6, June 2013, pg.126 – 139

RESEARCH ARTICLE

A STUDY AND PERFORMANCE ANALYSIS OF RSA ALGORITHM

M. Preetha[1], M. Nithya[2]
[1]Computer Science & Application & Periyar University, India
[2]Computer Science & Application & Periyar University, India

[1] preethasenthilkumar@gmail.com; [2] nithyamanivelan@gmail.com

Abstract— There are few end-users today who make use of real security applications. These applications tend to be too complicated, exposing too much detail of the cryptographic process. Users need simple inherent security that doesn't require more of them simply clicking the secure checkbox. Cryptography is a first abstraction to separate specific algorithms from generic cryptographic processes in order to eliminate compatibility and upgradeability problems. The core idea is enhance the security of RSA algorithm. In this dissertation public key algorithm RSA and enhanced RSA are compared analysis is made on time based on execution time.

I. INTRODUCTION

Data communication is an important aspect of our living. So, protection of data from misuse is essential. A cryptosystem defines a pair of data transformations called encryption and decryption. Encryption is applied to the plain text i.e. the data to be communicated to produce cipher text i.e. encrypted data using encryption key. Decryption uses the decryption key to convert cipher text to plain text i.e. the original data. Now, if the encryption key and the decryption key is the same or one can be derived from the other then it is said to be symmetric cryptography. This type of cryptosystem can be easily broken if the key used to encrypt or decrypt can be found. To improve the protection mechanism Public Key Cryptosystem was introduced in 1976 by Whitfield Diffe and Martin Hellman of Stanford University. It uses a pair of related keys one for encryption and other for decryption. One key, which is called the private key, is kept secret and other one known as public key is disclosed. The message is encrypted with public key and can only be decrypted by using the private key. So, the encrypted message cannot be decrypted by anyone who knows the public key and thus secure communication is possible. RSA (named after its authors – Rivest, Shamir and Adleman) is the most popular public key algorithm. In relies on the factorization problem of mathematics that indicates that given a very large number it is quite impossible in today's aspect to find two prime numbers whose product is the given number. As we increase the number the possibility for factoring the number decreases. So, we need very large numbers for a good Public Key Cryptosystem. GNU has an excellent library called GMP that can handle numbers of arbitrary precision. We have used this library to implement RSA algorithm. As we have shown in this paper number of bits encrypted together using a public key has significant impact on the decryption time and the strength of the cryptosystem.

II. CRYPTOGRAPHY

Cryptography from Greek , "hidden, secret" respectively is the practice and study of techniques for secure communication in the presence of third parties (called adversaries).More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation. Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce.

Cryptography prior to the modern age was effectively synonymous with *encryption*, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients, thereby precluding unwanted persons to do the same. Since World War I and the advent of the computer, the methods used to carry out cryptology have become increasingly complex and its application more widespread.

Modern cryptography is heavily based on mathematical theory and computer science practice. Cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system but it is infeasible to do so by any known practical means. These schemes are therefore termed computationally secure; theoretical advances and faster computing technology require these solutions to be continually adapted. There exist information-theoretically secure schemes that provably cannot be broken even with unlimited computing power—an example is the one-time pad—but these schemes are more difficult to implement than the best theoretically breakable but computationally secure mechanisms.

2.1 CRYPTOGRAPHY KEY BASICS

The two components required to encrypt data are an algorithm and a key. The algorithm generally known and the key is kept secret.

The key is a very large number that should be impossible to guess, and of a size that makes exhaustive search impractical.

In a symmetric cryptosystem, the same key is used for encryption and decryption. In an asymmetric cryptosystem, the key used for decryption is different from the key used for encryption.

2.2 KEY PAIR

In an asymmetric system the encryption and decryption keys are different but related. The encryption key is known as the public key and the decryption key is known as the private key. The public and private keys are known as a key pair.

Where a certification authority is used, remember that it is the public key that is certified and not the private key. This may seem obvious, but it is not unknown for a user to insist on having his private key certified!

2.3 KEY COMPONENT

Keys should whenever possible be distributed by electronic means, enciphered under previously established higher-level keys. There comes a point, of course when no higher-level key exists and it is necessary to establish the key manually.

A common way of doing this is to split the key into several parts (components) and entrust the parts to a number of key management personnel. The idea is that none of the key parts should contain enough information to reveal anything about the key itself.

Usually, the key is combined by means of the exclusive-OR operation within a secure environment.

In the case of DES keys, there should be an odd number of components, each component having odd parity. Odd parity is preserved when all the components are combined. Further, each component should be accompanied by a key check value to guard against keying errors when the component is entered into the system.

A key check value for the combined components should also be available as a final check when the last component is entered.

A problem that occurs with depressing regularity in the real world is when it is necessary to re-enter a key from its components. This is always an emergency situation, and it is usually found that one or more of the key component holders cannot be found. For this reason it is prudent to arrange matters so that the components are distributed among the key holders in such a way that not all of them need to be present.

For example, if there are three components (C1, C2, C3) and three key holders (H1, H2, H3) then H1 could have (C2, C3), H2 could have (C1, C3) and H3 could have (C1, C2). In this arrangement any two out of the three key holders would be sufficient.

