FUNCTION CODES FOR PROTECTION OF KEY IN PRIVATE KEY CRYPTOGRAPHY

[1]Neha Tyagi, [2]Ashish Agarwal, [3]Anurag Katiyar, [4]Shubham Garg, [5]Shudhanshu Yadav
[1][2][3][4][5]Department of Computer Science & Engineering,
[1][2][3][4][5]G.L. Bajaj Institute of Technology & Management, Greater Noida, India

Abstract—In order to check the applicability and correctness of any designed algorithm, it is must to write a code on any suitable platform that successfully implements the logic. The present work is actually an extension to what we have done in our previous works where we have tried to devise an algorithm which is based on the most popular and largely applied asymmetric key algorithm named RSA algorithm [1]. The basic idea of latter is not touched but modified only with a piece of further enhancement in its working. The flexibility of RSA algorithm to wide range of values has instigated us to choose it for our efforts to ensure secure communication between communicating parties with the probability of getting failed minimised to great level of its implementation.

Index Terms—Asymmetric key Algorithm, Symmetric key Algorithm, R.S.A. Algorithm, Encryption, Decryption, Cryptography

I. INTRODUCTION

Private key cryptography is simplest form of cryptography and easier to implement. It is easily targeted by the attackers to steal away the information or pamper the secret data that is needed to be kept confidential [2] between communicating parties. But advancements in the techniques make this competition much more yielding as can be noticed in the way, newer technologies with advanced features are being adopted and put to implementations in order to deal with the threats. Prior to taking our work to next level, we have already discussed the security measures that do carry worth potential to be thought of as an alternative(s) to already functioning tools. In this regard, we have suggested to consider an asymmetric algorithm [3] with certain level of modification to its structure, though the basic functionality as designed by its developers has been in use for long. As evident from works by other experts in this domain, increasing mathematical computational complexities may help in building robust cryptosystem(s) with no compromise in the basic idea of this methodology i.e. confidentiality. So, we have thought of applying some binary calculations in form of power of 2 and use the most applicable techniques of trailers and headers to introduce an enhanced amount of safety to our system. Calculating the median of the complex entity so generated by means of applying former method(s) may prove strong factor in safeguarding the information in transit between communicating ends.
In order to render easy understanding of different sections of our code, we have presented here separate program functions for each of them. As already known, the two main processes on which relies any cryptosystem are encryption and decryption respectively.

II. RELATED DISCUSSION

We have written our code in C programming language taking into account the secure working environment it provides to build strong working piece. The different code fragments supported with proper exemplification in order to justify our idea have been illustrated as follows.

III. C PROGRAM FUNCTIONS

1) **Header Files used**

   stdio.h, stdlib.h, math.h, string.h

2) **Functions**

   2.1) int prime(long int) = To check whether p and q are prime numbers or not
   2.2) void ce() = Compute values of e
   2.3) long int cd(long int) = Compute values of d
   2.4) void encrypt() = Encrypt the message
   2.5) void decrypt() = Decrypt the message

3) **Variables and their meaning**

   p,q = Two prime numbers as per conventional R.S.A. algorithm
   n = p*q
   t = (p-1) * (q-1)
   i,j,flag = Variables for intermediate computation
   l = Variable to hold length of message
   k,k1 = Variable to hold median of message length
   mm = Variable to hold 2^k
   e[100] = Array to store values of e (public key)
   d[100] = Array to store values of d (private key)
   temp[100] = Variable array used for intermediate computations
   m[100] = Message array
   en[100] = Encrypted Message Array

