Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 3 – No.1, October 2015 – www.caeaccess.org

Contemporary RSA- 1024 Cryptosystem: A Comprehensive Review Article

Sanjeev Karmakar
Bhilai Institute of Technology, Durg, Bhilai House, 491001, Chhattisgarh, India

Siddhartha Choubey
Shri Shankaracharya Technical Campus / Shri Shankaracharya group of Institutions, Bhilai, Durg, India

ABSTRACT
Security strength of RSA Cryptography is an enormous mathematical integer factorization problem. Deducing the private key'd' from its equation e. d ≡ (1 mod ψ) where ψ = (p-1). (q-1), £ n Є I+, such that n = p. q; is a world wide effort. This paper introduced very significant integer factoring algorithms such as trial division, ρ- method, ECM, and NFS and effort to factor RSA-150 composite number 'n' of 512 bits by using NFS. It is found that the 512 bit RSA number may be believed to safe from the intruder. However, this system is slow for large volume of data. The computation of c ≡ me mod n required O ((size e)(size n)* (size n)) and space O(size e + size n). Similarly, decryption process also has required O ((size d) (size n) * (size n)) and space O (size d + size n). Java 'BigInteger' class is introduced to overcome this shortcoming and successfully applied is presented through this paper.

Keywords
RSA, RMI, Cryptography, Encryption, Decryption, Network, Security, RSA-1024, NFS, ECM

1. INTRODUCTION
Cryptology (from the Greek *kryptós lógos*, meaning ``hidden word'') is the discipline of cryptography and cryptanalysis combined. To most people, cryptography is concerned with keeping communications private. Indeed, the protection of sensitive communications has been the stress of cryptography throughout much of its history [1]. Cryptography has two phases' encryption and decryption.

Encryption is the transformation of data into a form that is as close to impossible as possible to read without the suitable knowledge (a key). Its reason is to ensure privacy by keeping information hidden from anyone for whom it is not proposed, even those who have access to the encrypted data. *Decryption* is the reverse of encryption; it is the transformation of encrypted data back into an original form. Encryption and decryption generally require the use of some secret information, referred to as a *key*. For some encryption mechanisms, the same key is used for both encryption and; for other method, the keys used for encryption and decryption is different. There are two types of cryptosystems: *secret-key* and *public-key cryptography*. In secret-key cryptography, also referred to as symmetric cryptography, the same key is used for both encryption and decryption. The most popular secret-key cryptosystem in use today is the *Data Encryption Standard* (*DES).* In public-key cryptography, each user has a *public key* and a *private key*. The public key is made public while the private key remains secret. Encryption is performed with the public key while decryption is done with the private key. The *RSA public-key cryptosystem* is the most popular form of public-key cryptography. RSA stands for Rivest, Shamir, and Adleman, the inventors of the RSA cryptosystem. The *Digital Signature Algorithm* (*DSA)* is also a popular public-key technique, though it can only be used only for signatures, not encryption. *Elliptic curve cryptosystems* (*ECCs)* are cryptosystems based on mathematical objects known as elliptic curves. Elliptic curve cryptography has been gaining in popularity recently. Lastly, the *Diffie-Hellman key agreement protocol* is a popular public-key technique for establishing secret keys over an insecure channel [2].

Surveys by Rivest [3] and Brassard [4, 5] form an excellent introduction to modern cryptography. Some textbook treatments are provided by Stinson [6] and Stallings [7], while Simmons provides an in-depth coverage of the technical aspects of cryptography [8]. A comprehensive review of modern cryptography can also be found in Applied Cryptography [9]; Ford [10] provided detailed coverage of issues such as cryptography standards and secures communication.

This paper explained one of the most important public key cryptography called RSA cryptography and its practical execution by using Java in the following sections intended to provide more practical observation of this cryptosystem for the reader. Theoretical concept of RSA cryptography, example of RSA-1024 cryptography, its implementation RSA-1024 cryptography using java, security strength, shortcoming, research scope, and finally conclusions have been discussed.

1.1 RSA Cryptography
The RSA cryptosystem is a public-key cryptosystem that offers both encryption and digital signatures (authentication). Ronald Rivest, Adi Shamir, and Leonard Adleman developed the RSA system in 1977 [11]; RSA stands for the first letter in each of its inventors' last names. Its security is based on the intractability of the integer factorization on the problem [1, 2] will be discussed in this chapter afterward sections. In RSA algorithm encryption and decryption process is depicted in the following Fig.1 (a,b, & c) and its processes is explained in the following section.

