Int. J. Advanced Networking and Applications
Volume: 01, Issue: 04, Pages: 210-216 (2009)

Modified Trail division for Implementation of RSA Algorithm with Large Integers

Satyendra Nath Mandal
Dept. of I.T, Kalyani Govt. Engg College, Kalyani ,Nadia(W.B),India
Email: satyen_kgec@rediffmail.com
Kumarjit Banerjee
CSE, Kalyani Govt. Engg College, Kalyani, Nadia(W.B),India
Email: kumarkgec123@gmail.com
Biswajit Maiti
Dept. of Mathmatics, Kalyani Govt. Engg College, Kalyani ,Nadia(W.B),India
Email: bmkgec@gmail.com
J. Palchaudhury
Dept. of I.T, Kalyani Govt. Engg College, Kalyani ,Nadia(W.B),India
Email: jnpc193@yahoo.com

ABSTRACT

The RSA cryptosystem, invented by Ron Rivest, Adi Shamir and Len Adleman was first published in the August 1978 issue of ACM[4]. The cryptosystem is most commonly used for providing priva cy and ensuring authenticity of digital data. The security level of this algorithm depends on chooing two large prime numbers. But, to handle large prime in personal computer is huge time consuming. Further, each and every compiler has a maximum limit to integer handling capability. In this paper, an approach has been made to modify trial division technique for implementation of RSA algorithm for large numbers beyond the range of a compiler that has been used to implement it. The time complexity of this modified trial division method has been calculated using personal computer, at the end for large integer.

Keywords - **RSA cryptosystem, Prime Number, Trail division, Time Complexity.**

Date of Submission: December 03, 2009 Revised: January 30, 2010 Accepted: February 17, 2010

1. INTRODUCTION

The requirements of information security within an organization have undergone two major changes in the last few decades. With the introduction of the computer the lead of automated tools for protecting files and other information stored on the computer became evident, especially the case for a shared system. No one can deny the importance of security in data communication and networking. Security in networking is based on cryptography [7] [8], the science and art of transforming messages to make them secure and immune to attack. The RSA algorithm is the most popular and proven asymmetric key cryptographic algorithm [3]. For larger the primes [9], tougher is the factorization [1], [2]. This makes the RSA secure. From the study, it is evident that lots of work has been done to detect and handle large prime in RSA algorithm [12]. One of them is trail division method. In this paper, some modification has been done on trail division method. The first requirement of RSA algorithm is to choose two prime numbers. It can be done by taking two numbers as string and to check whether they are prime, modified trial division algorithm can be used for this purpose. To do this, first requirement is to requirement compute the length of the string, if it is less than 2*n-6 where n denotes the maximum number of decimal digits that a particular compiler supports then to convert the string into array of integers else to subtract iteratively the numbers for which the given string has to be compared until the reminder is less than the number which is the required modulus. If all the moduli computed are non-zero then the number is prime. After getting the two primes the product n=p*q is computed by means of adding the partial products. For a chosen e the gcd(e, f) is computed. If it is equal to 1 then d is generated else the user is asked to choose another e. Finally, by using p,q,n,e and d RSA algorithm has been developed. This modified trial division method will be much useful in handling large primes to be used in RSA.

2. RSA ALGORITHM

The RSA algorithm involves three steps: key generation, encryption and decryption [10].

2.1 Key Generation

RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The keys for the RSA algorithm are generated the following way:

Choose two distinct prime numbers p and q. For security purposes, the integers p and q should be chosen uniformly at random and should be of similar bit-length. Prime integers can be efficiently found using a Primality test. Compute $n = p*q$. n is used as the modulus for both the public and private keys. Compute the totient: $f(n) = (p-1)*(q-1)$. Choose an integer e such that $1<e< f(n)$, and e and $f(n)$ are coprime. e is released as the public key exponent. Choosing e having a short addition chain results in more efficient encryption. Determine d (using modular arithmetic) which satisfies the congruence relation $d*e = 1(\text{mod } f(n))$. d is kept as the private key exponent. The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d which must be kept secret.

2.2 Encryption

Alice transmits her public key (n,e) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice [12], [10], [5]. He first turns M into an integer $0<m< n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c corresponding to: $c=m^e(\text{mod } n)$. This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice.

2.3 Decryption

Alice can recover m from c by using her private key exponent d by the following computation: $m=c^d(\text{mod } n)$.

