Implementation of RSA algorithm Using Mersenne Prime

Shilpa M Pund, Chitra G Desai
Ph.D Student, Department of Computer Science,
University of Pune. Maharashtra, India.
Professor and Head, MCA Department,
MIT Aurangabad, India.

ABSTRACT

Cryptographic algorithms are oftenly based on large prime numbers. It is a difficult task to generate large prime numbers and test it for primality. This requires overheads on first generating a large number which should be probably prime and second, testing its primality. This is because the generated large prime number using different algorithms may not necessarily pass the primality test. In order to generate an assured large prime number we can make use of Mersenne primes. Small primes are the used to generate large mersenne primes. This paper implements RSA based cryptographic algorithm using mersenne prime.

General Terms

Network security, RSA algorithm, Mersenne primes , Integer factorization.

Indexing terms

File encryption and decryption, Euclidean algorithm, Bézout's Identity.

I. INTRODUCTION

Encryption is one of the principal means to grantee the security of sensitive information. It not only provides the mechanisms in information confidentiality, but is also functioned with digital signature, authentication, secret sub-keeping, system security and etc. Therefore, the purpose of adopting encryption techniques is to ensure the information's confidentiality, integrity and certainty, prevent information from tampering, forgery and counterfeiting [1].The well-known and most widely used public key system is RSA, which was first proposed in a paper " A method for obtaining digital signatures and public-key cryptosystems" R.L. Rivest, A. Shamir and L. Adleman in 1976. It is the most influential public-key encryption algorithm, and has been recommended for ISO public key data encryption standard. RSA algorithm is an asymmetric cryptographic algorithm, meaning that the algorithm requires a key pair, one for encryption, another for decryption. Its security is based on the difficulty of the large number prime factorization, which is a well-known mathematical problem that has no effective solution [1] .This paper designed a complete and practical RSA encoding and decoding solution on *.txt file and provided the analysis on keys generated and the file size. The implementation is done in MuPAD 5.4 i.e. a Symbolic Math Toolbox provided by Matlab.

RSA algorithm can be simply described as

- <Generate keys >
- Take the prime numbers p,q, compute n=p×q
- Select an integer e which is coprime with (p-1) × (q-1)
- Compute a private key d such that the equation : $d \times e \equiv 1 \pmod{(p-1)\times(q-1)}$.
- Pair (e, n) as a public key
- Pair (d, n) as the private key
- < Encryption and decryption >
- $b=a^e \bmod n$, where a is plain text
- $a=b^d \bmod n$, where b is cipher text.[2]

In RSA algorithm, it is recommended that each user must (privately) choose two large random prime numbers p and q to create encryption and decryption keys. These numbers must be large so that it is not computationally feasible for anyone to factor n = p * q. Also it recommends use of 100-digit (decimal) prime numbers p and q, so that n has 200 digits.[3]

II Mathematical Foundation For Public-key Cryptography

Integer factorization

Integer factorization is one of the most challenging problem of number theory and hardness of this problem is behind the security of RSA. In a public key cryptosystem, if one knows how to encrypt a message does not mean that it can be easily decrypted. So if anybody knows how a message is encrypted - that does not create any risk to the security of the system . The public key of RSA consists of n= p*q, where p and q are large, distinct positive primes. Another positive integer is e which is inverse modulo of Φ(n), where Φ(n) is Euler totient function. For encryption we need n and a public key e. For Decryption we need n and a private key d. Here in RSA we are interested in factoring n which has exactly two prime divisors p and q of approximately equal size (where p and q are very large numbers.) Due to its relationship with RSA ,factoring has become one of the most important research problems. Hence the size of modulus in RSA algorithm determines how secure the RSA cryptosystem is.[4]

Number theory

Various number theory algorithms form the base of the cryptographic algorithms. The important and widely used modular arithmetic concepts and algorithms are:

Prime number : A prime number is defined as an integer greater than 1 which has no positive divisor other than 1 and the number itself.

Fundamental Theorem of Arithmetic : Fundamental Theorem of Arithmetic states that any natural number n can be written uniquely as a product of prime numbers.

e.g. $4200 = 2^3 * 3 * 5^2 * 7$.

Modular arithmetic

Congruences

Three integers a,b and m , we say that " a is *congruent to b modulo m* " . This relationship is written as $a \equiv b \pmod{m}$.

Two integers are congruent modulo *m* if and only if their

difference is divisible by *m*. *m* is called the modulus of congruence.