To generate public key encryption RSA algorithm selects two large prime numbers p,q such that n=p. q and ψ = (p –1). (q – 1), and £ e Є I^+, 1<e< ψ, such that gcd (e, ψ) = 1. After that algorithm used Euclidean algorithm to compute the unique integer d, 1<d< ψ, such that e. d ≡ (1 mod ψ). Through that £ public key (n, e) and private key is d. these e and d is RSA key generation are called the encryption exponent and the

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 3 – No.1, October 2015 – www.caeaccess.org

decryption exponent respectively, while n is called the modulus.

To send message (m), where $m \in I^+$, £ $m \in [0, n – 1]$ interval, compute cipher text $c \equiv m^e \pmod n$ at sender end this task is known as *encryption*. At the receiver and again compute $c^d \pmod n \equiv m$ to get the original message (m), this task is known as *decryption*. To proof decryption task, since $e.\ d \equiv 1(\mathrm{mod}\ \psi)$, £ $k \in I$ such that $e.\ d = 1 + k\ \psi$. Now if gcd (m, p) = 1 then by Format's theorem mp-1 = mod p. Raising both side of this congruence to the power of k(q-1) and then multiplying both side by m yields

$$m^{1+k\,(p-1)(q-1)} = m \pmod p$$

On the other hand, if gcd(m,p) = p, then this last congruence is again valid since each side congruence to 0 modulo p.

Hence in all cases

$$m^{ed} \equiv m(\bmod\ p)$$

By same argument $m^{ed} \equiv m(\mathrm{mod}\ q)$. Finally, since q and q are distinct primes, it follows that $(m^e)^d \equiv m \pmod n$; and hence

$$e^d \equiv (m^e)^d \equiv m \pmod n.$$

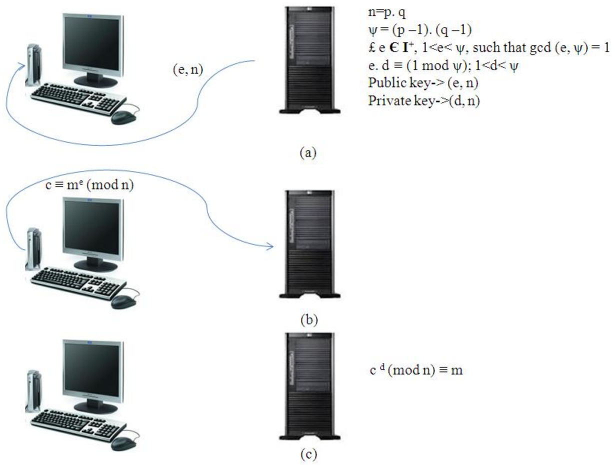

Fig. 1. RSA Cryptography

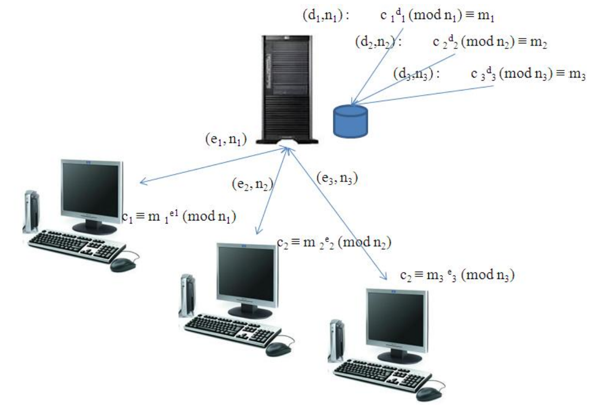

Fig. 2. Multi-client Server RSA Cryptography System

Here, single client is connected with the server. In general, for all computer network applications, often multiple clients are connected with sever. For multiple client server applications, it is noted that server computes unique public as well as private key for each connected clients as depicted in Fig. 2. It shows how server will send unique public key for each connected clients and maintain unique private key for each client as well. At this juncture, three clients are connected with the server is shown. Server will generate three set of public and private key for each connected clients. It is cleared that, each client have their own pair of key for the secure communication. This feature produces security potency of the system as well. In the subsequent sections of the chapter, detail design and Java code of RSA-1024 cryptography system over a multi-client server network is discussed. Readers of this book may use this code and exhibit the system in their machine or LAN. In the next section, an example of RSA-1024 cryptography is explained.

