Journal of the Nigerian Association of Mathematical Physics
Volume 10 (November 2006), **423 - 432**

A software for the RSA Encription

E. E. Obasohan
Department of Computer Science, University of Benin, Benin City
and
H. O. Omokaro,
Department of Mathematics, University of Benin, Benin City, Nigeria.

Abstract

In Omokaro 2003[12], we extended the RSA Congruence to a finite number of primes. The extended RSA Cryptosystem was later obtained in Omokaro 2004[13] as an analogue of the RSA Cryptosystem to obtain the extended RSA Cryptosystem. In this work we provide a software for the enciphering of data in RSA cryptosystem

1.0 Introduction

As explained in [12] so many times we are faced with a problem of sending information in such a way that if seen by unauthorized persons they will not be able to understand it. The way of sending information under some degree of protection is called **Cryptology.**

The RSA Congruence, as mentioned in [12], [6] is a cryptosystem, which was developed by Rivest, Shamir and Adleman. It states that: if p and q are primes and e and d are positive integers such that

$$ed \equiv 1 \bmod (p-1)(q-1)$$

then for any positive integer $m < pq$,

$$m^{ed} \equiv m \bmod pq. \ [8], [9], [12].$$

Its security is based on the difficulty of factorizing large primes. In the RSA Cryptosystem there are two keys namely the enciphering key S_k and the deciphering key p_k. As explained in [13] the keys S_k and P_k are obtained by solving a congruence modulo Euler-phi function of a product primes. Let us take a brief look at it before we move on to develop the software which is the target of this paper in the next section.

Let $S_k = e$ and $P_k = d$ respectively and let $n = pq$, a product of primes p and q be the modulus of the congruence. The encoded message is obtained by applying:

$$E(\mathrm{M}) = C = M^e \text{ Mod } n \ldots [\ \ ]$$

where M is the numerical equivalent of the message.

The original message is obtained from the Cipher text by applying, $D(C) = C^d \bmod n$, clearly, this is possible as shown in [7], [8], [9], [13]. We know that once one of the factors p or q of n is known, $\varnothing(n)$ can be obtained and so the private key can be determined thereby allowing the breaking of the security of the cryptosystem. Where ø denotes the Euler-phi function. Also as stated in [12], [13] the keys must satisfy: $ed \equiv 1 \bmod (p - 1)(q - 1)$. So that if p is known q can be calculated and hence e can be found. Let us take a look at a practical example of the RSA key generation and an RSA-based cryptographic exchange.

1. Generating primes to obtain modulus let $p = 17$, $q = 13$
   $\therefore n = pq = 17\times13 = 221$
2. Public key Calculation
   $\Phi(n) = (p - 1)(q - 1) = (17 - 1)(13 - 1) = 16 \times 12 = 192$
   Let e = 23, clearly (e, ø(n)) = (23, 192) = 1
3. Private key Calculation:

Now ed ≡ 1 mod $ø(n)$ ⇒ 23d ≡ 1 mod 192 i.e. 23d -1 = 192k for some positive integer k. this yield d = 167.

4. To obtain our cipher text given a message data block of numeric equivalent M we use

$C = M^e \bmod n$

i.e. $C = M^{23} \bmod n$ in this illustration.

5. Cipher text deciphered with private key to obtain the original data block

Recall $M = C^d \bmod n$, so we now use

$M = C^{167} \bmod n$.

2.0 RSA Analysis

2.1 Enciphering

Let us illustrate the RSA enciphering with the following example: First we make the following substitution:

Symbol	Space	A	B	C	D	.	.	.	.	X	Y	Z
Number	00	01	02	03	04	.	.	.	.	24	25	26

and then consider sending the message: "SEND MONEY" as an example. First we obtain the numerical equivalent of the letters and hence the words as follows:

$$19\ 05\ 14\ 04\ 00\ 13\ 15\ 14\ 05\ 25 \qquad (\alpha)$$

let us choose our primes as follows:

$$p = 29,\ q = 41 \text{ so that } M = pq = 29 \times 41 = 1189,\ e = 3$$

since M has 4 digits we break the message in (α) into groups of 3 digits.

$$190\ 514\ 040\ 013\ 151\ 405\ 250 \qquad (\beta)$$

let us label the integers in (β) as follows: $p_1 = 190$, $p_2 = 514$, …

It is interesting to note that anyone can decipher the sequence at this stage because this method of transforming the message into a sequence of numbers is agreed upon before hand. The enciphered message is now the sequence $c_1, c_2, c_3, \ldots$, where c_i is defined as:

$C_i = P^e \bmod m$. e is chosen such that (e,z) = 1 say $e = 3$.