Example: $11 \equiv 19 \pmod{8}$.

Modular Reduction

Modular arithmetic is based on a fixed integer $m > 1$ called the *modulus*. The fundamental operation is *reduction modulo m*. To reduce an integer *a* modulo *m*, one divides *a* by *m* and takes the remainder *r*.

This operation is written as $r := a \bmod m$.

The remainder must satisfy $0 \leq r < m$.

Examples: 3:= 15 mod 6

8:= –3 mod 11

Congruence modulo m.

For fixed m, each equivalence class with respect to congruence modulo m has one and only one representative between 0 and m-1. The set of equivalent classes (called residue classes) will be denoted Z/ mZ. Any set of representatives for the residue classes is called a complete set of residue modulo m represented as $Z_m = \{0, 1, \ldots, m-1\}$.

One performs addition, subtraction, and multiplication on the set Z_m by performing the corresponding integer operation and reducing the result modulo *m*. [5]

Multiplicative Inverse

The *multiplicative inverse* of *a* modulo *m* is the integer *b* modulo *m* such that $ab \equiv 1 \pmod{m}$. The multiplicative inverse of *a* is commonly written as $a^{-1} \pmod{m}$. It exists and is unique if *a* is relatively prime to *m* and not otherwise.

If *a* and *b* are integers modulo *m*, and *a* is relatively prime to *m*, then the *modular quotient a*/*b* modulo *m* is the integer $ab^{-1} \bmod m$.

If *c* is the modular quotient, then *c* satisfies $a \equiv bc \pmod{m}$.The process of finding the modular quotient is called *modular division.*

Euclidean algorithm

The *Euclidean algorithm* is an efficient method to compute the *greatest common divisor* (gcd) of two integers.

We write gcd(a, b) = d to mean that d is the largest number that will divide both a and b. If gcd(a, b) = 1 then we can say that a and b are coprime or relatively prime. The gcd is sometimes called the highest common factor (hcf).[6]

Program : Developed in MuPAD 5.4 Environment for computing the greatest common divisor of two integers using Euclidean algorithm.

Input: Two non-negative integers a and b with $a \geq b$.

```
euclid_algo:= proc(a, b)
 local r;
begin
 while b>0 do
    r:= a mod b;
    a:= b;
    b:= r;
 end_while:
print(a)
end:
```

Program Execution :

```
euclid_algo(15618427, 24137569)
1419857
euclid_algo(123467823465798212 3,13485785785738457384573987598475894758 47584393)
1
```

The proof uses the division algorithm which states that for any two integers a and b with b > 0 there is a unique pair of integers q and r such that $a = qb + r$ and $0 \leq r < b$. Essentially, a get reduces with each step, and so, being a positive integer, it must eventually converge to a solution (i.e. it cannot get smaller than 1).

Bézout's Identity

In number theory, Bézout's identity for two integers a, b is an expression ax + by = d, where x and y are integers called Bézout coefficients for (a,b) such that d is a common divisor of a and b. If d is the greatest common divisor of a and b then Bézout's identity ax + by = gcd(a,b) can be solved using Extended Euclidean Algorithm.

Finding the Modular Inverse

The modular inverse of an integer e modulo n is defined as the number d such that $ed \equiv 1 \bmod n$. We write $d = e^{-1} \bmod n$ or $d = e^{-1} \bmod n$. The inverse exists if and only if gcd(n,e)=1.To find this value for large numbers we use the extended Euclidean algorithm, but there are simpler methods for smaller numbers.

Extended Euclidian Algorithm

The Extended Euclidean Algorithm is an extension to the Euclidean algorithm. Besides finding the greatest common divisor of integers a and b, as the Euclidean algorithm does, it also finds integers x and y (one of which is typically negative) that satisfy Bézout's identity ax + by = gcd(a,b).

The Extended Euclidean Algorithm is particularly useful when a and b are coprime, since x is the multiplicative inverse of a modulo b, and y is the multiplicative inverse of b modulo a.

Program : Developed in MuPAD 5.4 Environment for Extended Euclidean algorithm.

Input: Two non-negative integers a and b.

