**IMPLEMENTATION OF RSA ALGORITHM**

**EX.NO. : 5**

**DATE :**

**AIM:**

To implement RSA Algorithm

**ALGORITHM :**

1.Start

2. Choose two prime numbers p and q.

3. Compute n = p\*q.

4. Calculate phi = (p-1) \* (q-1).

5. Choose an integer e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1; i.e., e and phi(n) are coprime.

6. Calculate d as d = e-1 (mod phi(n)); here, d is the modular multiplicative inverse of e modulo phi(n).

7. For encryption, c = me mod n, where m = original message.

8. For decryption, m = c d mod n.

9.Stop

**SOURCE CODE :**

#include<stdio.h>

#include<math.h>

//to find gcd

int gcd(int a, int h)

{

int temp;

while(1)

{

temp = a%h;

if(temp==0)

return h;

a = h;

h = temp;

}

}

int main()

{

//2 random prime numbers

double p = 3;

double q = 7;

double n=p\*q;

double count;

double totient = (p-1)\*(q-1);

//public key

//e stands for encrypt

double e=2;

//for checking co-prime which satisfies e>1

while(e<totient){

count = gcd(e,totient);

if(count==1)

break;

else

e++;

}

//private key

//d stands for decrypt

double d;

//k can be any arbitrary value

double k = 2;

//choosing d such that it satisfies d\*e = 1 + k \* totient

d = (1 + (k\*totient))/e;

double msg = 12;

double c = pow(msg,e);

double m = pow(c,d);

c=fmod(c,n);

m=fmod(m,n);

printf("Message data = %lf",msg);

printf("\np = %lf",p);

printf("\nq = %lf",q);

printf("\nn = pq = %lf",n);

printf("\ntotient = %lf",totient);

printf("\ne = %lf",e);

printf("\nd = %lf",d);

printf("\nEncrypted data = %lf",c);

printf("\nOriginal Message Sent = %lf",m);

return 0;

}

**OUTPUT :**

Message data = 12.000000

p = 3.000000

q = 7.000000

n = pq = 21.000000

totient = 12.000000

e = 5.000000

d = 5.000000

Encrypted data = 3.000000

Original Message Sent = 12.000000

**RESULT :**

Thus the RSA Algorithm is executed successfully

**IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE ALGORITHM**

**EX.NO. : 6**

**DATE :**

**AIM:**

To implement Diffie Hellman key Exchange Algorithm

**ALGORITHM :**

Key Generation

1. Select a large random prime p and a generator ? of Z?p.

2. Generate a random integer x such that 1?x?p?2.

3. Compute y = ?\*\*x mod p.

4. A’s public key is (p, ?, y).

5. A’s private key is x.

Signature Generation

A generates a signature for a message m (0 ? m < p?1) as follows:

1. Generatea random integer k such that 1?k?p?2 and gcd(k,p?1)=1.

2. Compute r = ?\*\*k mod p.

3. Compute k\*\*?1 mod (p ? 1).

4. Computes=k\*\*?1(m?xr)mod(p?1).

5. A’s signature for m is the pair (r, s),

Signature Verification

A signature (r, s) produced by A can be verified as follows:

1. Verify that 1 ? r ? (p?1); if not return False.

2. Compute v1 = (y\*\*r)(r\*\*s) mod p.

3. Compute v2 = ?\*\*m mod p.

4. Return v1 = v2.

**SOURCE CODE :**

from random import randint

if \_\_name\_\_ == '\_\_main\_\_':

# Both the persons will be agreed upon the

# public keys G and P

# A prime number P is taken

P = 23

# A primitve root for P, G is taken

G = 9

print('The Value of P is :%d'%(P))

print('The Value of G is :%d'%(G))

# Alice will choose the private key a

a = 4

print('The Private Key a for Alice is :%d'%(a))

# gets the generated key

x = int(pow(G,a,P))

# Bob will choose the private key b

b = 3

print('The Private Key b for Bob is :%d'%(b))

# gets the generated key

y = int(pow(G,b,P))

# Secret key for Alice

ka = int(pow(y,a,P))

# Secret key for Bob

kb = int(pow(x,b,P))

print('Secret key for the Alice is : %d'%(ka))

print('Secret Key for the Bob is : %d'%(kb))

**OUTPUT :**

The Private Key b for Bob is :3

Secret key for the Alice is : 9

Secret Key for the Bob is : 9

**RESULT :**

Thus, the Diffie hellman Key exchange Algorithm is executed successfully.

**IMPLEMENTATION OF DIGITAL SIGNATURE**

**USING RSA DIGITAL SIGNATURE**

**EX.NO. : 7**

**DATE :**

**AIM:**

To implement Digital Signature Using RSA Digital Signature Algorithm

**ALGORITHM :**

1.Start

2.Create a KeyPairGenerator object.

3. Initialize the KeyPairGenerator object.

4 Generate the KeyPairGenerator. ...

5 Get the private key from the pair. ...

6 Create a signature object. ...

7 Initialize the Signature object.

8.Stop

**SOURCE CODE :**

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.Signature;

import sun.misc.BASE64Encoder;

public class digijava {

public static void main(String[] args) throws Exception { // TODO code application logic here

KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA"); kpg.initialize(1024);

KeyPair Pair = kpg.genKeyPair();

byte[] data = "Sample Text".getBytes("UTF8"); Signature sig = Signature.getInstance("MD5WithRSA");

sig.initSign(Pair.getPrivate());

sig.update(data);

byte[] signatureBytes = sig.sign();

System.out.println("Signature: \n" + new BASE64Encoder().encode(signatureBytes)); sig.initVerify(Pair.getPublic());

sig.update(data);

System.out.println(sig.verify(signatureBytes));

}

}

**OUTPUT :**

Signature:

imwaKe99tkM6H6hiiP0rubmb/MrYJZLiwLdRSjslF2KlA5B23az5M2LKftQFCB+NH

Ce5F5/YfN8OsNSNLtucrrZTah0SrdWSzdGCOfYLdUZmPQ72j1SkLhYspsTsUb/U6 FPSYT4QebNSYobDtjKujkHdRimHI9TO4lLuqVQRdWU= true

**RESULT :**

Thus the Digital Signature Algorithm using RSA Digital Signature is executed successfully.