Then $C_1 = 190^3 \bmod 1189 = 848$

$C_2 = 514^3 \bmod 1189 = 1054$

Continuing in this manner we obtain

$$C_3 = 695,\ C_4 = 1008,\ C_5 = 796,\ C_6 = 695,\ C_7 = 351 \qquad (\gamma)$$

2.1 Deciphering

Let us now obtain the plaintext from (γ). First we obtain the private key as follows :($d.e$) mod m = 1, i.e. (d.3) mod 1189 = 393, then for e_i i = 2,3,…,7 we compute the p_i'' using $p_i \equiv C_i^d \bmod n$

***Remark* 2.1**

The computation above is very cumbersome if we are to encipher and decipher a very large amount of message for example a textbook of several pages. If we attempt to do this manually the effort may end up in fiasco. So we need to make use of the computer; the following program has been designed using C++ [4] to take care of this problem.

3.0 Program design for RSA

Design has to do with transforming the algorithm into data structures; how the system modules will interact with one another; and the overall architecture of the system.

The program design for the gcd algorithms and its area of application (RSA Cryptography) is made up of the following modules in the design model:

(a) Main() module ⇨ main program

(b) Gcd() module ⇨ function that calculate gcd (a,b)

(c) Euclid() module ⇨ function that return gcd (a,b) and integers x and y such that $ax + by = \gcd(a,b)$

(d) rsa() module ⇨function that return the public andprivate keys for data encryption and decryption using RSA cryptographic algorithm.

(e) Matmult() module ⇨ function that multiplies 2×2 matrix together.

(f) Isprime() module ⇨ function that determines if a number is prime (return 1 if a number is prime otherwise return o).