1.2 Paradigm of RSA -1024 Cryptography

RSA-1024 has 1,024 bits (309 decimal digits) have following steps in the server as well as client side.

```
Step-I:     Server   randomly   generates
two large prime number say p and q like -

p =
      19723548993819989754922565271561472
      94395422515733970967431625499248959
      91307983563222686430053692168812384
      84911627627177620887034862330061284
      3029900414459
q =
      31390070755831755004012405854969206
      97142967178123204474484712102553171
      92427751191891281347184368895910516
      45832273346080086600307934802998541
      1066089762133937
Step-II: Server computes n = p.q (called
            RSA Modulus 'n' is a 1024
            bits number has 309 decimal
            digits)
n =
      61912359847212369671288806246260138
      12212052387289151652664227575355377
      71531495613333037401658612544833474
      00619314792687346296536446518008009
      38861490432161507645110125287810122
      82211662158055141882953539502644281
      80250806507668549402775952254344988
      55114665852970853971989413138402658
      2757627117281192440569395083

Step-III: Server computes ψ = (p –1). (q
–1)

Ψ =
      61912359847212369671288806246260138
      12212052387289151652664227575355377
      71531495613333037401658612544833474
      00619314792687346296536446518008009
      38861490432130097850805299712816355
      49369638081210704520352900564198829
      65878001265580130343600497750311374
      12855857920086536787015705874181463
      3060973831257283320906846688

Step-IV: Server computes 'd' by relation
d ≡ 1(mod ψ)
d =
```

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 3 – No.1, October 2015 – www.caeaccess.org

41274906564808246447525870830840092
08141368258192767768442818383570251
81020997075555358267772408363222316
00412876528458230864357631012005339
59240993621420065233870199808544236
99579758720807136346901933709465886
43918667510386753562400331833540916
08570571946724357858010470582787642
2040649220838188880604564459

Step IV: Sets public key (e, n) like (3, n) and privet key (d, n)

Step V: Sends (e, n) to the client for encryption and maintains (d, n) for decryption.

Now consider following plaintext is to be sent from the client

Dear Aditya
RSA cryptosystem is based on Integer factorization problem thus produced high security however slow for large data.
Pradeep, K.,

Then message (m) can represent as big integer form

m =
19758115125724582576198000160674597
12583016966761746032370057093722430
51963748277169849190956858887451547
43642907299950510449042848568217693
32249140945779889887852065883208320
23348848960325963245453068018865124
07734093472769434986087213624552834
22288542914198429475157302869360615
56047504052969154914494631198585824
01904052641802207209205519206867694
0028046666867237965 24

Encryption- Client computes cipher text($C=m^e$ mod n) and sends to the server:

(19758115125724582576198000160674597
71258301696676174603237005709372243
05196374827716984919095685888745154
74364290729995051044904284856821769
33224914094577988988785206588320832
02334884896032596324545306801886512
40773409347276943498608721362455283
42228854291419842947515730286936061
55604750405296915491449463119858582
40190405264180220720920551920686769
4002804666686723796524)^{3} mod
(6191235984721236967128880624626013
81221205238728915165266422757535537
77153149561333303740165861254483347
40061931479268734629653644651800800
93886149043216150764511012528781012
28221166215805514188295353950264428
18025080650766854940277595225434498
85511466585297085397198941313840265
82757627117281192440569395083
=
44326296821353505391897989919853592
62047511266314161727099616609282580
27246426386714426724099435622357224
30512750635517157672727259076937736
45082205223910300016772518791028648
44076863964179858259819887729695550
99915036539903477778404270933912857
45188389337258374685111141236226317
04324047269764743641754473

Decryption –Server computes original message (m) by using ($m= C^d$mod n) as

(4432629682135350539189798991985359
26204751126631416172709961660928258
02724642638671442672409943562235722
43051275063551715767272725907693773
64508220522391030001677251879102864
84407686396417985825981988772969555
09991503653990347777840427093391285
74518838933725837468511114123622631
704324047269764743641754473)^{41274906564808246447525870830840092081413682581927677684428183835702518102099707555535826777240836322231600412876528458230864357631012005339592409936214200652338701998085442369957975872080713634690193370946588643918667510386753562400331833540916085705719467243578580104705827876422040649220838188880604564459} mod
(6191235984721236967128880624626013
81221205238728915165266422757535537
77153149561333303740165861254483347
40061931479268734629653644651800800
93886149043216150764511012528781012
28221166215805514188295353950264428
18025080650766854940277595225434498
85511466585297085397198941313840265
82757627117281192440569395083)
=
19758115125724582576198000160674597125830
16966761746032370057093722430519637482771
69849190956858887451547436429072999505104
49042848568217693322491409457798898878520
65883208320233488489603259632454530680188
65124077340934727694349860872136245528342
22885429141984294751573028693606155604750
40529691549144946311985858240190405264180
22072092055192068676940028046666867237965
24.