III. TYPES OF CRYPTOGRAPHY

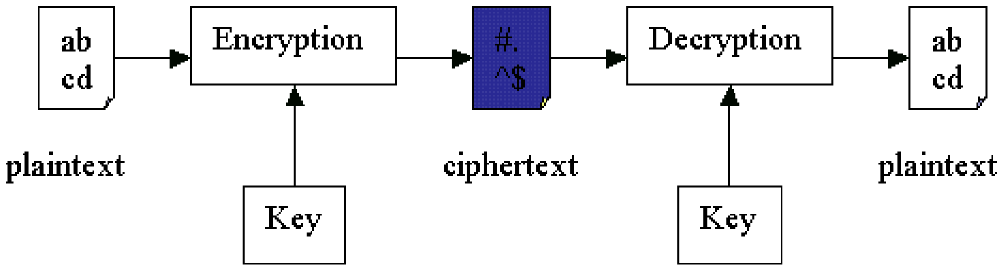
3.1 Symmetric-key cryptography

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key.

Symmetric key ciphers are implemented as either block ciphers or stream ciphers. A block cipher enciphers input in blocks of plaintext as opposed to individual characters, the input form used by a stream cipher.

The data Encryption Standard(DES) and the Advanced Encryption Standard(AES) are block cipher designs which have been designated cryptography standards by the US government Despite its deprecation as an official standard, DES remains quite popular, it is used across a wide range of applications, from ATM encryption to e-mail privacy and secure remote access. Many other block ciphers have been designed and released, either considerable variation in quality. Many have been thoroughly broken,, such as FEAL.

Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character –by-character, somewhat like the one time pad. In a stream cipher, the output stream based on a hidden state which changes as the cipher operates.



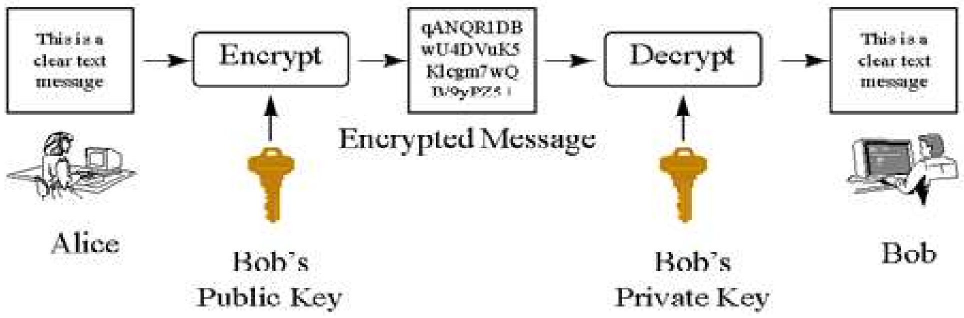
3.1.1 Symmetric key Cryptography

3.2 Asymmetric key Cryptography

Public-key cryptography refers to a cryptographic system requiring two separate keys, one of which is secret and one of which is public. Although different, the two parts of the key pair are mathematically linked. One key locks or encrypts the plaintext, and the other unlocks or decrypts the cipher text. Neither key can perform both functions by itself. The public key may be published without compromising security, while the private key must not be revealed to anyone not authorized to read the messages.

Public-key cryptography uses asymmetric key algorithms (such as RSA), and can also be referred to by the more generic term "asymmetric key cryptography." The algorithms used for public key cryptography are based on mathematical relationships that presumably have no efficient solution. Although it is computationally easy for the intended recipient to generate the public and private keys, to decrypt the message using the private key, and easy for the sender to encrypt the message using the public key, it is extremely difficult (or effectively impossible) for anyone to derive the private key, based only on their knowledge of the public key.

This is why, unlike symmetric key algorithms, a public key algorithm does *not* require a secure initial exchange of one (or more) secret keys between the sender and receiver. The use of these algorithms also allows the authenticity of a message to be checked by creating a digital signature of the message using the private key, which can then be verified by using the public key. In practice, only a hash of the message is typically encrypted for signature verification purposes.

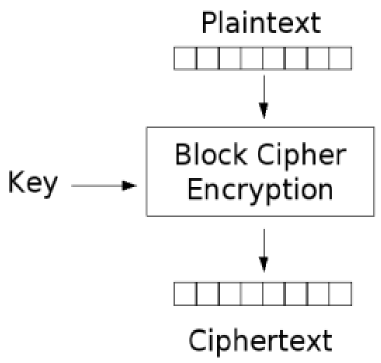

This is a clear text message
Encrypt
qANQR1DB wU4D VuK5 Klcgm7wQ D/9yPZ5
Decrypt
This is a clear text message
Encrypted Message
Alice
Bob's Public Key
Bob's Private Key
Bob

3.2.1 Asymmetric key Cryptography

3.3 Block Cipher and Stream Cipher Cryptanalysis

Definition of Block Cipher

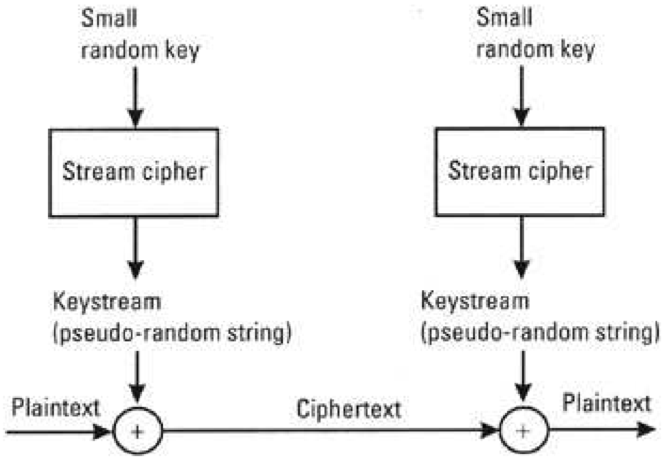
Block ciphers encrypt information by breaking it down into blocks and encrypting data in each block. A block cipher encrypts data in fixed sized blocks (commonly of 64 bits).



