www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 5 Issue 09 September 2016 Page No.18074-18080

The Comparative Study on Public Key Algorithm using RSA AND OAEP

E.Vinithra[#1], *N.Hyrunnisha*[#2]
Department of Computer Science
Muthayammal College of Arts and Science
Namakkali District, Tamilnadu, India
[1]vinithra18@gmail.com
Department of Computer Science
Muthayammal College of Arts and Science
Namakkali District, Tamilnadu, India
[2]canha@muthayammal.com

ABSTRACT
The cryptography algorithms 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.

Index Terms* - *Public Key ,Encryption, RSA, Security,OAEP

I INTRODUCTION

Data communication is an important aspect of our living. So, protection of data from misuse is essential.[1] 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[2][3] 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. Cryptographic algorithms are divided into two types viz., symmetric key and public key. In symmetric key algorithm, only one key is used for both encryption and decryption. [4][5]The key must be known well in advance to both the parties before the messages being encrypted. However these algorithms suffer from disadvantages of the key distribution problem. Despite this drawback, the various symmetric key algorithms have been developed recently viz., DES, AES, BLOWFISH, CAST etc. In public key algorithms, every person has a pair of keys viz., private key and public key. Further one key is calculated from other key. To perform encryption using public key algorithm the sender uses the receiver's public key, and at other end the receiver uses his/her private key to perform decryption. Numerous public key algorithms are available viz., RSA, Rabin, ELGamal, etc.

In recent times, the analysis of cryptographic algorithm gained attention from researcher. Enhancing security is a major challenging task in cryptography. [6]The security of the many cryptographic algorithms depends on the generation of and predictable quantities such as the key stream in vernam's one-time pad, the primes p and q in RSA encryption, secret key in the DES algorithm. In all these cases the quantities must be generated sufficient in size and at random, in the sense that the probability of any particular value being selected must be sufficiently small. Even if the above said parameters are taken carefully none of the computational problem is fully secured enough. Moreover, to enhance the security in some symmetric key algorithms, normally the key is selected in such a way that the size of the key is equal to the size of the plaintext. Similarly the number of bits in the key is chosen very large in public key algorithms and some problems occur in storing and distributing the key. To avoid them, instead of increasing the key size, the focus is given for plaintext.

The rest of the paper is structured as follows. Related work is described in Section 2. The proposed Algorithm is presented in section 3. Section 4 describes an illustration of the RSA and proposed methodology. Section 5 illustrates a brief implementation of RSA algorithm based on our proposed methodology. Results and Discussion are presented in section 6. Finally section 7 ends with conclusion.

II. RELATED WORK

According to " Rajorshi Biswas, Shibdas Bandyopadhyay, Anirban Banerjee", Fast implementation of rsa algorithm. Organizations in both public and private sectors have become increasingly dependent on electronic data processing. Protecting these important data is of utmost concern to the organizations and cryptography is one of the primary ways to do the job. Public Key Cryptography is used to protect digital data going through an insecure channel from one place to another. RSA algorithm is extensively used in the popular implementations of Public Key Infrastructures. In this paper, we have done an efficient implementation of RSA algorithm using gmp library from GNU. We have also analyzed the changes in the performance of the algorithm by changing the number of characters we are encoding together
According to "Xin Zhou ",Research and implementation of RSA algorithm for encryption and decryption Cryptographic technique is one of the principal means to protect information security. Not only has it to ensure the information confidential, but also provides digital signature, authentication, secret sub-storage, system security and other functions. Therefore, the encryption and decryption solution can ensure the confidentiality of the information, as well as the integrity of information and certainty, to prevent information from tampering, forgery and counterfeiting. Encryption and decryption algorithm's security depends on the algorithm while the internal structure of the rigor of mathematics, it also depends on the key confidentiality. Key in the encryption algorithm has a pivotal position, once the key was leaked, it means that anyone can be in the encryption system to encrypt and decrypt information, it means the encryption algorithm is useless. Therefore, what kind of data you choose to be a key, how to distribute the private key, and how to save both data transmission keys are very important issues in the encryption and decryption algorithm. This paper proposed an implementation of a complete and practical RSA encrypt/decrypt solution based on the study of RSA public key algorithm

III PROPOSED ALGORITHM

3.1 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
$[M \oplus G(r)] \| [r \oplus H(M \oplus G(r))]$
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.