2. IMPLEMENTATION OF RSA-1024 BY USING JAVA

Java BigInteger provided analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations. Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. It has been found the implementation of RSA algorithm can be easily obtained from following code segment. Detail of implementation procedure as well as code is given in the subsequent sections of this chapter.

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 3 – No.1, October 2015 – www.caeaccess.org

```
int size1 = size/2;
int size2 = size1;
int offset1 =
(int)(5.0*(rnd.nextDouble()) +
5.0);
 int offset2 = -offset1;
 if (rnd.nextDouble() < 0.5) {
offset1 = -offset1; offset2 = -
offset2;}
size1 += offset1; size2 += offset2;
/* generate two random primes, so
that p*q = n has size bits */
BigInteger p1= new
BigInteger(size1, rnd);
// random int
p= nextPrime(p1);
BigInteger pM1=
p.subtract(BigInteger.ONE);
BigInteger q1 = new
BigInteger(size2, rnd);
q= nextPrime(q1);
BigInteger qM1=
q.subtract(BigInteger.ONE);
n= p.multiply(q);
BigInteger phiN= pM1.multiply(qM1);
// (p-1)*(q-1)
BigInteger e= THREE;
d= e.modInverse(phiN);
```

Where rnd is random number is obtained by Random class which is defined in java.util package. And next prime is obtained from following piece of code.

```
public BigInteger
nextPrime(BigInteger x)
{
if
((x.remainder(TWO)).equals(BigInteg
er.ZERO))
x = x.add(BigInteger.ONE);
while(true)
{
BigInteger xM1 = x.subtract
(BigInteger.ONE);
if
(!(xM1.remainder(THREE)).equals(Big
Integer.ZERO))
if (x.isProbablePrime(10)) break;
x = x.add(TWO);
}
      return x;
}
}
```

Encryption and Decryption is obtained from following code segment

```
public BigInteger RSADecrypt(BigInteger
c)
{
        return c.modPow(d, n);
}
public BigInteger RSAEncrypt(BigInteger
c)
{
        return m.modPow(e, n);
}
```

3. SECURITY STRENGTH OF RSA CRYPTOGRAPHY

Security strength of RSA is enormous mathematical integer factorization problem. Deducing the private key 'd' from its equation e. d ≡ (1 mod ψ) where ψ = (p-1). (q-1), £ n Є I+, such that n = p. q (p and q is large distinct prime number) is a world wide effort. It required more processing power along with storage. This paper introduced significant integer factoring algorithms number field sieve (NFS) and its effort to factor while The RSA-150 (512 bit) composite number 'n' have been considered. Other significant mathematical methods/algorithm also has been discussed for assessment through survey of Mathematical Review 94-2008A60. It has been found that the 512 bit RSA number may be believed to safe from the intruder.

4. POSSIBLE ATTACK ON RSA

If the advisory is able to factor the public modulus 'n' of same entity A, then the adversary can compute ψ and then, using Extended Euclidean algorithm deduce the private key d from ψ and the public exponent 'e' by solving ed ≡ 1(mod ψ). This constitutes a total break of the system. To guard against this, A must set p and q so that factoring 'n' is a computationally infeasible task [12].

4.1 Methodology

One of the importation applications of integer factorization is RSA public key cryptosystem. The security of cryptosystem depends on the intractability of factoring integers [13]. The integer factorization is one of the problems that have been long considered in the world of the number theory [12]. In the last few decades, together with the rapid progress of computer technology, methods for factoring integers efficiently were studied, and as a result some algorithms were invented through review. The major methodology or algorithm through which'd' can deduce from $e^d \equiv 1 (\mathrm{mod}\ \psi)$, if we capable to factor n. In fact it is strength of RSA security; nobody can say which algorithm is best for factor 'n' on the word of computational complexity.