3.3.1 Block Cipher

Definition of Stream Cipher

A stream cipher consists of a state machine that outputs at each state transition one bit of information. This stream of output bits is commonly called the running key. The state machine is nothing more than a pseudo-random number generator.



IV. ALGORITHMS

4.1 DES

The Data Encryption Standard (DES) was developed in the 1970s by the National Bureau of Standards with the help of the National Security Agency. Its purpose is to provide a standard method for protecting sensitive commercial and unclassified data. IBM created the first draft of the algorithm, calling it LUCIFER. DES officially became a federal standard in November of 1976.

Fundamentally DES performs only two operations on its input, bit shifting, and bit substitution. The key controls exactly how this process works. By doing these operations repeatedly and in a non-linear manner you end up with a result which cannot be used to retrieve the original without the key. Those familiar with chaos theory should see a great deal of similarity to what DES does. By applying relatively simple operations repeatedly a system can achieve a state of near total randomness.

DES works on 64 bits of data at a time. Each 64 bits of data is iterated on from 1 to 16 times (16 is the DES standard). For each iteration a 48 bit subset of the 56 bit key is fed into the encryption block represented by the dashed rectangle above. Decryption is the inverse of the encryption process. The "F" module shown in the diagram is the heart of DES. It actually consists of several different transforms and non-linear substitutions. Consult one of the references in the bibliography for details.

Recent analysis has shown despite this controversy, that DES is well designed. DES is theoretically broken using Differential or Linear Cryptanalysis but in practise is unlikely to be a problem yet. Also rapid advances in computing speed though have rendered the 56 bit key susceptible to exhaustive key search, as predicted by Diffie & Hellman. Have demonstrated breaks:

- 1997 on a large network of computers in a few months
- 1998 on dedicated h/w (EFF) in a few days
- 1999 above combined in 22hrs!

4.1.1 TRIPLE DES

The potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative. One approach is to design a complete new algorithm. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryptions with DES and multiple keys.

Encryption of more than one block

As with all block ciphers, encryption and decryption of multiple blocks of data may be performed using a variety of modes of operation, which can generally be defined independently of the block cipher algorithm.

4.1.2 Triple DES Security

In general, Triple DES with three independent keys (keying option 1) has a key length of 168 bits (three 56-bit DES keys), but due to the meet-in-the-middle attack, the effective security it provides is only 112 bits. Keying option 2 reduces the key size to 112 bits. However, this option is susceptible to certain chosen-plaintext or known-plaintext attacks, and thus, it is designated by NIST to have only 80 bits of security.

The best attack known on keying option 1 requires around 2^{32} known plaintexts, 2^{113} steps, 2^{90} single DES encryptions, and 2^{88} memories. This is not currently practical and NIST considers keying option 1 to be appropriate through 2030. If the attacker seeks to discover any one of many cryptographic keys, there is a memory-efficient attack which will discover one of 2^{28} keys, given a handful of chosen plaintexts per key and around 2^{84} encryption operations.

DES is now considered to be insecure for many applications. This is chiefly due to the 56-bit key size being too small.

4.2 AES

AES is based on a design principle known as a substitution-permutation network, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, to that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4×4 column-major order matrix of bytes, termed the *state*, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the cipher text.

For cryptographers, a cryptographic "break" is anything faster than a brute force—performing one trial decryption for each key (see Cryptanalysis). This includes results that are infeasible with current technology. The largest successful publicly known brute force attack against any block-cipher encryption was against a 64-bit RC5.

4.3 BLOW FISH

Blowfish is a keyed, symmetric block cipher, designed in 1993 by Bruce Schneier and included in a large number of cipher suites and encryption products. Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date. However, the Advanced Encryption Standard now receives more attention.

Schneier designed Blowfish as a general-purpose algorithm, intended as an alternative to the ageing DES and free of the problems and constraints associated with other algorithms. At the time Blowfish was released, many other designs were proprietary, encumbered by patents or were commercial/government secrets.

Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits. It is a 16-round Feistel cipher and uses large key-dependent S-boxes.

Decryption is exactly the same as encryption, except that P1, P2,..., P18 are used in the reverse order. This is not so obvious because XOR is commutative and associative.

Because the P-array is 576 bits long, and the key bytes are XOR through all these 576 bits during the initialization, many implementations support key sizes up to 576 bits. While this is certainly possible, the 448 bits limit is here to ensure that every bit of every sub key depends on every bit of the key, as the last four values of the P-array don't affect every bit of the cipher text. This point should be taken in consideration for implementations with a different number of rounds, as even though it increases security against an exhaustive attack, it weakens the security guaranteed by the algorithm. And given the slow initialization of the cipher with each change of key, it is granted a natural protection against brute-force attacks, which doesn't really justify key sizes longer than 448 bits.

Blowfish was one of the first secure block ciphers not subject to any patents and therefore freely available for anyone to use. This benefit has contributed to its popularity in cryptographic software.

V. PUBLIC KEY CRYPTOGRAPHY

5.1 Introduction

The RSA public key cryptosystem was invented by R. Rivest, A. Shamir and L.
Adleman. The RSA cryptosystem is based on the dramatic difference between the ease of finding large primes and the difficulty of factoring the product of two large prime numbers.

