PPS

Programming for Problem Solving

Mini Project

Aditya Jain

RA2111019010064

Problem Statement

Write a program in C to implement the RSA Algorithm for Encryption and Decryption.

Analysis:

* Accept the Prime numbers
* Accept the message
* Calculate the Public and Private Keys
* Calculate the encrypted and decrypted message
* Display the encrypted and decrypted message

FDT-Function Description Table

|  |  |  |  |
| --- | --- | --- | --- |
| Function Name | Return Type | Purpose | Parameter List |
| main | int | To take user’s inputs | - |
| prime | int | To check whether a number is prime or not | long int pr |
| ce | void | To calculate the e and d values | - |
| cd | long int | To calculate the corresponding d values | long int x |
| encrypt | void | To encrypt the message | - |
| decrypt | void | To decrypt the message | - |

Algorithm

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private Key is kept private.

An example of asymmetric cryptography:

* A client (for example browser) sends its public key to the server and requests for some data.
* The server encrypts the data using client’s public key and sends the encrypted data.
* Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

The mechanism behind RSA algorithm:

>> Generating Public Key:

• Select two prime no's. Suppose P = 53 and Q = 59.

• Now First part of the Public key: n = P\*Q = 3127.

• We also need a small exponent say e:

• But e must be an integer, not a factor of n.

• 1 < e < Φ(n) [Φ(n) is discussed below],

• Let us now consider it to be equal to 3.

• Our Public Key is made of n and e.

>> Generating Private Key:

• We need to calculate Φ (n):

• Such that Φ (n) = (P-1) (Q-1) so, Φ (n) = 3016

• Now calculate Private Key, d:

d = (k\*Φ (n) + 1) / e for some integer k

For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key (d = 2011)

Now we will encrypt “HI”:

• Convert letters to numbers: H = 8 and I = 9

• Thus Encrypted Data c = 89e mod n.

Thus our Encrypted Data comes out to be 1394

• Now we will decrypt 1394:

• Decrypted Data = cd mod n.

Thus our Decrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

Source Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RSA Algorithm for Encryption and Decryption

Implementation of the RSA Algorithm for small values

Aditya Jain RA2111019010064

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;

char msg[100];

int prime(long int);// To check whether the number is prime

void ce(); // To generate e and d values

long int cd(long int); // To generate the consecutive d value

void encrypt(); // To encrypt the message

void decrypt(); // To decrypt the message

void main()

{

printf("ENTER FIRST PRIME NUMBER\n");

scanf("%d",&p);

flag=prime(p);

if(flag==0)

{

printf("WRONG INPUT\n");

getch();

exit(1);

}

printf("ENTER ANOTHER PRIME NUMBER\n");

scanf("%d",&q);

flag=prime(q);

if(flag==0||p==q)

{

printf("WRONG INPUT\n");

getch();

exit(1);

}

printf("ENTER MESSAGE\n");

fflush(stdin);

scanf("%s",msg);

for (i=0;msg[i]!='\0';i++)

m[i]=msg[i];

n=p\*q; // Public Key

t=(p-1)\*(q-1); // Private key

ce();

printf("POSSIBLE VALUES OF e AND d ARE\n");

for (i=0;i<j-1;i++)

printf("\n%d\t%d",e[i],d[i]);

encrypt();

decrypt();

getch();

}

int prime(long int pr)

{

int i;

j=sqrt(pr);

for (i=2;i<=j;i++)

{

if(pr%i==0)

return 0;

}

return 1;

}

void ce()

{

int k;

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)

{

long int k=1;

while(1)

{

k=k+t;

if(k%x==0)

return(k/x);

}

}

void encrypt()

{

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("\nTHE ENCRYPTED MESSAGE IS\n");

for (i=0;en[i]!=-1;i++)

printf("%c",en[i]);

}

void decrypt()

{

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\n");

for (i=0;m[i]!=-1;i++)

printf("%c",m[i]);

}

VDT-Variable Description Table

Sample Input Output

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Data Type | Purpose | Scope |
| p | int | To store the first prime number | Program |
| q | int | To store the second prime number | Program |
| n | int | To store the public key | Program |
| t | int | To store the private key | Program |
| flag | int | To check for prime numbers | Program |
| e[] | int | To store the possible key values | Program |
| d[] | int | To store the possible key values | Program |
| temp[] | int | To calculate the encrypted and decrypted messages | Program |
| j | int | To calculate the encrypted and decrypted messages | Program |
| m[] | int | To store the encrypted message | Program |
| en[] | int | To store the decrypted message | Program |
| i | int | LCV-Loop Control Variable | Program |
| msg[] | char | To accept the message to be encrypted | Program |
| pr | long int | Formal variable to copy the number to be checked | int prime() |
| k | long int | To calculate the encrypted and decrypted messages | void ce(), long int cd(), void encrypt(),void decrypt() |
| x | long int | To calculate the encrypted and decrypted messages | long int cd() |
| pt | long int | To calculate the encrypted and decrypted messages | void encrypt(),void decrypt() |
| ct | long int | To calculate the encrypted and decrypted messages | void encrypt(),void decrypt() |
| key | long int | To store the private key parts | void encrypt(),void decrypt() |
| len | long int | To store the length of the message | void encrypt(),void decrypt() |

Sample Input Output





