# Program- 9

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. 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 anumber (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. The RSA algorithm comprises of three steps, which are depicted below:

## Key Generation Algorithm

1. Generate two large random primes, p and q, of approximately equal size such that their product n = p*q
2. Compute n = p*q and Euler's totient function (φ) phi(n) = (p-1)(q-1).
3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.
4. Compute the secret exponent d, 1 < d < phi, such that e*d ≡ 1 (mod phi).
5. The public key is (e, n) and the private key is (d, n). The values of p, q, and phi should also be kept secret.

## Encryption

Sender A does the following:-

1. Using the public key (e,n)
2. Represents the plaintext message as a positive integer M
3. Computes the cipher text C = M^e mod n.
4. Sends the cipher text C to B (Receiver).

## Decryption

Recipient B does the following:-

1. Uses his private key (d, n) to compute M = C^d mod n.
2. Extracts the plaintext from the integer representative m.

**Source Code**:

```
import
java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import
java.util.Random; public
class RSA
{
private BigInteger p,q,N,phi,e,d;
private int bitlength=1024;
private Random r;
public RSA()
{
r=new Random();
p=BigInteger.probablePrime(bitlength,r);
q=BigInteger.probablePrime(bitlength,r);
System.out.println("Prime number p is"+p);
System.out.println("prime number q is"+q);
N=p.multiply(q);
phi=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e=BigInteger.probablePrime(bitlength/2,r);
while(phi.gcd(e).compareTo(BigInteger.ONE)>0&&e.compareTo(phi)<0)
{
e.add(BigInteger.ONE);
}
System.out.println("Public key is"+e);
d=e.modInverse(phi);
System.out.println("Private key is"+d);
}
public RSA(BigInteger e,BigInteger d,BigInteger N)
```

```
{
this.e=e;
```

```
this.d=d;
this.N=N
;
}
public static void main(String[] args)throws IOException
{
RSA rsa=new RSA();
DataInputStream in=new DataInputStream(System.in);
String testString;
System.out.println("Enter the plain text:");
testString=in.readLine();
System.out.println("Encrypting string:"+testString);
System.out.println("string in bytes:"+bytesToString(testString.getBytes()));
byte[] encrypted=rsa.encrypt(testString.getBytes());
byte[] decrypted=rsa.decrypt(encrypted);
System.out.println("Dcrypting Bytes:"+bytesToString(decrypted));
System.out.println("Dcrypted string:"+new String(decrypted));
}
private static String bytesToString(byte[] encrypted)
{
String test=" ";
for(byte
b:encrypted)
{
test+=Byte.toString(b);
}
return test;
}
public byte[]encrypt(byte[]message)
{
```

```
return(new BigInteger(message)).modPow(e,N).toByteArray();
}
```

```
public byte[]decrypt(byte[]message)
{
return(new BigInteger(message)).modPow(d,N).toByteArray();
}
}
```

*OUTPUT*

```
user@user-OptiPlex-3050:~/Desktop$ javac RSA.java
Note: RSA.java uses or
overrides a deprecated
API. Note: Recompile
with -Xlint:deprecation
for details.
user@user-OptiPlex-30
50:~/Desktop$ java RSA
Prime number p
is9434041555966760419639699719435550062061726189236379933318613058155503055902225
8412648913166938447749
3961729428365096220955385005483376543907713311494557955147433272886732679976160671070727712342726994 73
3162584335398183813161086772539485207170660797421308721913910268769789383015952978417294
50104777515744
329
prime number q
is1629598763359750702207903579265136270865303408645117941279139515901843403968657
4419084445457091309019
9580113566820777621671168006905128610948236857294124930049854806951514183645573 5
7441519382087492354812
3032812044432931292848736615654197347263976899910566745149374649461981244760589513416639
06940428877098
9747
Public key
is1253531723193161509753529267918552214239812435914857863149186755566054159490814
2243803236928383564409 0912254854167494838496767313072925642546688763477901 13
Private key
is1344706880345414725001654809915499428800120500205202938088936644254244800976690
2483706511629378778814
2789900344967384343100612773800789922040313687017872188482586221945968694333415 6
9244363873935384095105
5926165110014337598697391648637283364171285558916224971739406680661949116920057384111991
65928696303164
5237793100278291950667237988329177494311051276135991727609807177530173552280631256685060
68598724533589
1043279316573169297587771633591336635849467735200401267468604631713173628047213434141917
```

34927496021690
915795912693044571397754154768310721320751218537689010905006735039126790700979397990051206085125951747
130833
Enter the plain text:
Hello everyone
Encrypting string:Hello everyone
string in bytes:
721011081081113210111810111412111110101 Dcrypting Bytes:
721011081081113210111810111412111110101
Dcrypted string:Hello everyone