Output d = gcd(a, b) and integers x and y satisfying ax + by = d.

```
ex2:= proc(a, b)
  local x,y,x1,y1,x2,y2,q,r,d,z;
begin
  s:=a;
  z:=0;
  print(s);
if b = 0 then
    d:= a;
    x:= 1;
    y:=0;
    print(d,x,y);
end_if;
if b >a then
    temp:= a;
    a:= b;
    b:=temp;
    print(a,b);
end_if;
 x2:= 1;
 x1:= 0;
 y2:=0;
 y1:=1;
while b>0 do
  q:= a div b;
  r:= a - q*b;
  x:= x2 -q*x1;
  y:= y2 -q* y1;
  a:=b;
  b:=r;
  x2:=x1;
  x1:=x;
  y2:=y1;
  y1:=y;
end_while;
d:= a;
x:= x2;
y:= y2;
print("y=",y);
if y < 0 then
  z:= s + y;
```

```
else
  z :=y;
end_if;
print("gcd=",d);
print("Coefficient of a",x);
print("Coefficient of b",y);
print("The modular inverse is ",z);
end_proc:
```

Program Execution :

ex2(421,111)

"y=", 110

"gcd=", 1

"Coefficient of a", -29

"Coefficient of b", 110

"The modular inverse is ", 110

ex2(552804821524705862425441525968984758892867659653333928103098624447507121872017357237088653540245170100667448996030608537729589597718200642115861634269352828134942465627287734406691594852739527031685058691123363086516100574455617076610072735123003373746977128787080931756865334031267832727830621541930401689858580184076555523047632328249323799801555863725934065872761919687994269363332733098113631273063500936538279827139921177184254580496225475925377983148758433762633822209538018615759203855924906940090591601683914236230986508063906069830582301163217262386970047283 6,429392673709)

"y=",

257959486022090474386811667380977933172360311716614312431100886295222073248022803685199813158947995430511453902306035500248604204258556382107618069865938111301820856059913121443594108724153101814921669271017392882246629502206679732736251070824312449506444111214575527711350923550812653995726677405782836686505980192575722265617588355367664097621146419855273916932128980131903036655373391921802415276762010430074884112883110958681249137642848174417725509810267238766981671958967147329590753298214398208694692579840338736633377029134068778407331743840667610764509226507 1

"gcd=", 1

"Coefficient of a", 200370744065

"Coefficient of b"

257959486022090474386811667380977933172360311716614312431100886295222073248022803685199813158947995430511453902306035500248604204258556382107618069865938111301820856059913121443594108724153101814921669271017392882246629502206679732736251070824312449506444111214575527711350923550812653995726677405782836686505980192575722265617588355367664097621146419855273916932128980131903036655373391921802415276762010430074884112883110958681249137642848174417725509810267238766981671958967147329590753298214398208694692579840338736633377029134068778407331743840667610764509226507 1

"The modular inverse is ",

294845335502615388038629858588006825720507347936719615671997738152285048623994553551888840381297174670155995093724573037480985393459644260008243564403414716833121609567374612963097486128586425216763389420105970204269471072248937343873821664298690924240533017572505404045514410480455178732103944136147565003352599991500833257430043972881659702180409443870660148940632939556091232707959341176311215996301490506463395714256810218503005442853377301507652473338491194995652150250570876168450557710508731395899021843575499597609478929837291423250557322549651622460820776 5.

III. Mersenne Prime Numbers

In mathematics, a **Mersenne number**, named after Marin Mersenne (a French monk who began the study of these numbers in early 17th century), is a positive integer that is one less than a power of two. **Mersenne Primes** *are of the form* $2^p - 1$*, where* ***p*** *is itself prime represented as* $M_p = 2^p - 1$. Since 1997, all newly-found Mersenne primes have been discovered by the **"Great Internet Mersenne Prime Search" (GIMPS)**, a distributed computing project on the Internet. The search was on when it was noticed that most of the early primes worked, but $2^{11}-1$ was not prime. The great Mersenne Prime race has been in progress now for over 600 years and shows no sign of ending. Some of the more reasonably sized prime numbers which generates mersenne primes are given in this list : 2,3,5,7,13,17,19,31,61,89,107,127,521,607,1279,2203,2281,3217,4253,4423,9689,9941,11213,19937,21701,23209,44497,86243,

110503..... so on.

As of: Date: 2012/06/12 05:06:11 the last Mersenne prime shown above was the largest known prime. $2^{43112609}-1$[7].

So we can generate very large primes by using M p = 2^p -1 where Mp itself is a Mersenne Prime.

IV. RSA Algorithm Using Mersenne Primes

Declarations:

a) Ptext.txt : Plain text (source) file to be encrypted .

b) Ctext.txt : Cipher text file to be decrypted.

c) Array L : Used to store the contents of Ptext.txt for encryption.

d) Array S : Used to store the contents of Ctext.txt for decryption.