Dixon's method based on finding a congruence of squares. Format's factorization algorithm finds such congruence by selecting random or pseudo-random x value and hoping, one satisfied the congruence. $X^2 \equiv y^2 \pmod n$; $X = \underline{+}\ Y$; values will take an impractically long time to find a congruence of squares. In Pollard's (ρ-1) algorithm meaning that it is only suitable for integers with specific types of factors. The algorithm is based on the insight that number of the form (a^b-1) tends to be highly composite when b is itself composite. Since it is computationally simple to evaluate number of this form in modular arithmetic, the algorithm allows one to quick check many potential factors with greatest efficiency. In particular the method will find a factor ρ if b is divisible by ρ -1. When ρ -1 is smooth (the product of only small integer) then this algorithm is well suited to discovering the factors.

The *trial division* is the simplest and easiest to understand of the integer factorization algorithm. Given a composite integer n, *trial division* consists of trial dividing n by every prime number less than or equal to √n. If a number is found which divides evenly into n, a factor of n has been found. Trail division is guaranteed, to find a factor of n, since it checks all possible prime factors of 'n'. Then if the algorithm is fail, it is proof. That n is prime. In worst case, trial division is very inefficient algorithm. It starts from 2 and work up to the $\sqrt{n^2}$, the algorithm requires ($\pi\sqrt{n}$) trial divisions, where $\pi(x)$ is the number of primes less than *x*.

NFS (Number field sieve) algorithm uses four main steps; polynomial selection, sieving, linear algebra and square root. In polynomial section step two irreducible polynomial $f_1(x)$ and $f_2(x)$ which are commonly root m (mod N) selected having as many as practically possible smooth values over given factor base. In the sieving steps, which is by far the most time consuming steps of NFS pair (a,b) are found with gcd (a,b) =1 such that both

$$b^{\deg(f1)} f_1(a/b) \;\&\; b^{\deg(f2)} f_2(a/b)$$

Are smooth over given factor base i.e., factor completely over the factor bases. Such a pair (a,b) called relation. The purpose of this step is to collect so many relations that several subsets S of them can be found with the property that a product taken S yield on expression of the form $X^2 = Y^2$ (mod N) for approximately half of these subsets, computing gcd (X-Y, N) yields a non-trial factor of N (if N has exactly to distinct factors). We only discussed sieving steps in NFS here [14-16].

ECM (Elliptical curve method) by Lenstra, and Takayuki [17,18] makes use of property of Groups (G) of points on the elliptic curve to find factor of composite number n. ECM can find relatively small factors (< 50 digits) of so lange integers that NFS cannot treat, because ECM does not have a limitations with respect to n. For this reason ECM is still an important technique for factorization at the present time [19,20]. Lenstra, [21], and since than various improvements has been worked out. The time complexity of some algorithm is dependent on the given composite number n, and that other is dependent on the smallest prime factor p of n. As per algorithms discussed above, the average (or worst time) complexity is shown below [21, 22]-

NFS	:	$L_n[1/3, 1.901]$
Trial Division	:	$O(\rho)$
ρ method	:	O(under root of ρ)
ρ -1 method	:	$O(\rho')$
ECM	:	$L_p[1/2, 1.414]$

Here(,ρ' is the largest prime division of (ρ -1). Here the function $L_x(1,c)$ defined as follows:

$$L_x(1,c) = \exp((c + O(1)\,(\log x)^I (\log\log x)^{I-1}))$$

And an algorithm that has the time complexity $L_n(I,c)$ for some c and I<1 (where, n is the input integer) is called a sub exponential time algorithm. Note that $L_n[O,C] = O(\log n)^c$ (polynomial time) and $L_n[1,c] = O(n^c)$ (exponential time with respect to the input length log n) [21]. It is concluded that the best-known algorithm for factoring integer is NFS, asymptotically and practically for very large composite number (over 150 digits). The sieving phase that search fixed set of prime number for candidate that have a particular algebraic relationship, modulo the number to be factor. The sieving phase can be done in distributed fashion, on large number of processors simultaneously. The matrix-solving phase require large amount of storage space [23-25] The RSA-150 (512 bit) composite number 'n' have been considered for factor by assuming NFS algorithm wherein n = p q while for the sieving steps 200000 parallel Pentium-4 microprocessors (2.53 GHz), FSB 53 MHz, Intel Desktop Motherboard D850EV2, i850 chip set, 1024 MB RAM, PC800, Free BSD were used [25].

```
n=
    1550898124283484405096067543700118
    6177065454583099543065546694577431
    2632703463465954363335027577729025
    3914539967847140270035016317721868
    40890795964683
```