- RSA is a public key algorithm invented by Rivest, Shamir and Adleman. The key used for encryption is different from (but related to) the key used for decryption.
- The algorithm is based on modular exponentiation. Numbers e, d and N are chosen with the property that if A is a number less than N, then (Ae mod N)d mod N = A.
- This means that you can encrypt A with e and decrypt using d. Conversely you can encrypt using d and decrypt using e (though doing it this way round is usually referred to as signing and verification).
    - The pair of numbers (e,N) is known as the public key and can be published.
    - The pair of numbers (d,N) is known as the private key and must be kept secret.
- The number e is known as the public exponent, the number d is known as the private exponent, and N is known as the modulus. When talking of key lengths in connection with RSA, what is meant is the modulus length.
- An algorithm that uses different keys for encryption and decryption is said to be asymmetric.
- Anybody knowing the public key can use it to create encrypted messages, but only the owner of the secret key can decrypt them.
- Conversely the owner of the secret key can encrypt messages that can be decrypted by anybody with the public key. Anybody successfully decrypting such messages can be sure that only the owner of the secret key could have encrypted them. This fact is the basis of the digital signature technique.

5.2 RSA Encryption

Suppose Bob wishes to send a message (say 'm') to Alice. To encrypt the message using the RSA encryption scheme, Bob must obtain Alice's public key pair (e , n). The message to send must now be encrypted using this pair (e , n). However, the message 'm' must be represented as an integer in the interval [0, n-1]. To encrypt it, Bob simply computes the number 'c' where c = m ^ e mod n. Bob sends the cipher text c to Alice.

5.3 RSA Decryption

To decrypt the cipher text c, Alice needs to use her own private key d (the decryption exponent) and the modulus n. simply computing the value of c ^ d mod n yields back the decrypted message (m).

5.4 Attacks against RSA

Through the basic algorithm is secure, there are attacks on how RSA is implemented

1. Forward search attack: If message space is predictable, attacker can decrypt C simply by
Encrypting all possible messages until a match with C is obtained.
2. Common modulus attack: If everyone is given the same modulus „n‟ but different (e,d) pair, then under certain conditions, it is possible to decrypt the message without d.
3. Low encryption exponents: When encrypting with low encryption exponents (e.g., $e = 3$) and small values of the m, (i.e. $m < n1 / e$) the result of me is strictly less than the modulus n. In this case, cipher texts can be easily decrypted by taking the eth root of the cipher text over the integers.
4. RSA has the property that the product of two cipher texts is equal to the encryption of the product of the respective plaintexts. That is m1em2e=(m1m2)e(mod n) Because of this multiplicative property a chosen-cipher text attack is possible.

5.5 Key Generation Algorithm

1. Generate two large random primes, *p* and *q*, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits.
2. Compute n = pq and (phi) φ = (p-1)(q-1).
3. Choose an integer *e*, 1 < e < phi, such that gcd(e, phi) = 1.
4. Compute the secret exponent *d*, 1 < d < phi, such that ed ≡ 1 (mod phi).
5. The public key is (n, e) and the private key (d, p, q). Keep all the values d, p, q and phi secret. [We prefer sometimes to write the private key as (n, d) because you need the value of n when using d.]

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent* or just the *exponent*.
- d is known as the *secret exponent* or *decryption exponent*

5.6 RSA ALGORITHM

1. Choose two distinct prime numbers, *p* and *q*.
2. Let $n = pq$.
3. Let $\varphi(pq) = (p - 1)(q - 1)$. ($\varphi$ is totient function).
4. Pick an integer *e* such that $1 < e < \varphi(pq)$, and *e* and $\varphi(pq)$ share no divisors other than 1 (*e* and $\varphi(pq)$ are coprime).
5. Find *d* which satisfies

.

d is a secret private key exponent.

1. The public key consists of *e* (often called public exponent) and *n*(often called modulus). The private key consists of *e* and *d* (private exponent).

The message *m* is encrypted using formula

Where *c* is the encrypted message. The encrypted message is decrypted using formula

Encryption and decryption formulas show how to encode and decode a single integer. Bigger (or different) pieces of information are encoded by converting them into (potentially large) integers first. As RSA is not particularly fast, it is usually only to encrypts the key of some faster algorithm. After RSA decrypts the key, this supplementary algorithm uses it to decrypt the rest of the message.

RSA algorithm is fundamentally based on the Euler theorem:

Where *a* and *n* are positive integers and *a* is a co-prime to *n*.

To break the algorithm from the mathematical side, one needs to solve the factoring problem (find the two prime numbers that, when multiplied, produce the given result). When the picked numbers are large enough, the problem cannot be easily solved by brute force and at least currently it also does not have easier analytic solution

.

Encryption

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer *m*, $1 < m < n$.
3. Computes the cipher text $c = m^e \bmod n$.
4. Sends the cipher text *c* to B.

Decryption

The plaintext message can be quickly recovered when the decryption key d, an inverse of e modulo (p-1)(q-1) is known. (Such an inverse exists since gcd(e,(p-1)(q-1))=1). To see this, note that if d e □ 1 (mod (p-1)(q-1)), there is an integer k such that d e = 1 + k(p-1)(q-1). It follows that.

$C^d = (M^e)^d = M^{de} = M^{1+k\,(p-1)(q-1)}$

By Fermat's theorem (assuming that gcd(M,p) = gcd(M,q) = 1, which holds except in rare cases, it follows that M^{p-1} □ 1 (mod p) and M^{q-1} □ 1 (mod q), consequently.