The main () function accesses the gcd() function, euclid() function, and the rsa() function through its main menu options. The euclid() function invokes the matmult() function when called. The rsa() function also invokes the isprime() funtion and when called. The overall program design and its sub modules are here presented.

```
# include <iostream.h>
# include <iomanip.h>
# include <math.h>
# include <stdlib.h>
# include <ctype.h>
int gcd( int , int );
void euclid(int , int , int& , int& , int& );
void matmult( int[2][2] , int[2][2] , int[2][2] );
int isprime( int );
void rsa (void );
main()
 {
        char flag;
cout<<"\n\n\n\n\n\t\t\t**************************************"
             <<"\n\t\t\t THE GREATEST COMMON DIVISOR PROBLEM"
             <<"\n\t\t\t*************************************"
  <<"\n\n\n\t-------------------------------------------------------------------"
   <<"\n\ \t This application compute the greatest common divisor of two integers"
     <<"\n\tand also apply the greatest common   divisor to generate a public key"
            <<" \n\tand its private key using RSA cryptographic algorithm ."
<<"\n\t-------------------------------------------------------------------";
loop:cout<<"\n\n\t\tDo you wish to continue(yes/No)?:press(Y/N OR y/n)\t";
again: cin >> flag;
                        if ( flag == 'Y' || flag == 'y' )
                        {
  menu:   cout <<"\n\n\t\t\tMAINMENU"<<"\n\t\t\t=========="
     <<"\n\t\tSELECT CODE 1 TO 4 FOR OPERATION"<<"\n\t\t-----------------------------------"
<<"\n\t\t1 - \tCalculate gcd \n\t\t2 - \tExtended Euclidean Algorithm"
                          <<"\n \t\t3 - \tApplication of gcd " << "\n\t\t4 - \tExit\n\t\t\t " ;
                        }
        else
         if ( flag == 'N' || flag ==  'n' )
           {
            cout<<"\n\tclick the close button [x] at the top left corner of this window";
            return 0 ;
           }
         else
          {
            cout << "\n\t\tpress ( yes/no ): (Y/N OR y/n )\t" ;
            goto again ;
          }
        char opcode[10] ;
        cin >> opcode ;
        int op  = atoi( opcode );
        switch (op)
         {
```

```
          case 1:
             char a[10] , b[10] ;
             int temp1 ,  temp2 ;
again_1:cout<<"\n\t\t-----------------------------------"
                          <<"\n\t\tCalculating the gcd of two integers”
                          <<"\n\t\t----------------------------------- "
                        <<" \n\n\t\tEnter the first number\t ";
             cin >> a ;
              temp1 = atoi(a) ;
             cout << " \n\t\tEnter the second number\t ";
             cin >> b;
              temp2 = atoi(b);
     cout <<"\n\t\t-------------------------------------------------------"
            << "\n\t\tgcd("<<a<<","<<b<<") = "<<gcd(temp1,temp2)
            <<"\n\t\t-------------------------------------------------------";
              char tag ;
  cout<<"\n\n\t\tAny other gcd calculation(YES/NO):PRESS(Y/N OR y/n)";
              cin>>tag;
              if ( tag == 'Y' || tag == 'y' )
                        goto again_1 ;
              else
                        goto loop ;
          case 2 :
             char numb1[10], numb2[10] ;
             int g_c_d ,  x , y  ;
again_2:cout<<"\n\n\t\t-----------------------------------"
              <<"\n\t\tExtended Euclidean algorithm "
              <<"\n\t\t-----------------------------------";
             cout << "\n\t\tenter the first number\t" ;
             cin >> numb1 ;
             temp1 = atoi(numb1);
             cout << " \n\t\tenter the second number\t" ;
             cin >> numb2 ;
             temp2 = atoi(numb2);
             euclid(temp1,temp2,g_c_d , x , y ) ;
cout <<"\n\t\t-------------------------------------------------------"
<<" \n\t\tgcd("<<numb1<<","<<numb2<<") = "<<g_c_d    <<",\tx = "<<x <<",\ty  = " << y
            <<"\n\t\t-------------------------------------------------------";
cout<<"\n\n\t\tAny other calculations on Extended Euclidean algorithm"
            <<"\n\t\t(YES/NO):Press(Y/N OR y/n)\t";
              char tag1 ;
              cin>>tag1;
              if ( tag1 == 'Y' || tag1 == 'y' )
                        goto again_2 ;
              else
                        goto loop ;
          case 3 :
                        char tag4 ;
 opcode3:               cout <<"\n\n\t***********************************************************************"
<<"\n\tGenerating public and private key via RSA cryptographic algorithm "
<<"\n\t******************************************************************";
rsa() ;