Algorithm :

1. Choose two large Prime numbers p , q.

2. Generate two very large mersenne prime numbers as: m = 2^p -1 and n = 2^q -1.

3. Calculate c= m* n .

4. Calculate the value of Φ using the formula: Φ(c) = (m-1) * (n-1).

5. Generate the public key 'e' such that it is coprime with Φ(c).

6. Find the value of private key 'd' such that (d *e) ≡ 1 mod Φ(c)

7. Read plaintext in the form of binary data from the file **Ptext.txt** ,store it in an array(**L)** .

8. Perform L^e mod c (on each element of an array L) to get cipher text and store it in the file **Ctext.txt**.

9. For decryption, read the file **Ctext.txt** , store it in an array (S) .

10. Perform S^d mod c (on each element of an array S) to get a plain text .

V. Output Details

1. Input : p=127,q=2203

"The value of m is", 170141183460469231731687303715884105727 (39-digits)

"The value of is n"

14759799152141802350848986227373817363120661453331697751477712164785702978780789493774073370493892893827485075314964804772812648387602591918144633653302695404969612011134301569023960939890902262593269350252814096149834993882228314485986018343185362309237726413902094902318364468996082107954829637630942366309454108327937699053999824571863229447296364188906233721717237421056364403682184596496329485386969058726504869144346374575072804418236768135178520993486608471725794084223166780976702240119902801704748944874269247421088235368084850725022405194525875428753499765585726702296339625752126374778977855015526465226099888699140135404838098656812504194976866977710007. (658 digits)

"The value of c is"

25112496953842366118263729475441408070286242277657790903062203704936919271595794205599126851183594143498152389443247272512724813223997023323657591173057701682734203804753649533992054202779926084894442519637085753274356375927821354198106857762709531252922504043551767434172766293247445664232781107416474396571741599975199307583666818282133873258428509963749082638547017533743580910844085672421744347995263682452863081307452993528542338191057567544692158111898135787578979647338149471629231811440438483861330314975110281023455997795613791015656

15697815691060270436422430816778355617405820453599558614408223598804280306383394526250820823731245694043692157093842690637750780782894620990626960314207042325708 9. (702 digits)

"The value of is Φ (c) ",

25112496953842366118263729475441408070138644286136372879553713842663181097964587591065809873668817021850295359655439377574984079519058094385382740419908053635006077320877623614810607866246899130844746399525742737584116766528912331572174164260181390291424154104729484289312906109815592041140403843277453447548557955285238486504120321905824449595333968880469705648007019288024948616371122030532682010778091308242299437270631147563579043337187876957427109420454672041828251603155781790277446601505572399144072374132878613213688975394414762998568208366948367796225540077135929848367181768508340804326873225942947013283419998268729557075941467143888427504832843803646332412616167587499328749634066784138035 6.

"The value of public key is", 745580037409

"The value of private key is",

11544772734328475852673935272132987011622415538172654067570375942923342165377029161693575788061852297758249831144830982677972518399232756186350192371436756119044093571917499897032690000083452802031770488641065155076630257418930079141309628371875408014936091925216980409600977148463475922049869124077263537343044263679638220305047081789892498241825102183787199386235971722366876177692832900689206637687283785716085246020446746745758167200285084738625292282363753257996443919424927655749118781740594773694602553696317478097568391027655787265052681806784839469584721942774754756087338039441102080842003260118885083496683414253556514510933760975111818296888620030121796763893455488078906179647490522326 77. (689 digits)

VI. Implementation Details

Initially, we the algorithms were implemented using MatLab. But as it does not support large integer for computations, MuPAD has used. MuPAD can compute big numbers efficiently. The length of a number that can be computed is limited only by the available main storage [8] . So, MuPAD has provided preferable environment to develop this program. The algorithmic steps are as follows.

1. Choose two large prime numbers p and q.

2. Generate two very large mersenne prime numbers using the formula $m= 2^p -1$ and $n= 2^q -1$. As, a prime can generate a large mersenne using the above formula.

3. Calculate the value of c = m * n.

4. Calculate the value of Φ(c) , Euler Totient function using the formula: Φ(c) = (m-1) (n-1).

5. Generate the public key 'e' randomly using **random** function such that it is coprime with Φ(c) . Using **gcd** function based on Euclid's Algorithm. i.e. gcd(e, Φ(c)) = 1.