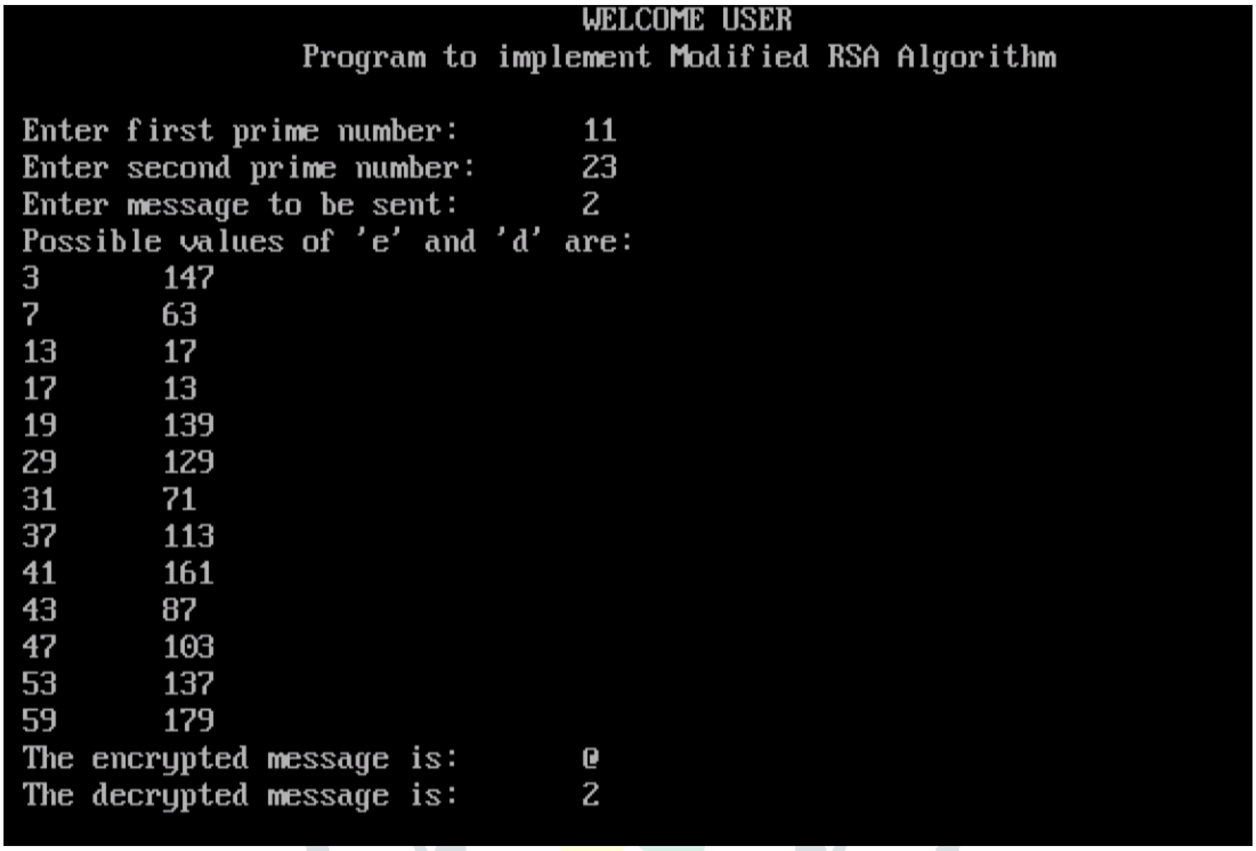
4) **Data Types of variables**

```
long int i,j,p,q,n,t,flag,e[100],d[100],temp[100],m[100],  en[100];
float k1=0.0,k2=0.0,l=0.0;
int mm=0,k=0;
```

5) **Function Body**

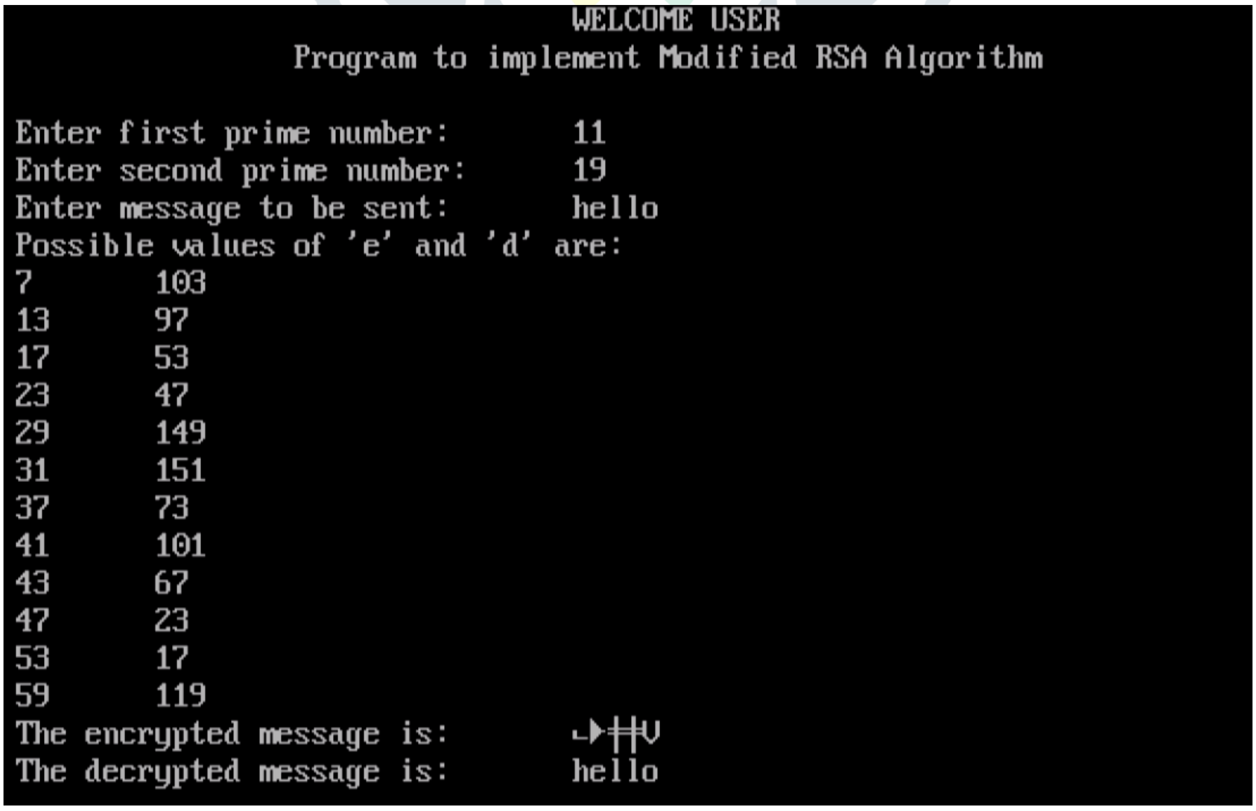
```
int prime(long int pr) //Function 2.1
        {int i;
         j=sqrt(pr);
         for (i=2;i<=j;i++)
                {if(pr%i==0)
                 return 0;
                }
        return 1;
         }
void ce() // Function 2.2
        {int k=0;
         for (i=2;i<t;i++)
                {if(t%i==0)
                 continue;
                 flag=prime(i);
                 if(flag==1&&i!=p&&i!=q)
                        {e[k]=i;
                         flag=cd(e[k]);
                         if(flag>0)
                                {d[k]=flag;
                                 k++;
                                 }
                         if(k==99)
                         break;
                         }
                 }
        }
long int cd(long int x) //Function 2.3
        {long int k=1;
         while(1)
                {k=k+t;
                 if(k%x==0)
                 return(k/x);
                 }
         }
void encrypt() //Function 2.4
        {long int pt,ct,key=e[0],k,len;
         i=0;
         len=strlen(msg);
         while(i!=len)
                {pt=m[i];
                pt=pt-96;
                k=1;
                for (j=0;j<key;j++)
                        {k=k*pt;
                        k=k%n;
                        }
                temp[i]=k;
                ct=k+96;
                en[i]=ct;
                i++;
                }
        en[i]=-1;
        printf("The encrypted message is:\t");
        for (i=0;en[i]!=-1;i++)
        printf("%c",en[i]*mm);
        }
void decrypt() // Function 2.5
        {long int pt,ct,key=d[0],k;
         i=0;
         while(en[i]!=-1)
                {ct=temp[i];
                 k=1;
                 for (j=0;j<key;j++)
                        {k=k*ct;
                         k=k%n;
                         }
                 pt=k+96;
                 m[i]=pt;
                 i++;
                 }
         m[i]=-1;
         printf("\nThe decrypted message is:\t");
         for (i=0;m[i]!=-1;i++)
         printf("%c",m[i]);
         }
```