Given m, she can recover the original message M by reversing the padding scheme.

The above decryption procedure works because:

$m=(m^e)^d(\text{mod } n) = m^{ed}(\text{mod } n)$.

Now, since $e*d=1+k*f(n)$,

$m^{ed}=m^{1+k*f(n)} = m*(m^k)^{f(n)} = m(\text{mod } n)$

The last congruence directly follows from Euler's theorem when m is relatively prime to n. By using the Chinese remainder theorem it can be shown that the equations hold for all m. This shows that the original message is retrieved:

$c^d=m(\text{mod } n)$.

3. METHODS FOR ARITHMETIC OPERATIONS OF TWO LARGE NUMBERS

3.1 Addition

Step 1 Take two numbers as Strings as input.

Step 2 Compute the length of two Strings.

Step 3 If the lengths are equal go to Step 4 else add zeros in front of the String of smaller length.

Step 4 If the lengths of two Strings are equal add a zero to each String which will handle if there is a carry.

Step 5 Take another two arrays of integer of the length equal to the present length of the Strings. Initialize one of them to all zeros which will hold the carry if any.

Step 6 The elements of the array which will hold the sum, is obtained by adding the elements of the initial two integer arrays and the carry array.

Step 7 The carry array elements are obtained by the operation as carry [i-1] = (a[i] + b[i])/10 where I denotes the index.

Step 8 Convert the array of sum to String and return.

3.2 Subtraction

Step 1 Take two numbers as Strings as input.

Step 2 Compute the length of two Strings.

Step 3 If the lengths are equal go to Step 4 else add zeros in front of the String of smaller length.

Step 4 Take another array of integers of the number elements equal to the length of the Strings at present.

Step 5 If there is a borrow, subtract one from the previous indexed element if it is greater than zero else set the previous element to 9 and continue Step5 until there is any element greater than zero.

Step 6 Convert the result obtained to String and return.

3.3 Multiplication

Step 1 Take the two numbers as Strings as input.

Step 2 Convert the Strings into array of characters and subsequently into array of numbers.

Step 3 Compute partial products for each of the element and add the partial product to a variable initially set to zero.

Step 4 The partial product is computed with the above addition algorithm.

Step 5 The final sum is the required product.

3.4 Division

3.4.1 Quotient

Step 1 Take the two numbers as Strings as input.

Step 2 If the length of the first (l1) to the second (l2) String differs by 1or less compute quotient by using a loop which counts the number of iterations for the subtraction of divisor from the dividend else go to Step 3.

Step 3 Take the substring of first l2+1 characters of the first String. Compute the quotient by using a loop which counts the number of iterations for the subtraction of divisor from the present dividend of length l2+1, and find the remainder.

Step 4 Concatenate the next positioned character in the first String to the remainder and find the quotient for the second String and the new String obtained.

Step 5 Concatenate the quotient obtained to the previous quotient. Compute remainder.

Step 6 Repeat Steps 4 and 5 until no characters left for first String.

Step 7 The quotient obtained in the final step is the required quotient.

3.4.2 Remainder

Step 1 Take the two numbers as Strings as input.

Step 2 If the length of the first (l1) to the second (l2) String differs by 1or less compute quotient by using a loop which counts the number of iterations for the subtraction of divisor from the dividend else go to Step 3.

Step 3 Take the substring of first l2+1 characters of the first String. Compute the quotient by using a loop which counts the number of iterations for the subtraction of divisor from the present dividend of length l2+1, and find the remainder.

Step 4 Concatenate the next positioned character in the first String to the remainder and find the quotient for the second String and the new String obtained.

Step 5 Concatenate the quotient obtained to the previous quotient. Compute remainder.

Step 6 Repeat Steps 4 and 5 until no characters left for first String.

Step 7 The remainder obtained in the final step is the required remainder.

3.5 GCD
Step 1 Take the two numbers as String.
Step 2 Compute the modulus and swap the divisor and remainder as dividend and divisor. Repeat Step 2 until the modulus is zero.
Step 3 Return the divisor.