cout <<"\n\n\tAny more key generation(YES/NO)?:press(Y/N OR y/n )\t" ;
            cin >> tag4 ;
```

```
                    if ( tag4 == 'Y' || tag4 == 'y' )
                              goto   opcode3 ;
                    else
                              goto loop ;
        case 4:
            cout<<"\n\n\tAre you sure you want to exit(YES/NO)?:"
                       <<" Press(Y/N OR y/n)";
            char tag_3   ;
            cin>> tag_3 ;
            if ( tag_3 == 'Y' || tag_3 == 'y')
             {
             cout<<"\n\tClick the the left button [x] on the top left corner of this window";
             break ;
            }
           else
            goto loop ;
         default:
            cout<<"\n\n\t\tSelect the right code for operation ";
            goto menu ;
        }
        return 0 ;
 }
 //this function  evaluate gcd of two integers
int gcd( int a , int b )
  {
         if (a < b )
          {
            int temp = a ;
            a = b ;
            b = temp ;
          }
         if ( b == 0 )
           return( abs( a ));
         else
          return(gcd( b , a % b ));
  }
 //  this function computes d = gcd( u, v ) and integers a , b such that au + bv = d
 void euclid( int u , int v , int& d , int& a , int& b )
  {
  //     int const index = 2 ;
        int m[2][2] = {{ 1,0 }, {0,1} }, prod[2][2], quotient[2][2];
        int n = 0 , q , i , j  , temp;
        while( v != 0 )
          {
            q = u/v ;
            quotient[0][0] = q ;
            quotient[0][1] = 1 ;
            quotient[1][0] = 1 ;
            quotient[1][1] = 0 ;
            matmult( m, quotient , prod );
            for( i = 0 ; i < 2 ; i++ )
                    {
                     for( j = 0 ; j < 2 ; j++ )
                               m[i][j] = prod[i][j] ;
                     }
           temp = u ;
```

```
            u = v ;
            v = temp - (q * v ) ;
            n++ ;
           }
          d = u ;
          a = pow( -1 , n ) *  m[1][1] ;
          b = pow(-1 , ++n) *  m[0][1] ;
          return ;
  }
//matrix multiplication function
  void matmult( int a[2][2] , int b[2][2] , int c[2][2] )
           {
             int i , j , k ;
             for( i = 0 ; i < 2 ; i++ )
                         {
                                     for ( j = 0 ; j < 2 ; j++ )
                                      {
                                                  c[i][j] = 0 ;
                                                  for( k = 0 ; k < 2 ;  k++ )
                                                    c[i][j] = c[i][j] + a[i][k] * b[k][j] ;
                                      }
                          }
              return ;
           }
 // isprime() return 1 if a number is a prime else return 0
 int isprime(int prime)
 {
          int sum = 0  , q , r  ;
          for (int i = 1 ; i <= prime ; i++ )
           {
            q  =  prime / i ;
            r  =  prime - q * i ;
            if( r == 0 ) sum  =  sum  + 1 ;
           }
          if ( sum == 2 )
            return  1 ;
          else
          return 0 ;
  }
  // key_rsa()function calculate public and private key
  void rsa ( void )
  {
           int pp , qq , mx , ee , d,  f , g , temp1 , temp2, b ;
           char p[12] , q[12]   , e[12] ;
          // begin keys generation
           p1:             cout <<"\n\n\tenter a prime number,p :\t " ;
           cin >> p ;
           pp = atoi( p ) ;
           if ( isprime( pp ) )
            temp1 = pp - 1 ;  // calcu;ate Euler phi function of p ;
           else
           {
            cout << "\n\t"<<p<<" is not a prime number try, again." ;
            goto p1  ;
           }
```

```
p2:         cout <<"\n\tenter a prime number,q :\t " ;
cin >> q ;
qq = atoi(q );
if ( isprime( qq ) )
 temp2 = qq - 1 ;  // calcu;ate Euler phi function of q ;
else
{
 cout << "\n\t"<<q<<" is not a prime number try, again." ;
 goto p2  ;
 }
int n = pp * qq   ;
mx = temp1 * temp2 ; //  (p - 1 )( q - 1 )
b = (temp1 * temp2)/gcd(temp1, temp2) ;
cout <<"\n\n\tthe product p * q \t :\t"<< n;
cout <<"\n\n\tthe product (p -1)(q -1):\t"<< mx;
pub : cout <<"\n\n\tenter a public key,e     :\t" ;
cin >> e  ;
ee = atoi( e );
if ( gcd( ee , mx ) == 1  )
 cout <<"\n\n\tcorrect!  "<< e <<" is the public key for encryption " ;
else
 {
   cout << " \n\n\tthe public key("<< e <<") you chose is not a coprime of "
              <<"\n\n\t"<< mx <<"  please try again " ;
   goto pub ;
 }
//calculate private key
euclid( ee,mx,d, f,g );
t = 0 ;
while(f <= 0) f = f + (b * ++t) ;
cout << " \n\n\tthe required private key for decryption is\t" << f ;
return ;
}
```