$C^d = M \cdot (M^{p-1)\,k\,(q-1)})$ □ $M \cdot 1$ □ M (mod p)

and:-$C^d = M \cdot (M^{q-1)\,k\,(p-1)})$ □ $M \cdot 1$ □ M (mod q)

Since gcd(p,q) = 1, it follows that:-

C^d □ M (mod pq)

5.7 WORKING PRINCIPLES OF RSA ALGORITHM

1. To generate the primes *p* and *q*, generate a random number of bit length b/2 where *b* is the required bit length of *n*; set the low bit (this ensures the number is odd) and set the *two* highest bits (this ensures that the high bit of *n* is also set); check if prime (use the *Rabin-Miller* test); if not, increment the number by two and check again until you find a prime. This is *p*. Repeat for *q* starting with a random integer of length b-b/2. If p<q, swop *p* and *q* (this only matters if you intend using the CRT form of the private key). In the extremely unlikely event that p = q, check your random number generator. Alternatively, instead of incrementing by 2, just generate another random number each time.

   There are stricter rules in ANSI X9.31 to produce *strong primes* and other restrictions on *p* and *q* to minimize the possibility of known techniques being used against the algorithm. There is much argument about this topic. It is probably better just to use a longer key length.
2. In practice, common choices for *e* are 3, 17 and 65537 (2^{16}+1). These are Fermat primes, sometimes referred to as F0, F2 and F4 respectively (Fx=2^(2^x)+1). They are chosen because they make the modular exponentiation operation faster. Also, having chosen *e*, it is simpler to test whether gcd(e, p-

1)=1 and gcd(e, q-1)=1 while generating and testing the primes in step 1. Values of *p* or *q* that fail this test can be rejected there and then. (Even better: if *e* is prime and greater than 2 then you can do the less-expensive test (p mod e)!=1 instead of gcd(p-1,e)==1.)

3. To compute the value for *d*, use the *Extended Euclidean Algorithm* to calculate d = e^{-1} mod phi, also written d = (1/e) mod phi. This is known as *modular inversion*. Note that this is not integer division. The modular inverse *d* is defined as the integer value such that ed = 1 mod phi. It only exists if *e* and *phi* have no common factors.
4. When representing the plaintext octets as the representative integer *m*, it is usual to add random padding characters to make the size of the integer *m* large and less susceptible to certain types of attack. If m = 0 or 1 or n-1 there is no security as the cipher text has the same value. For more details on how to represent the plaintext octets as a suitable representative integer *m*, see PKCS#1 Scheme below or the reference itself [PKCS1]. It is important to make sure that m < n otherwise the algorithm will fail. This is usually done by making sure the first octet of m is equal to 0x00.
5. Decryption and signing are identical as far as the mathematics is concerned as both use the private key. Similarly, encryption and verification both use the same mathematical operation with the public key. That is, mathematically, for m < n,

$m = (m^e \bmod n)^d \bmod n = (m^d \bmod n)^e \bmod n$

However, note these important differences in implementation:-

- The signature is derived from a message digest of the original information. The recipient will need to follow exactly the same process to derive the message digest, using an identical set of data.
- The recommended methods for deriving the representative integers are different for encryption and signing (encryption involves random padding, but signing uses the same padding each time).

6. The original definition of RSA uses the Euler totient function φ(n) = (p-1)(q-1). More recent standards use the *Charmichael function* λ(n) = lcm(p-1, q-1) instead. λ(n) is smaller than φ(n) and divides it. The value of d' computed by d' = e^{-1} mod λ(n) is usually different from that derived by d = e^{-1} mod φ(n), but the end result is the same. Both d and d' will decrypt a message m^e mod n and both will give the same signature value s = m^d mod n = $m^{d'}$ mod n. To compute λ(n), use the relation
7. λ(n) = (p-1)(q-1) / gcd(p-1, q-1).

5.8 Example of RSA encryption

Step 1: Select primes p=11, q=3.

Step 2: n = pq = 11.3 = 33

phi = (p-1)(q-1) = 10.2 = 20

Step 3: Choose e=3

Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),

and check gcd(e, q-1) = gcd(3, 2) = 1therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1

Step 4: Compute d such that ed ≡ 1 (mod phi)

i.e. compute d = e-1 mod phi = 3-1 mod 20

i.e. find a value for d such that phi divides (ed-1)

i.e. find d such that 20 divides 3d-1.

Simple testing (d = 1, 2, ...) gives d = 7

Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.

Step 5: Public key = (n, e) = (33, 3)

Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA Algorithm works. Now say we want to encrypt the message m = 7,

c = me mod n = 73 mod 33 = 343 mod 33 = 13.

Hence the cipher text c = 13.

Step 6: To check decryption we compute

m' = cd mod n = 137 mod 33 = 7.

Note that we don't have to calculate the full value of 13 to the power 7 here. We can Make use of the fact that

a = bc mod n = (b mod n).(c mod n) mod n

so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

One way of calculating m' is as follows:-

m' = 137 mod 33 = 13(3+3+1) mod 33 = 133.133 .13 mod 33

=(133 mod 33).(133 mod 33).(13 mod 33) mod 33
=(2197 mod 33).(2197 mod 33).(13 mod 33) mod 33
=19.19.13 mod 33 = 4693 mod 33
=7.
Now if we calculate the cipher text c for all the possible values of m (0 to 32), we get
m 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
c 0 1 8 27 31 26 18 13 17 3 10 11 12 19 5 9 4
m 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
c 29 24 28 14 21 22 23 30 16 20 15 7 2 6 25 32

Note that all 33 values of m (0 to 32) map to a unique code c in the same range in a sort of random manner. In this case we have nine values of m that map to the same value of c - these are known as unconcealed messages. m = 0, 1 and n-1 will always do this for any n, no matter how large. But in practice, higher values shouldn't be a problem when we use large values for n in the order of several hundred bits.

