Experiment No: 9 **Date:**

RSA Algorithm to Encrypt and Decrypt the Data

Aim: Program for Simple RSA Algorithm to encrypt and decrypt the data

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted *e* and *d*, respectively) and taking the remainder of the division with *N*. A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo.

A very simple example of RSA encryption

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 can probably even do it by hand on paper).

1. Select primes p = 11, q = 3.
2. n = pq = 11.3 = 33

   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3

   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),

   and check gcd(e, q-1) = gcd(3, 2) = 1

   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that ed ≡ 1 (mod phi)

   i.e. compute d = e^{-1} mod phi = 3^{-1} mod 20

   i.e. find a value for d such that phi divides (ed-1)

   i.e. find d such that 20 divides 3d-1.

   Simple testing (d = 1, 2, ...) gives d = 7

   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.
5. Public key = (n, e) = (33, 3)

   Private key = (n, d) = (33, 7).

   This is actually the smallest possible value for the modulus n for which the RSA algorithm works.

Now say we want to encrypt the message m = 7,

c = m^e mod n = 7^3 mod 33 = 343 mod 33 = 13.

Hence the ciphertext c = 13.

To check decryption we compute

$m' = c^{\wedge d} \bmod n = 13^{\wedge 7} \bmod 33 = 7.$

Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

One way of calculating m' is as follows:-

$m' = 13^{\wedge 7} \bmod 33 = 13^{\wedge(3+3+1)} \bmod 33 = 13^{\wedge 3}.13^{\wedge 3}.13 \bmod 33$

$= (13^{\wedge 3} \bmod 33).(13^{\wedge 3} \bmod 33).(13 \bmod 33) \bmod 33$

$= (2197 \bmod 33).(2197 \bmod 33).(13 \bmod 33) \bmod 33$

$= 19.19.13 \bmod 33 = 4693 \bmod 33$

$= 7.$

Now if we calculate the cipher text c for all the possible values of m (0 to 32), we get

m 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

c 0 1 8 27 31 26 18 13 17 3 10 11 12 19 5 9 4

m 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

c 29 24 28 14 21 22 23 30 16 20 15 7 2 6 25 32

Note that all 33 values of m (0 to 32) map to a unique code c in the same range in a sort of random manner. In this case we have nine values of m that map to the same value of c - these are known as *unconcealed messages*. m = 0 and 1 will always do this for any N, no matter how large. But in practice, higher values shouldn't be a problem when we use large values for N.

If we wanted to use this system to keep secrets, we could let A=2, B=3, ..., Z=27. (We specifically avoid 0 and 1 here for the reason given above). Thus the plaintext message "HELLOWORLD" would be represented by the set of integers m1, m2, ...

{9,6,13,13,16,24,16,19,13,5}

Using our table above, we obtain ciphertext integers c1, c2, ...

{3,18,19,19,4,30,4,28,19,26}

Note that this example is no more secure than using a simple Caesar substitution cipher, but it serves to illustrate a simple example of the mechanics of RSA encryption.

Remember that calculating m^e mod n is easy, but calculating the inverse c^-e mod n is very difficult, well, for large n's anyway. However, if we can factor n into its prime factors

p and q, the solution becomes easy again, even for large n's. Obviously, if we can get hold of the secret exponent d, the solution is easy, too.

Key Generation Algorithm

1. Generate two large random primes, p and q, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits. [See note 1].
2. Compute n = pq and (φ) phi = (p-1)(q-1).
3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1. [See note 2].
4. Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi). [See note 3].
5. The public key is (n, e) and the private key is (n, d). The values of p, q, and phi should also be kept secret.

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

Encryption

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer m [see note 4].
3. Computes the ciphertext c = $m^{\wedge e}$ mod n.
4. Sends the ciphertext c to B.

Decryption

Recipient B does the following:-

1. Uses his private key (n, d) to compute m = $c^{\wedge d}$ mod n.
2. Extracts the plaintext from the integer representative m.

Source Code:

```
import java.util.*;
import java.io.*;
public class rsa
{       static int gcd(int m,int n)
        {       while(n!=0)
                  {       int r=m%n;
                          m=n;
                          n=r;
                  }
                return m;
        }
```

```
public static void main(String args[])
{
        int p=0,q=0,n=0,e=0,d=0,phi=0;
        int nummes[]=new int[100];
        int encrypted[]=new int[100];
        int decrypted[]=new int[100];

        int i=0,j=0,nofelem=0;
        Scanner sc=new Scanner(System.in);
        String message ;
        System.out.println("Enter the Message to be encrypted:");
        message= sc.nextLine();
        System.out.println("Enter value of p and q\n");
                p=sc.nextInt();
        q=sc.nextInt();
        n=p*q;
        phi=(p-1)*(q-1);

        for(i=2;i<phi;i++)
        if(gcd(i,phi)==1) break;
        e=i;

        for(i=2;i<phi;i++)
        if((e*i-1)%phi==0)
                break;
        d=i;

        for(i=0;i<message.length();i++)
        {
                char c = message.charAt(i);
        int a =(int)c;
        nummes[i]=c-96;

        }
        nofelem=message.length();
        for(i=0;i<nofelem;i++)
        {
                encrypted[i]=1;
                for(j=0;j<e;j++)
                encrypted[i] =(encrypted[i]*nummes[i])%n;
        }
        System.out.println("\n Encrypted message\n");
        for(i=0;i<nofelem;i++)
        {
                System.out.print(encrypted[i]);
                System.out.print((char)(encrypted[i]+96));
        }
        for(i=0;i<nofelem;i++)
        {       decrypted[i]=1;
                for(j=0;j<d;j++)
```

```
                    decrypted[i]=(decrypted[i]*encrypted[i])%n;
                }

                System.out.println("\n Decrypted message\n ");
                for(i=0;i<nofelem;i++)
                System.out.print((char)(decrypted[i]+96));
                return;
        }

        }
```

#**********

****RESULT****

```
Enter the text:
hello
Enter the value of P and Q:
5
7
Encrypted Text is: 8 h 10 j 17 q 17 q 15 o
Decrypted Text is: hello
```