After factor through NFS obtained

```
p =
    3480098671022836954839704510475934
    2483101281735038545688955963754827
    810717
q =
    4456477449036407415332411257870861
    7600544253629776615349341972453246
    0296199
```

The estimated sieving time to factor above RSA-150 (512 bit) composite number 'n' was 20597260 second, approximately of 239 days. This work supported by the CRIPTREC project is promoted by Telecommunication advancement organization of Japan. (mailto: macro@ntt.co.jp) [25]. Zimmermann, P., is a French computational mathematician, working at INRIA (The National Institute for Research in Computer Science and Control (French: *Institut national de recherche en informatique et en automatique*,INRIA). His interests include asymptotically-fast arithmetic — he wrote a book [26] on algorithms for computer arithmetic with Richard Brent. He has developed some of the fastest available code for manipulating polynomials over GF(2), and for calculating hyper geometric constants to billions of decimal places. He is presently associated with the CARAMEL project (http://caramel.loria.fr/members.en.html) to develop efficient arithmetic, in a general context and in particular in the context of algebraic curves of small genus; arithmetic on polynomials of very large degree turns out to be useful in algorithms for point-counting on such curves. He factored RSA -704 on July 2, 2012 [26]. The CARAMEL project-team has three main research themes [27]:

1. The number field sieve algorithm and its siblings, for integer factorization and discrete logarithm in finite fields,
2. Algebraic curves for cryptography, in particular genus 2 curves and pairings,
3. Arithmetic in general, from integers to floating-point numbers, in software and hardware.

Thus, it is concluded that, even 'n' is obtainable by the intruder however, deducing the private key'd' from its equation e. d ≡ (1 mod ψ) where ψ = (p-1). (q-1), £ n Є I^+, such that n = p. q (p and q is large distinct prime number) is almost complicated. Currently RSA modulus 'n' of size 1024 bits is being used for all protocols/applications [28, 29].

5. SHORTCOMING AND OVERCOME BY JAVA BIGINTEGER.

It is observed that this system is quite slow for large volume of data. Foundation of this shortcoming is, in RSA public key 'e' is an integer and message m Є{0,1,2,3,4…..n-1} to be encrypted therefore the computation of "m^e mod n" required processing time O((size e)(size n $)^2$) and space O(size e + size n). Similarly, cause of slow for large volume of data is again complexity occurred during the decryption process to compute original message from cipher text c. The complexity in the decryption process is O ((size d) (size n) ^{2}). Means the computational complexity is directly proportional to the multiplication of (size e) and (size n) ^{2} while 'm' is a large integer to be send. In other hand complexity is directly proportional to the multiplication of (size d) and (size n) ^{2} during the decryption process while 'c' is a large integer. It may be 1024 bits or more. The time and space complexity of both the operations is given in the Tab. 1. This problem can solve by fast exponentiation algorithm [1]. Many researchers are used this algorithm to do so. However, it is found that, Java BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods as discussed in the section 1.3. The encryption and decryption is discussed as :

```
return m.modPow(e, n);
return c.modPow (d, n);
```

This feature of Java is applied instead of fast exponentiation algorithm to overcome the shortcoming.

Table 1. Complexity in RSA cryptography

Process	Time	Space
Encryption	O((size e)(size n)^2	O (size e + size n).
Decryption	O((size d)(size n)^2	O (size d + size n).

6. CONCLUSIONS

RSA cryptosystem is presently used in a wide variety of products (TCP/IP, MIME, WAN, TELNET etc), platform (Apple, Sun, Novel, and Microsoft) around the world computer network for safely communication and transformation. This paper discussed the comprehensive view of the RSA cryptosystem, its straight, limitations, and various methods such as trial division, ρ- method, ECM, and NFS for breaking RSA system. It is found that RSA-1024 is absolutely secure from the intruder, however has introduced more computational complexity while increasing public key (e, n) and private key (d, n). As far as concern to solve this problem one can solve by fast exponentiation algorithms. However many researchers are often used these algorithm. It is found that, Java BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods. This feature of Java is successfully applied on the algorithm instead of fast exponentiation algorithm to overcome this shortcoming.

7. REFERENCES

[1] Kahn, D., 1967: The Codebreakers, Macmillan Co., New York.