If we wanted to use this system to keep secrets, we could let A=2, B=3, ..., Z=27. (We specifically avoid 0 and 1 here for the reason given above). Thus the plaintext message "HELLOWORLD" would be represented by the set of integers m1, m2, ...
(9,6,13,13,16,24,16,19,13,5)
Using our table above, we obtain cipher text integers c1, c2, ...
(3,18,19,19,4,30,4,28,19,26)
Remember that calculating me mod n is easy, but calculating the inverse c-e mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

VI. PROPOSED ALGORITHM

6.1 OAEP

Optimal Asymmetric Encryption Padding (OAEP) is a method for encoding messages developed by Mihir Bellare and Phil Rogaway . The technique of encoding a message with OAEP and then encrypting it with RSA is provably secure in the random oracle model. Informally, this means that if hash functions are truly random, then an adversary who can recover such a message must be able to break RSA.

An OAEP encoded message consists of a ``masked data" string concatenated with a ``masked random number". In the simplest form of OAEP, the masked data is formed by taking the XOR of the plaintext message *M* and the hash *G* of a random string *r*. The masked random number is the XOR of *r* with the hash *H* of the masked data. The input to the RSA encryption function is then

[*M* □*G*(*r*)] □□ [*r* □*H*(*M* □*G*(*r*))]

Often, OAEP is used to encode small items such as keys. There are other variations on OAEP (differing only slightly from the above) that include a feature called ``plaintext-awareness". This means that to construct a valid OAEP encoded message, an adversary must know the original plaintext. To accomplish this, the plaintext message M is first padded (for example, with a string of zeroes) before the masked data is formed. OAEP is supported in the ANSI X9.44, IEEE P1363 and SET standards.

The following notation will be used.

An octet is the eight-bit representation of an integer with the leftmost bit being the most significant
bit; this integer is the value of the octet.
An octet string is an ordered sequence of octets, where the first octet is the leftmost.
GCD(a,b) greatest common divisor of the two nonnegative integers a and b
LCM(a,b) least common multiple of the two nonnegative integers a and b
bxc the real number x rounded down to the closest integer
dxe the real number x rounded up to the closest integer
XkY concatenation of octet strings X and Y
X _ Y bitwise exclusive-or of octet strings X and Y

6.2 Security properties

The security of RSAES-OAEP depends on the security of the underlying RSA encryption and Decryption primitives, RSAEP and RSADP and the Security of the OAEP encoding method. The advantage of the technique that is generically known as OAEP (Optimal Asymmetric Encryption Padding) is that under one model of analysis -- the so-called random oracle model -- the security of RSAES-OAEP can be tightly related to the security of RSAEP/RSADP. This allows us to consider the security of RSAES-OAEP

RSA encryption and decryption primitive over the years many different researchers have considered the security of RSAEP/RSADP. Boneh gives an excellent survey of the main attacks which we summarize here. In some cases, the discussion of the private exponent d refers to the inverse of e mod (p − 1)(q − 1) as opposed to the alternative definition given in this document; knowledge of either is of course sufficient to compromise security.

1. Taking eth roots of c modulo n when the factorization of n is unknown.

This is an open problem and there are currently no practical techniques for achieving this when typical parameter choices are made. Although the RSA problem of taking eth roots modulo n is not known to be equivalent to factoring the modulus, factorization is the only method known for solving the problem in the general case. Boneh and Venkatesan have shown that if there is an algebraic reduction from factoring to eth roots in time T, then it is possible to factor in (roughly) time 2eT. This means that, for very small e (say, less than 64), if factoring is hard, then the problems are not equivalent (at least via algebraic reductions). For larger e (for instance, e = 216 + 1), there still might be an efficient reduction. However, see further notes below for possible methods of determining the private key d, and hence solving the problem as well as factoring the modulus, when sufficient information about the private key is leaked.

2. Factoring n and then taking eth roots of c modulo n.

Trends in the effectiveness of factoring integers are carefully collated and scrutinized by the cryptographic community. Progress over past years has been gradual but steady. Under a variety of models it is possible to provide a range of predictions for the continued resistance of an RSA modulus n to a factoring attack .The most recent factorization of an RSA modulus was RSA-512, a 512-bit RSA modulus .It is possible to use this empirical evidence as a base point from which to make estimates for- 19the likely security of RSA module of different sizes. While there are a variety of comparisons available which sometimes offer divergent views, there seems to be a general consensus that the security offered by 1024-bit RSAEP/RSADP is roughly equivalent to that offered by 80-bit symmetric key cryptography in terms of computational effort1. Note that the user can freely choose appropriate parameter choices to give a level of protection appropriate to the user's own risk assessment and key lengths of 2048-bit and higher offer an increasingly significant margin for security. Recent proposals to use an opto-electronic device TWINKLE to speed up part of the factoring process are unlikely to have any significant impact at the recommended parameter choices today.

3. Two users sharing a common modulus.

Two users should never share the same modulus n, even if they use different encryption/Decryption exponent pairs. Systems that allow users to share moduli are using RSAEP/RSADP inappropriately.

4. Using a small private exponent d.

It may be tempting to use a small private exponent d for reasons of efficiency. A basic implementation of RSAEP/RSADP can be susceptible to attack if d < n0.292. It is conjectured that this might continue to be the case if d < n0.5. A small private exponent d should not be used.