3.0 Requirements

For implementation we need to two types of requirements namely:

- Hardware requirement
- Software requirement

4.1 Hardware requirement

- Personal computer (PC) (Pentium III system or higher)
- RAM size of at least 128 MB
- Hard disk type of at least 20GB
- Keyboard
- Mouse
- Stabilizer
- Uninterrupted Power Supply (UPS)

4.2 Software Requirement:

- Windows Operating System (version 98 or later version)
- Turbo C++ compiler.

5.0 Getting started

This section provides a user guide as to how to execute the implemented design. For simplicity reasons, it has been presented in steps:

***Step* 1**

How to Start the Program

- Switch on the PC
- Allow the PC to boot to the window desktop
- Double click the Turbo C++ icon on the desktop to take you to Integrate Development Environment (IDE)
- Click the FILE menu option or the menu bar, select OPEN from the dropdown list.
- Select the program name gcd_pro in a dialog box that appear and click the OPEN tab
- The source program will be displayed on the IDE

***Step* 2**

How to Compile the Program

- In the C++ IDE click PROJECT on the menu bar, on the PROJECT dropdown list, click compile.
- Alternatively press Alt + F7 for short cut on the keyboard.

***Step* 3**

How to Run the Program

- Click debug on the menu bar, on the debug dropdown list select RUN
- Alternatively press CTRL + F9 for short cut on the keyboard

Step 4 How to Exit the Program

- Click FILE option on the menu bar
- Click EXIT from the dropdown list

Note

If you install the gcd_pro CD in your system you do not need to pass through the C++ IDE to compile and run the program. All you need to do is to double click on the gcd_pro icon on the desktop after booting the system (or click on the start tab on the desktop, select program, and then click on the gcd_pro in the dropdown list) and the program starts execution.

6.0 System/user response at run-time

The system is a console based application. It is very interactive and user-friendly. It has been validated not to crash on any bad or invalid inputs.

When the program is running, the system will prompt you to enter the figures, character, strings and so on input any of these appropriately and press the ENTER key. If you mistakenly enter an invalid data the system will prompt you to enter the data again.

6.1 Menu options and their functions

6.1.1 Calculate gcd [5], [10]

Selecting this option enable you to calculate the gcd of two integers and display the result.

Example

[SYSTEM RESPONSE]	[USER RESPONSE]
Enter the first number	21 press enter key
Enter the second number	6 press enter key
Output: gcd(u,v)	3

6.1.2 Extended Euclidean Algorithm [1], [2], [3]

This option enables us to calculate the gcd of two integers say u and v, and integer coefficients x and y such that $\gcd(u,v) = ux + vy$

Example

[SYSTEM RESPONSE]	[USER RESPONSE]
Enter the first number	45 press enter key
Enter the second number	35 press enter key

Output: gcd(45,35) = 5, x = -3, y = 4 i.e. 5 = 45x + 35y

6.1.3 Application of gcd Algorithm

Selecting this option allow you to generate public and private keys for data encryption and decryption using the RSA Cryptographic algorithm.

6.1.4 Exit

This option when selected terminates program execution and you can then exit from the program.

7.0 Conclusion

As mentioned in section.2 of this paper the use of this software enhances accuracy, speed and optimal utilization of computer resources.

8.0 Recommendations

Numerous methods and algorithms were presented in this research for calculating the greatest common divisor. These methods yield the same if applied accurately. However, in terms of computer implementations some are not very good i.e. they might be inefficient and also waste the system resources. We recommend that the binary gcd algorithm should be used for computer implementation, because it only involves division by 2 and no modular operation is needed. This makes it faster for bit-wise operation, unlike the Euclidean algorithm and the prime factorization algorithms which are recursive and involve modular operations. This tends to waste more system resources and slow down processing.

References

[1] Abraham P.H and Gerald L.A. (1973), “A first graduate course in Abstract Algebra”, Wadsworth publishing company, Inc, Belmont California.

[2] Alexandra B, (2005), “Euclid Algorithm”, available online at: http://www.cut-the-knot.org/blue/Euclid.shtml, downloaded on the 26th October, 2005.

[3] Bogomolny A. (2005), “Two properties of the greatest common divisor”, available at: http://www.cut-the-knot.org/Generalization/gcd.shtml, downloaded on the 26th October, 2005.

[4] Dany K.(1999), “The ASI/ISO C++ professional programmers Handbook”, Que corporation, New York, USA.

[5] Eric B and Jeffrey S. (1996), “Algorithm Number Theory Volume 1: Efficient Algorithms”, MIT press publishing company, Cambridge; Massachusetts London; England.

[6] Eynden C.V. (1987), “elementary Number Theory”. McGraw publishimg company, Inc, NewYork, USA.

[7] Chaum D, Revest R.L and Shermans A.Y. (1984), Advances in Cryptology, proceeding of Crypto 82, Plenum; New York.

[8] Chaum D, Revest R.L and Shermans A.Y. (1984), Advances in Cryptology, proceeding of Crypto 83, Plenum; New York.

[9] Rivest R.L, Shamir A and Adleman L. (1978), A method for obtaining digital signatures and public key cryptosystem, communications of ACM Vol 21.

[10] Rosen K.H. (1992), Elementary Number Theory and its applications, Addison-Wesley publishing company, New York.

[11] Williams H.C. ed(1986), Advances in Cryptology, Crpto 85, Springer-Verlay, Berlin.

[12] Omokaro H.O. A generalization of the RSA Congruence, Journal of the Nigerian Association of Mathematical physics Vol. 7, PP 27, 2003.

[13] Omokaro H.O. An application of the extended RSA Congruence, Journal of the Nigerian Association of Mathematical physics Vol. 8, pp 341-343, 2004