4. IDENTITIES

The existing trial division method cannot be applied for large integer if it is beyond the compiler limit. To Compute (a*b)%n and (a+b)%n for large numbers a and b as follows. From division algorithm [11], [6] it can be expressed any integer as a= p1*n+q1; b=p2*n+q2; for a given n, a, and b and for some p1, q1, p2, q2. So, (a*b)%n and (a+b)%n can be rewritten as (a*b)%n=((p1*n+q1)*(p2*n+q2))%n = (p1*p2*n^2 + p1*q2*n + p2*q1*n + q1*q2)%n
=(q1*q2)%n=((a%n)*(b%n))%n
(a+b)%n=((p1+p2)*n+q1+q2)%n = (q1+q2)%n =
((a%n)+(b%n))%n. Hence,
(a*b)%n = ((a%n)*(b%n))%n
(a+b)%n = ((a%n)+(b%n))%n

5. MODIFIED TRIAL DIVISION ALGORITHM

Step 1 Take the input number as String.
Step 2 Convert the String into array of integers.
Step 3 Contrary to the exact square root for large number a number greater than the square root near the exact square root is taken instead which is cost effective with respect to time.
Step 4 Compute the length of the String. If it is less than twice the number of digits that a particular compiler supports, then go to Step 5 else go to Step 9.
Step 5 Take a function which will take the value of the array element, the index and the length of the String concerned and the number with which the modulus is to be calculated. Increment the index.
Step 6 Multiply the element with 10 find the modulus and perform the operation iteratively and subtract 1 from length-index until it reaches 0. Compute moduli each time and add, compute the modulus of the sum. Go to Step 3 until all elements are exhausted.
Step 7 The final modulus obtained is the required modulus compared to zero. If the modulus results to zero, it is not prime.
Step 8 The number of numbers with which the input number is to be compared is equal to near_square-root(input number)/2; only the odd numbers below near_square-root(input number) are only compared.
Step 9 Compute the modulus for large number. If it matches the String “0” in any case the number is composite. If the number space for the number concerned is exhausted and none gives the modulus as “0”, hence the number is prime.

6. RSA FOR LARGE NUMBERS BEYOND THE RANGE OF COMPILER LIMIT USING MODIFIED TRIAL DIVISION ALGORITHM

6.1 Key Generation
Step 1 Choose two large number beyond the compiler limit as strings
Step 1.1 If the length is less than 2*n-6 where n denotes the maximum number of decimal digits that a particular compiler supports convert the strings into array of integers else subtract the numbers, below the near_squarertoot of the number equivalent o he string, iteratively from the strings following the method of subtraction (3.1) with which the given string is to be compared.
Step 1.2 The moduli obtained (3.4.1) for each step is compared to 0. If in any case the modulus turns out to be zero the number is not prime, else the number is prime.
Step 1.3 If the length is less than 2*n-6, the elements of the array along with the index and length of th string are fed to a function as arguments which returns the modulus. If each of the moduli tuns out to be non-zero the number is prime else the number is not prime.
Step 1.4 Compute the above methods for both the strings and thus p and q are selected.
Step 2 Convert p an q into array of integers.
Step 3 To compute the value of n=p*q . It is computed with the implementation of the partial product and adding the partial products by adding element by element and handling the carry if any(3.3).
Step 4 Compute the value of f =(p-1)*(q-1). The subtraction of 1 from p and q are obtained by the implementation of the subtraction of large numbers where subtraction is done by element by element and the borrow is handled likewise. Then f is computed with the multiplication with partial products (3.2).
Step 5 Compute the value of e relatively prime to f less than f . The gcd(e, f) is calcuted and convert the result to String, compare it to “1”. If it is equal to 1 e is chosen (3.5).
Step 6 Compute the value of d by using a loop for k in the equation e*d=1+ f*k. If d is equal to 1 go to Step 5.
6.2 Encryption
Step 1 Input a plain text file.
Step 2 Convert the integer value from file into String.
Step 3 Convert the String into array of integers.
Step 4 Compute arithmetic operations as per RSA algorithm on the array of numbers (4).
Step 5 Obtain the result as an array of integers.
Step 6 Convert the array of integers as String to write to the output file as cipher text .
6.3 Decryption
Step 1 Input a cipher text file.
Step 2 Convert the integer value from file into String.
Step 3 Convert the String in to array of integers.
Step 4 Compute arithmetic operations as per RSA algorithm on the array of numbers (4).
Step 5 Obtain the result as an array of integers.
Step 6 Convert the array of integers as String to write to the output file as plain text .
6.4 Example
Choose two numbers
p = 796633327000000971
q = 908819900008701977
Check the first number:
1. The number is not divisible by 2.
2. Check the number if it is divisible by 3.

This checking will continue until the final modulus for each testing number is non-zero upto a number near to the square root = 899999999.