5. Using a low public exponent e.

Some progress has been made [13] on exploiting the use of a low public exponent. While there is no particular attack within the context of RSAES-OAEP that compromises the security of the public exponent e = 3, more conservative users may prefer to use other public exponents such as e = 17 or e = 216 + 1 while still retaining a very competitive performance for encryption. Also, as noted further in Annex D.4.3.4 of IEEE Std 1363-2000 , a larger public exponent can provide an additional level of defense in the case that the underlying random number generation fails in an implementation of the OAEP method, undermining the security properties offered by that method.

6. Broadcasting the same message to multiple users.

It has been known for some time that it can be unsafe to broadcast the same message to different users if no padding or a very simple padding scheme is used. Application of allows improvements to this original work to be made. The application of EME-OAEP as the padding scheme prior to encryption is sufficient to resist these attacks.

7. Sending related messages to the same user.

For small e it can be possible to recover simply-related messages that are encrypted under the same public-key . Extensions showed some practical applications of this work when small amounts of random padding are used prior to encrypting with RSAEP. In an attack is described that applies to a case where the plaintext ends by sufficiently many zeroes, and two or more cipher texts corresponding to the same plaintext are available. The application of EME-OAEP as the padding scheme prior to encryption is sufficient to resist these attacks. 1Under an equivalent-cost analysis 1024-bit RSAEP/RSADP is viewed as offering greater security than 80-bit Symmetric key cryptography.

8. Using partial information about the private key d.

Given the dlog2 n/4e least significant bits of the private exponent d, it is possible to reconstruct all of d if e < p n . Furthermore, when a small exponent e is used, the most significant half of the bits of d can be leaked.

Although determining the remaining bits is of course still difficult, if the private exponent is protected by symmetric encryption, knowledge of the most significant half of the bits of d may facilitate a known-plaintext attack on the symmetric Encryption method. Accordingly, it is essential that the remaining bits of the private key d Should be well protected.

9. Using partial information about the factors p, q.

Given the dlog2n/4e least significant bits of p (resp. q) or the dlog2n/4e most significant bits of p (resp. q), one can efficiently factor n . The entirety of the secret primes p and q should be protected. It is generally accepted that when RSAEP is used with appropriate parameter choices and coupled with a secure padding scheme like OAEP, then the most effective attack is to factor the modulus n.

Under this assumption we can relate the security of RSAES-OAEP to the effort required to factor the underlying modulus of different sizes. A crude estimate for the increased computing resources required beyond that for factoring RSA-512can be derived for different sizes of RSA moduli. For 1024-bit RSA moduli, the factor increasing computational power is estimated as 7 × 106 while for 2048-bit RSA the estimate is 9 × 1015.Increases in computing power might be accounted for by some combination of the use of more machines, increasingly powerful machines, or more calendar time. The calendar time required for the factorization of RSA-512 was 3.7 months. Other issues like the cost and availability of memory may also figure in deriving predictions for the future security of RSAEP/RSADP reasons to call the security of RSAES-OAEP in question. Luckily, this is not the case; RSAES-OAEP

Basic techniques to avoid implementation weaknesses

The analytical securities of RSAEP/RSADP along with RSAES-OAEP have been considered in previous section. However we still need to implement RSAES-OAEP securely.

It is possible that weaknesses could be introduced when writing RSAES-OAEP as a component of an application, or when running an application from which so-called side-channel information can be deduced. Within an engineering environment, implementation errors can be virtually eliminated by adopting good product design and engineering practices. Adequate testing and product management throughout the development cycle are essential. With regards to running an application using RSAES-OAEP, protection of all private key information, secure memory management, and secure error handling are all needed. Issues like the source of random numbers are, of course, fundamental to the security of the implementation. Over recent years there have been several proposals to break cryptosystems by utilizing so-called side-channel information. Examples include timing attacks, power analysis , and fault analysis .Implementations of RSAES-OAEP can be made resistant to timing attacks and power analysis by ensuring that all the steps in the computation of a private key operation take the same amount of time or consume the same amount of power .A more elegant approach to providing resistance to timing attacks is to use blinding as suggested by Ronald L. Rivest.

6.3 Optimal asymmetric encryption padding

In cryptography, **Optimal Asymmetric Encryption Padding** (**OAEP**) is a padding scheme often used together with RSA encryption. OAEP was introduced by Bellare and Rogaway.

The OAEP algorithm is a form of Feistel network which uses a pair of random oracles G and H to process the plaintext prior to asymmetric encryption. When combined with any secure trapdoor one-way permutation f, this processing is proved in the random oracle model to result in a combined scheme which is semantically secure under chosen plaintext attack. When implemented with certain trapdoor permutations, OAEP is also proved secure against chosen cipher text attack. OAEP can be used to build an all-or-nothing transform.

OAEP satisfies the following two goals:

1. Add an element of randomness which can be used to convert a deterministic encryption scheme into a probabilistic scheme.
2. Prevent partial decryption of cipher texts (or other information leakage) by ensuring that an adversary cannot recover any portion of the plaintext without being able to invert the trapdoor one-way permutation f.

The original version of OAEP (Bellare/Rogaway, 1994) showed a form of "plaintext awareness" in the random oracle model when OAEP is used with any trapdoor permutation. Subsequent results contradicted this claim, showing that OAEP was only IND-CCA1 secure. However, the original scheme was proved in the random oracle model to be IND-CCA2 secure when OAEP is used with the RSA permutation using standard encryption exponents, as in the case of RSA-OAEP. An improved scheme that works with any trapdoor one-way permutation was offered by Victor Shoup to solve this problem. More recent work has shown that in the standard model, that it is impossible to prove the IND-CCA2 security of RSA-OAEP under the assumed hardness of the RSA problem.