IV. PROGRAM SCREENSHOTS



```
                              WELCOME USER
                 Program to implement Modified RSA Algorithm

Enter first prime number:        11
Enter second prime number:       23
Enter message to be sent:        2
Possible values of 'e' and 'd' are:
3       147
7       63
13      17
17      13
19      139
29      129
31      71
37      113
41      161
43      87
47      103
53      137
59      179
The encrypted message is:        @
The decrypted message is:        2
```



```
                              WELCOME USER
                 Program to implement Modified RSA Algorithm

Enter first prime number:        11
Enter second prime number:       19
Enter message to be sent:        hello
Possible values of 'e' and 'd' are:
7       103
13      97
17      53
23      47
29      149
31      151
37      73
41      101
43      67
47      23
53      17
59      119
The encrypted message is:        ∟►╪╪√
The decrypted message is:        hello
```

V. FUTURE SCOPE

The time and space complexities of the code fragments are two main points to be considered as increasing complexity on one side is helpful in rendering enhanced security to information exchange, at the same time, we need to consider the system on which our cryptosystem is implemented. We also need to consider the underlying operating system's compatibility to run our code. Therefore, making efforts in this regard is equally important and carry potential for to make further effort(s). There may be some values which may encounter as we move ahead that may not fit our framework, so to make our code deal with such values also carries a scope to work on. The efforts are to make it robust against most of the security threats that we generally encounter in network communication and overcome the shortcomings that other existing tools(s) prove incomplete to deal with.

VI. CONCLUSION

Cryptography field cannot be confined with just a few techniques. It is a domain which requires introducing new tools in a consistent and comprehensive manner. Having glimpse on the wide range of security attacks over the years and specifically in today's context where everything from smallest to biggest is on the path to go digital, it is of utmost importance to look for more strengthened security tools. It is not mandatory to think something new in its approach but we can also try to add something new to the existing methods that we have tried. We hope that our program code will prove to be helpful in dealing with the challenge(s) covering the communication.

VII. REFERENCES

[1] Neha Tyagi, Ashish Agarwal, Anurag Katiyar, Shubham Garg, Shudhanshu Yadav, "Protection of Key in Private Key Cryptography" published by "International Journal of Advanced Research", Volume 5, Issue 2, Feb 2017.

[2] Ayushi, "A symmetric Key Cryptographic Algorithm" published by "International Journal of Computer Applications", Volume 1, Issue No. 15, 2010.

[3] Prof. Mukund R. Joshi, Renuka Avinash Karkade, "Network Security with Cryptography" published by "International Journal of Computer Science and Mobile Computing", Volume 4, Issue 1, Jan 2015.