796633327000000971		element	index	length
= 700000000000000000	=	7	0	18
+ 90000000000000000	=	9	1	18
+ 6000000000000000	=	6	2	18
+ 600000000000000	=	6	3	18
+ 30000000000000	=	3	4	18
+ 3000000000000	=	3	5	18
+ 300000000000	=	3	6	18
+ 20000000000	=	2	7	18
+ 7000000000	=	7	8	18
+ 0	=	0	9	18
+ 0	=	0	10	18
+ 0	=	0	11	18
+ 0	=	0	12	18
+ 0	=	0	13	18
+ 0	=	0	14	18
+ 900	=	9	15	18
+ 70	=	7	16	18
+ 1	=	1	17	18

$(7\times10^{17})\ \%3=((7\%3)\times10^{17})\%3=(1\times10^{17})\%3=1\times1=1$

$(9\times10^{16})\%3=((9\%3)\times10^{16})\%3=(0\times10^{16})\%3=0\times1=0$

. =0

. =0

. .

$(9\times10^{2})\%3=((9\%3)\times10^{2})\%3=(0\times10^{2})\%3=(0\times10)\%3=(0\times10)\%3=0$

$(7\times10)\%3=((7\%3)\times(10\%3))\%3=1\times1=1$

$(1\%3)=1$.

Final modulo = $(1+0+0+0+0+0+0+2+1+0+0+0+0+0+0+0+1+1)\%3=6\%3=0$

So the number is not prime, so we choose the nearest prime number 796633327000000969.

Check the second number.

1. The number is not divisible by 2
2. Check the number if it is divisible by 3
3. Next check with 5, then with 7 and so on if any of the final moduli is zero, then the upto 999999999 .

908819900008701977		element	index	length
= 900000000000000000	=	9	0	18
+ 0	=	0	1	18
+ 8000000000000000	=	8	2	18
+ 800000000000000	=	8	3	18
+ 10000000000000	=	1	4	18
+ 9000000000000	=	9	5	18
+ 900000000000	=	9	6	18
+ 0	=	0	7	18
+ 0	=	0	8	18
+ 0	=	0	9	18
+ 0	=	0	10	18
+ 8000000	=	8	11	18
+ 700000	=	7	12	18
+ 0	=	0	13	18
+ 1000	=	1	14	18
+ 900	=	9	15	18
+ 70	=	7	16	18
+ 7	=	7	17	18

This number turns out to be divisible by 61. So, the nearest prime number is 908819900008701973.

Two primes are chosen as:

p = 796633327000000969

q = 908819900008701973

n = 796633327000000969*908819900008701973

$$\begin{array}{r} 796633327000000969 \\ \times\ 908819900008701973 \end{array}$$

Table 1 Computing Multiplication

Partial Products	Computing Partial Products [Table 2.1(a)&(b)]
2389899981000002907	796633327000000969+796633327000000969+ 796633327000000969
5576433289000 0067830	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969
716969994300000872100	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969
796633327000000969000	796633327000000969
0	
557643328900000678300000	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969
6373066616000007752000000	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969
0	
0	
0	
0	
716969994300000872100000000000	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969
7169699943000008721000000000000	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969
7966333270000009690000000000000	796633327000000969
637306661600000775200000000000000	796633327000000969+79663332700 0000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969
6373066616000007752000000000000000	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969
0	
716969994300000872100000000000000000	796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969+796633327000000969+ 796633327000000969

n = 2389899981000002907 + 55764332890000067830 + 716969994300000872100 + 796633327000000969000 + 0 + 557643328900000678300000 + 6373066616000007752000000 + 0 + 0 + 0 + 0 + 716969994300000872100000000000 + 7169699943000008721000000000000 + 7966333270000009690000000000000 + 637306661600000775200000000000000 +

637306661600000775200000000000000 + 0 + 71696999430000087210000000000000000
(Likewise Table 2.1a and Table 2.1b)
n = 723996220587740462348937279432211837

To compute 796633327000000969 + 796633327000000969 + 796633327000000969

Table 2.1a Computing Addition

C	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0
N1	0	7	9	6	6	3	3	3	2	7	0	0	0	0	0	0	6	9
N2	0	7	9	6	6	3	3	3	2	7	0	0	0	0	0	0	6	9
R	1	5	8	6	6	6	6	6	5	4	0	0	0	0	0	1	3	8