To encode,

1. Messages are padded with k_1 zeros to be $n - k_0$ bits in length.
2. r is a random k_0-bit string

3. *G* expands the k_0 bits of *r* to $n - k_0$ bits.
4. $X = m00..0 \oplus G(r)$
5. *H* reduces the $n - k_0$ bits of *X* to k_0 bits.
6. $Y = r \oplus H(X)$
7. The output is $X \| Y$ where *X* is shown in the diagram as the leftmost block and *Y* as the rightmost block.

To decode,

1. recover the random string as $r = Y \oplus H(X)$
2. Recover the message as $m00..0 = X \oplus G(r)$

The "all-or-nothing" security is from the fact that to recover m, you must recover the entire X and the entire Y; X is required to recover r from Y, and r is required to recover m from X. Since any changed bit of a cryptographic hash completely changes the result, the entire X, and the entire Y must both be completely recovered.

6.4 Implementation of RSA Algorithm

```
Step 1: Create p & q
Step 2: Calculate N=p*q
Step 3: Calculate N1=(p-1)(q-1)
Step 4: Select Encryption
E=D.ModInverese(N1)
Step 5: Select Decryption
D=Choose 256 bits of random number
Step 6: Calculate OAEP
Padding
            m=Give any value (random number)
Step 7: padding=m-plaintext length
Padded message M bit length of m
     k=give a value (random number)
Create new random variable r of k bits
Step 8: Create G(r) which m bit integer from r bit integer
     Gofr=r.shiftleft(m-k)
Step 9: Create p1 & p2
     p1=M.xor(gofr)
     Hofp1=p1.shiftright(k-m)
     P2=Hofp1.xor(r)
Step 10: plain=Concatenate p1 and p2
Step 11: Decryption (OAEP)
r=(p1.shiftright(k-m)).xor(p2)
M=(r.shiftleft(m-k)).xor(p1)
Plaintext=M.shiftright(padding)
Step 12: Encryption
Cipher=plain.modpow(E,N)
Step 13: Decryption
Plaintext = Cipher.modpow(D,N)
OAEP Decryption (p1,p2)
Step 14: Plaintext=Plain.
```

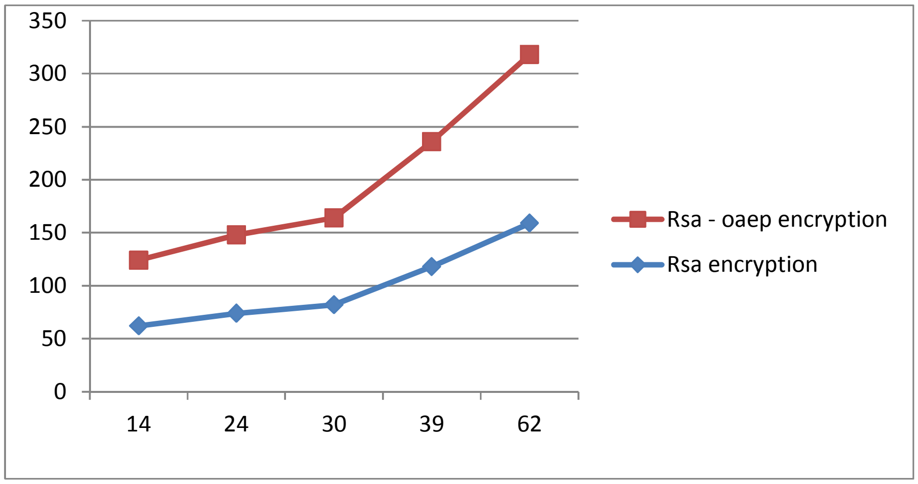
VII. RESULT AND DISCUSSION

RSA and RSA-OAEP are implemented in java. These algorithms are tested by different file size and calculate the encryption and decryption times. The results are tabulated as follows

7.1 RSA-OAEP encryption decryption

Input Size	Encryption Time	Decryption Time	Total Time
14kb	10	52	62
24kb	13	61	74
30kb	16	66	82
39kb	17	101	118
62kb	17	142	159

Input Size(kb)	Encryption Time	Decryption Time	Total Time
14kb	15	63	78
24kb	16	76	92
30kb	15	78	93
39kb	15	170	185
62kb	16	190	206



7.2 RSA AND RSA – OAEP ENCRYPTION DECRYPTION

When comparing with RSA, RSA – OAEP algorithm requires more time for encryption decryption. Whereas RSA-OAEP is more secured cryptography algorithm than RSA, because RSA –OEAP includes OAEP concept, which is more difficulty for the intruder to find the plain text from the encrypted message. So it is finalized that RSA –OAEP is secured encryption and decryption algorithm.

VIII. Conclusion

A slight modification of the well-known and practical RSA-OAEP has been included encryption. According to this scheme it has extra advantages, namely its IND-CCA, security remains highly related to hardness of the RSA problem, even in the multi-query setting. The RSA provides highest security to the business application. Moreover, this scheme can be used for encryption of long messages without employing the hybrid and symmetric encryption.

References

[1] R.L.Rivest,A.Sharmir,L.Adleman : A method for obtaining digital signatures and public key Cryptosystems", Tata McGraw-Hill

[2] W. Stallings, Cryptography and Network Security: Principles and Practice 3/e. Prentice Hall

[3] Cormen , Thomas H, Charles E.Leiserson, aronald L.Rivest Clifford Stein "Introduction to algorithms". MIT Press and McGraw-Hill

[4] www.rsa.com

[5] www.cc.gatech.edu

[6] www.symmtech.com

[7] www.rsasecurity.com

[8] www.cryptotools.com