3.2 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 ciphertexts 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 increasein 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 moremachines, increasingly powerful machines, or more calendar time. The calendar time required for thefactorization of RSA-512 was 3.7 months . Other issues like the cost and availability of memorymay 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.

3.3 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.[1]

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 $X \parallel Y$ 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.

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 ciphertext 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).

IV RSA Encryption with Proposed Methodology

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 = m^e mod n = 7^3 mod 33 = 343 mod 33 = 13.
Hence the ciphertext c = 13.
Step 6: To check decryption we compute
m' = c^d mod n = 13^7 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' = 13^7 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 ciphertext 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 ciphertext integers c1, c2, ...

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

(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

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

n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

V 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
```

VI. RESULT AND DISCUSSION

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

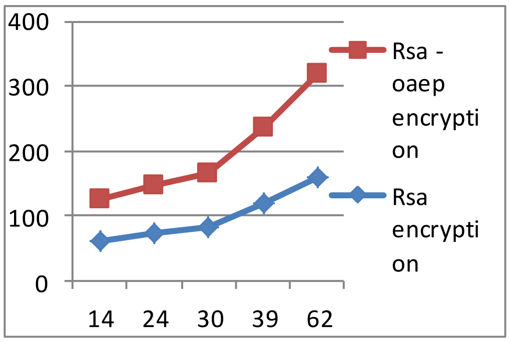
6.1 RSA ENCRYPTION DECRYPTION

6.2 RSA-OAEP ENCRYPTION DECRYPTION

6.3 RSA AND RSA – OAEP ENCRYPTION DECRYPTION

When comparing with RSA , RSA – OAEP algorithm requires more time for encryption decryption. Where as 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 encrption and decryption algorithm.

DOI: 10.18535/ijecs/v5i9.48



VII 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.More over , this scheme can be used for encryption of long messages woithout 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] Cormen , Thomas H, Charles E.Leiserson, aronald L.Rivest Clifford Stein "Introduction to algorithms". MIT Press and McGraw-Hill

[3] A. J. Menezes, P. C. Van Oorschot and S. Vanstone,"*Handbook of Applied Cryptography*", CRC Press, Boca Ration, Florida, USA, 1997.

[4] B.Schneier,"*Applied Cryptography*", 2nd ed, John Wiley & Sons Inc., New York, 1996.

[5] William Stallings, "*Cryptography and Network Security*", 2nd ed, Prentice Hall,Upper saddle River,new Jersey, USA, 1997.

[6] William Stallings "*Cryptography and Network Security Principles and Practice*", 4th ed, Prentice Hall, 2006

E.Vinithra, received her B.Sc(Computer Science)degree in Nehru Memorial college form Bharathidhasan university,Trichy [Tamilnadu(India)](2007-2010).She received her MCA degree in Kongunadu Arts and Science from Bharathiyar university, Coimbatore[Tamilnadu(India)](2010-2013).She is the M.phil Research scholar of Muthayammal college of Arts and science form Periyar university, Salem [Tamilnadu(India)].Her area of interest is Networking.

N.Hyrunnisha, received her B.sc degree in Nehru Memorial college Form Bharathidhasan university, Trichy [Tamilnadu(India)].She received her MCA degree in university of Madras, chennai [Tamilnadu(India)].She is the M.phil Research scholar of periyar university, salem [Tamilnadu(India)].she is assiant Professor of Computer Application in Muthayammal college of Arts and science Form periyar university, salem [Tamilnadu (India)].she is 8 year work of experience in Muthayammal College of Arts and science .