Table 2.1b Computing Addition

C	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	0
N1	0	1	5	8	6	6	6	6	6	5	4	0	0	0	0	0	1	3	8
N2	0	0	7	9	6	6	3	3	3	2	7	0	0	0	0	0	0	6	9
R	0	2	3	8	3	2	9	9	9	8	1	0	0	0	0	0	2	0	7

Table 2.1c Computing Subtraction

N1	7	9	6	6	3	3	3	2	7	0	0	0	0	0	0	6	9
N2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
R	7	9	6	6	3	3	3	2	7	0	0	0	0	0	0	6	8

f = (p -1)*(q -1) = (796633327000000969-1)*(908819900008701973-1) [Table 2.1c]
= 796633327000000968*908819900008701972 (Likewise Table 2.1a and Table 2.1b)
= 723996220587740460643484052423508896
Choose e relatively prime to f and less than f.
e is chosen as e = 597730678320781
gcd(e, f) = gcd (597730678320781, 723996220587740460643484052423508896)
=gcd(723996220587740460643484052423508896, 597730678320781)
=gcd(597730678320781,522314937592172)
=gcd(522314937592172, 75415740728609)
=gcd(75415740728609, 69820493220518)
=……………………….
=……………………….
=………………………
=………………………
=gcd(1,1)
=1
From the equation e*d=1+ f*k
597730678320781*350096237795509290502435457320771333
=
1+723996220587740460643484052423508896*289039163112182
d is calculated as d = 350096237795509290502435457320771333

7. RESULTS

The input plain text , cipher text and text after decryption is describe in section 7.1 and the time is needed in different operation is described in section 7.2. To test a number is prime or not is given in table 4. The encryption and decryption time for different file size is furnished table 4.

7.1 Encryption and Decryption
Encryption
A text file is taken as input which contains the plain text:
"This is an implementation of RSA algorithm.
The cipher text is :
321608683768299940577790416009023267545400903235839835977883535534127551
383486098927907800508745124527030055372567718327019004888049249883114843
305006651068030347376681625720317589383486098927907800508745124527030055
372567718327019004888049249883114843305006651068030347376681625720317589
200616657248894833780707759391432479109429525023508137678629803185561222
305006651068030347376681625720317589383486098927907800508745124527030055
23892136854073903222176136841066759292794980504221492971328847789561009
54135690194036557853179004251041518124276562481510613982071022901458181
23892136854073903222176136841066759224276562481510613982071022901458181
109429525023508137678629803185561222247666352896593797503075179955093248
200616657248894833780707759391432479247666352896593797503075179955093248
383486098927907800508745124527030055180575332595018557036123611699469825
109429525023508137678629803185561222305006651068030347376681625720317589
180575332595018557036123611699469825489322524165024894693552060448279197
305006651068030347376681625720317589628929105710247896601309516101856837
500940395461107237765769724850388916349586590894424571185583633154317592
305006651068030347376681625720317589200616657248894833780707759391432479
541356901940365578531790042510415181509158580789464414827778737891311194
180575332595018557036123611699469825125599208770405882129774350203584684
383486098927907800508745124527030055247666352896593797503075179955093248
545400903235839835977883535534127551238921368540739032221761368410667592
212411490368907346642902570781305521
Decryption
The plain text recovered as:
This is an implementation of RSA algorithm.
7. 2 Time for different operation

Table 3. Time to test primes using modified trial division

Digits	Prime	Time to Compute
3	101	<1 sec
3	751	<1 sec
4	1201	< 1sec
4	9091	< 1 sec
5	10753	< 1 sec
5	76801	< 1 sec

6	160001	< 1 sec
6	980801	< 1 sec
7	1146881	< 1 sec
7	9011201	< 1 sec
8	12600001	< 1 sec
8	99328001	< 1 sec
9	104857601	< 1 sec
9	756100001	< 1 sec
10	1027200001	< 1 sec
10	9524994049	1 sec
11	10256250001	1 sec
11	97656250001	2 secs
12	100907200001	2 secs
12	947147262401	3 secs
13	1079916250001	5 secs
13	9982699110401	8 secs
14	12123750000001	10 secs
14	87770788000001	25 secs
15	101702694862849	53 secs
15	944377409044481	113 secs
16	1136591040000001	127 secs
16	9502720000000001	305 secs
17	12136000000000001	702 secs
17	95348273971200001	1410 secs
18	100663296000000001	1630 secs
18	908800000000000001	3990 secs