[2] Johannes, A., Buchmann, 2000: Introduction To Cryptography, Springer-Verlog, Berlin, pp. 45-50, New York.

[3] Rivest, R.L., 1990: Cryptography, Handbook of Theoretical Computer Science, volume A (editor: J. van Leeuwen), MIT Press/Elsevier, Amsterdam, pp.719-755.

[4] Brassard, G., 1988: Modern Cryptology, Springer-Verlag.

[5] Brassard, G., 1993: Cryptography column - Quantum cryptography: A bibliography, Sigact News (3) 24, pp.16-20.

[6] Stinson, D.R., 1995: Cryptography - Theory and Practice, CRC Press, Boca Raton.

[7] Stallings, W., 1995: Network and Internetwork Security - Principles and Practice, Prentice-Hall, New Jersey.

[8] Simmons, G.J., 1992:, Contemporary Cryptology - The Science of Information Integrity, IEEE Press.

[9] Schneier, B.,1995: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition, Wiley.

[10] Ford, W., 1994: Computer Communications Security Principles, Standard Protocols and Techniques, Prentice-Hall, New Jersey.

[11] Rivest, R.L., Shamir, A., Adleman, L.M., 1978:A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM (2) 21, pp.120-126.

[12] Cavallar, S., Dodson, B., Lenstra, A.K., Lion ,W., Mongomery, P.L., Murphy, B., Riele, H., ,2000: Factorization of RSA Modulus , Eurocrypt 2000, Bruges, Belgium, May 14-18.

[13] RSA Data Security Corporation Inc. Sci. Crypt. Dec 18, 2004: Information available by sending E-mail : challenge-rsa-list@rsa.com.

[14] Buhler, J.P., Lenstra Jr H.W., Pomerance, C., 1993: Factoring Integer with NFS, Lecture notes in Maths , 1554, Springer-Verlog, Berlin, pp. 50-94.

[15] Lenstra, A.K., Lenstra, H.W., 1990:The Number Field Sieve, Proc., 22nd STOC, pp- 564-572.

[16] Aoki,K., Ueda, H., Kida, Y., 2004: NFS Factoring Statistics, NTT Labs, Rikkuo University, Japan

[17] Lenstra, H.W. Jr, 1987: Factoring Integers with Elliptic curve, Annals of Maths., pp. 649-673.

[18] Takayuki yato, 2006: Study on ECM for integer factorization, A seminar thesis, Department of Information Science, The university of Tokyo, 15 Feb 2006.

[19] Koblitz, N.,1989: Hyperelliptic Cryptosystem, J.Cryptology, Vol.1, pp. 139-150.

[20] Sakai, Y.,Sakurai, K., 1998, Design of Hyperelliptic Cryptosystem in small characteristics and software implementation, ASIACRYPT'98, LNCS, Vol. 1514, pp. 80-94.

[21] Lenstra, A. K., 2000: Integer Factoring, Design code and Cryptography, 19, Kluwer Academic publisher, pp.101-128, Boston, Netherland.

[22] Cavallar, S., Dodson, B., Lenstra, A.K., Lion ,W., Mongomery,P.L., Murphy, B., Riele,H., 2000: Factorization of RSA Modulus , Eurocrypt 2000, Bruges, Belgium, May 14-18,

[23] Lenstra, A.K., Lenstra, H.W., 1990: The Number Field Sieve, Proc., 22nd STOC, pp.564-572.

[24] Buhler, J.P., Lenstra Jr H.W., Pomerance, C., 1993: Factoring Integer with NFS, Lecture notes in Maths , 1554, Springer-Verlog, Berlin, pp. 50-94.

[25] Aoki,K., Ueda,H., Kida, Y., 2004: NFS Factoring Statistics, NTT Labs, Rikkuo University, Japan..

[26] Zimmermann, P., Cheng, H., Hanrot, G., Thomé, E., Zima, E., 2007: Time- and Space-Efficient Evaluation of Some Hypergeometric Constants. In C W Brown. Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC) 2007. pp. 85–91.

[27] CARAMEL Project, URL: http://caramel.loria.fr/index.en.html.

[28] RSA Laboratories, The RSA Factoring Challenge. Retrieved on 2008-03-10. URL: http://en.wikipedia.org/wiki/RSA_Factoring_Challenge.

[29] RSA Laboratories, The RSA Factoring Challenge FAQ. Retrieved on 2008-03-10.