6. Find the value of private key 'd' by implementing Extended Euclidian Algorithm.

7. For Encryption, use **readbytes** function which reads plaintext and converts it into binary data ,store it in an array.(**L)** . Permissible values for binary data are Bytes, SignedBytes, Short, Signedshort,Float and Double.

8. Perform L^e mod c (on each element of an array L) by using **powermod**(L,e,c) function provided by MuPAD. Map each element of array L with the powermod value using **map** function.

9. Open a new file say ctext.txt in write mode using **fopen** function.

10. Now L contains cipher text, write array L in the file **ctext.txt** using **write** function.

11. For decryption, read the file ctext.txt by using **read** function , store it in an array (S) .

12. Perform S^d mod c (on each element of an array S) by using **powermod**(L,e,c) function. Map each element of array S with the powermod value using **map** function.

13. Open a new file say ptext.txt in write mode using **fopen** function.

14. Use **writebytes** function to convert binary data into plain text and write it in the file ptext.txt. The above algorithm works for the mersenne primes greater than 17.

VII. Comparison

The size of the encrypted file depends on the size of keys generated, which further depends on the values of p and q. As we increase the values of p and q, the size of the encrypted file also increases. As the size of private key increases the execution time and memory requirement are also increased.

Table 1. Comparison of the original , encrypted and decrypted file sizes with respect to the value of p and q.

Value of p	Value of q	Private Key Size(In digits)	File name	Original file size	Encrypted File Size	Decrypted File size	Execution Time	Memory Used
521	607	340	conversion.txt	3.14KB	1.08 MB	3.14KB	13 sec	7MB
127	521	195	conversion.txt	3.14KB	644KB	3.14KB	3 sec	8MB
127	2203	689	modrsa.txt	2.27KB	1.62 MB	2.27KB	69 sec	8 MB
607	1279	568	rsa.txt	1.01KB	599KB	2KB	17 sec	7 MB
3217	4253	2249	rsa.txt	1.01KB	230MB	1.01KB	823 sec	7MB
4253	4423	2562	rsa.txt	1.01KB	2.68 MB	1.01KB	1220 sec	8MB
9689	9941	5910	T1.txt	27 Bytes	161 KB	27 Bytes	262 sec	7MB
9689	9941	5909	x25.txt	172 Bytes	1 MB	172 Bytes	1656 sec	7MB
11213	19937	9378	T1.txt	27 Bytes	256 KB	27 Bytes	879 sec	6MB

VIII. Conclusion

In this paper we have implemented RSA algorithm using mersenne primes. The real challenge in case of RSA is the selection of public key and generation of private key. Here public key is generated randomly. The use of merssene prime has been made to generate large primes to enhance the security. This algorithm can generate a private key in the range of 15 to 5000 digits. We have executed this program for the primes greater than 9000 on Intel(R) Core i5 -2410 M CPU with 6144 MB RAMS, it generates the private and public key. The system is able to do encryption and decryption for the file size up to 180 bytes. If the file size is too large and the primes are beyond 9000 then MuPAD does not respond because of the main storage limitation. The above algorithm will not give proper result for the primes less than 17.

The future work intends to reduce the size of encrypted file which is the limitation of the present system.

Acknowledgment

First author would like to thank Dr. Nivedita H Mahajan for her valuable guidance in Mathematical concepts.

References

[1] Xin Zhou and Xiaofei Tang," Research and Implementation of RSA Algorithm for Encryption and Decryption" 2011 The 6th International Forum on Strategic Technology. IEEE.

[2] Suli Wang, Ganlai Liu, "File encryption and decryption system based on RSA algorithm" 2011 International Conference on Computational and Information Sciences. IEEE.

[3] L. Adleman, R. Rivestand, A. Shamir "A Method for Obtaining Digital Signatures and Public Key Cryptosystems" Comm. ACM 21, 1978, pp120-126.

[4] Prashant Sharma, Amit Kumar Gupta, Ashish Vijay, "Modified Integer Factorization Algorithm using V-Factor Method ", 2012 Second International Conference on Advanced Computing & Communication Technologies. IEEE

[5] "A course in Number Theory and Cryptography" Second Edition By Neal Koblitz.

[6] http://www.dimgt.com.au/euclidean.html#KNU298

[7] http://en.wikipedia.org/wiki/Mersenne_prime.

[8] Symbolic Math Toolbox™ 5 MuPAD® Tutorial by The Mathworks.

[9] http://en.wikipedia.org/wiki/Integer_factorization.