Table 4. Time to encrypt and decrypt Text files of different sizes

File Size	Encryption Time	Decryption Time
1 KB	4 secs	210 secs
2 KB	8 secs	420 secs
3 KB	12 secs	641 secs
4 KB	16 secs	855 secs
5 KB	20 secs	1070 secs
6 KB	24 secs	1290 secs
7 KB	28 secs	1500 secs
8 KB	32 secs	1721 secs
9 KB	36 secs	1943 secs
10 KB	39 secs	2174 secs

8. CONCLUSION AND FUTURE WORKS

In this paper, modified trial division algorithm has been used to find large prime numbers. Even of the integer number be beyond the compiler limit. The time complexity of this algorithm will be always less than the existing trial division algorithm as to check for primality only odd numbers have been used. This method can be used in personal computer for implementation of RSA algorithm with large integer.

REFERENCES

[1] Boneh and Durfee, "*Cryptanalysis of RSA with private key d less than* $n^{0.292}$", IEEETIT: IEEE Transactions on Information Theory, Volume 46, Issue 4, Jul 2000 pp:1339–1349.

[2] Whitfield Diffie and Martin E. Hellman, "*New directions in cryptography*", IEEE Transactions on Information Theory IT-22, no. 6, 1976, pp644-654.

[3] Tatsuaki Okamoto and Shigenori Uchiyama, "*A new public key cryptosystem as secure as factoring*", Lecture notes in Computer Science 1403 (1998), 308-318. MR 1 729 059

[4] Ron L. Rivest, Adi Shamir, and Len Adleman, "*A method for obtaining digital Signatures and public-key cryptosystems*", Communications of the ACM 21 (1978), pp 120-126.

[5] M. Wiener, "*Cryptanalysis of short rsa secret exponents*", IEEE Transactions on Information Theory 36 (1990), pp.553-558.

[6] William Dunham, "*Euler – the master of us all*", The Mathematical Association of America, 1999. ISBN: 0883853280

[7] Steven Levy, "*Crypto-secrecy and privacy in the new code war*", Penguin Books, 2000

[8] Alfred J Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "*Handbook of applied cryptography*", CRC Press, 1996.

[9] Hans Riesel, "*Prime numbers and computer methods for factorization*" *(2nd ed.)*, Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.

[10] Douglas R. Stinson, "*Cryptography, theory and practice*", CRC Press, 1995.

[11] David M. Burton, "*Elementary Number Theory*" *(2nd ed)*, Universal Books Stall, New Delhi, 2004.

[12] M. Agrawal, N. Kayal, and N. Saxena, "*Primes is in p*", Preprint, Aug. 6, 2002. Date of access 25.04.2009. http://www.cse.iitk.ac.in/primality.pdf.

Authors Biography

Satyendra Nath Mandal has received his B.Tech & M.Tech in Computer Science & Engineering from university of Calcutta, West Bengal India. He is now working as a lecturer in department of Information Technology at Kalyani Govt. Engg. College , Kalyani , Nadia, West Bengal, India. Hs field of research areas includes cryptography & network Security, fuzzy logic, Artificial Neural Network, Genetic Algorithm etc. He has about 25 research papers in national and International conferences. His three research papers have been published in International journal.

Kumarjit Banerjee has received his B. Tech degree in Computer Science and Engineering form West Bengal University of Technology, West Bengal, India. His field of interest includes Image Processing, Number Theory and Artificial Intelligence.

Dr. *Biswajit Maiti* is a reader of Dept. of Physics in Kalyani Govt. Engg. College. His has 11 international journals and 3 conference papers. His area of Specializations are Synthesis and characteriszation of semiconducting Meterial.

Dr. *J. Paul Choudhury*(Jagannibas Paul Choudhury) completed Bachelor of Electronics and Tele-Communication Engineering(Hons) from Jadavpur University, Kolkata M. Tech in Electronics and Electrical Engineering under the specialization of Control and Automation Engineering from Indian Institute of Technology, Kharagpur and. thereafter completed PhD(Engg) from Jadavpur University, Kolkata. At present Dr. Paul Choudhury is with Information Technology Department, Kalyani Government Engineering College, Dt Nadia, West Bengal, India, and he has 60 publications in National and International Journals and in Conference Proceedings. His field of interest is Soft Computing, Data Base, Object Oriented Methodology, etc.