Practical NO:- 6

Aim: Write a program to implement RSA Algorithms

Theory: 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 public key is given to everyone and private key is kept private.

An example of Asymmetric cryptography
1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives the 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 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

```
{
 String s = "prathamesh";
System.out.println("MD5 hash:");
System.out.println("Your Hashcode Generated by
                    MD5 is: "+getMd5(s));
}
}
```

Result: Hence, program to implement message digest (MD-5) algorithm is implemented successfully.

hexadecimal Numericals values. Each stage includes four message-digest passes, which manipulate values in the current data block and values processed from the previous block. The final values computed from the last block becomes the MD5 digest for that block.

Program: 
```
           import java.math.BigInteger;
           import java.security.MessageDigest;
           import java.security.NoSuchAlgorithmException
  public class main {
     public static String getMd5( String input )
  {
    try {
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] messageDigest = md.digest (input.getBytes());
BigInteger no = new BigInteger(1, messageDigest);
  String hashtext = no.toString(16);
while (hashtext.length() < 32) {
      hashtext = "0" + hashtext;
  }
  return hashtext;
  }
     Catch (NoSuchAlgorithmException e) {
    throw new RuntimeException(e);
   }
 }
  Public static void main( String args[]) throws
NoSuchAlgorithmException
```

Practical NO:- 8

RAISONI GROUP
a vision beyond

Aim: Write a program to implement message Digest - 5 (MD-5)

Theory: • What is MD5 ?

The MD5 (message - digest algorithm) hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed - length digest value to be used for authenticating the original message.

The MD5 Hash function was originally designed for use as a secure cryptographic hash algorithm for authenticating digital signatures. But MD5 has been deprecated for uses other than a noncryptographic checksum to verify data integrity and detect unintentional data corruption.

• How does it works ?

The MD5 message - Digest hashing algorithm process data in a 512 bit strings, broken down into 16 words composed of 32 bit each. The output from MD5 is a 128 - bit message - digest value.

Computation of the MD5 digest value is performed in separate stages that process each 512 - bit block of data along with the value computed in the preceding stage. The first stage begins with the message - digest values initialized using consecutive

RAISONI GROUP
a vision beyond

$y = (8^2 \bmod 33) = 64 \bmod 33 = 31$

5. user 1 and user 2 exchange public keys, i.e 17 and 31.

6. user1 receives public key $y = 31$ and user 2 receives public key $x = 17$.

7. user1 and user 2 calculate symmetric keys:

user 1 ~~and~~ :- $k_a = y_a \bmod p = 31_3 \bmod 33$

$= 29791 \bmod 33$

$= 25$.

user 2 : $k_b = x_b \bmod p = 17_2 \bmod 33 = 289 \bmod 33$

$= 25$

8. 25 is the shared secret.

# Program :- class main

```
{
private static long power (long a, long b, long P)
{
  if (b==1)
  return a;
  else
  return ((long) math.pow (a,b) %P);
}
  public static void main (String [] args)
{
  long P, G, x, a, b, y, ka, kb;
  P = 23;
System.out.println ("Implementation of Diffie-
 Hellman Algorithm:");
System.out.println ("the value of P: "+P);
   G = 9;
System.out.println ("the value of G : "+G);
a = 4 ;
System.out.println ("The Private key a for
 Alice : " + a) ;
  x = power (G, a, p) ;
  b = 3
System.out.println ("The private key b for bob :
                        " + b) ;
 y = power (y, a, p) ;  //Secret key for Alice
 y = power (G, b, p) ;
 ka = power (y, a, p) ;  //Secrete key for Alice
 kb = power (x, b, p) ;  //Secrete key for Bob

System.out.println ("Secrete key for the Alice
                  is : " + ka) ;
System.out.println ("Secrete key for the Bob is :
                        " + kb) ;
}
}
```

**Result**: Hence A program to implement Diffie Hellman key exchange Algorithm is implemented Successfully.

Practical No:-7

Aim: Write a program to implement Diffie-Hellman key exchange Algorithm

Theory: The Diffie-Hellman Algorithm is being used to establish a shared secrete that can be used for secrete Communications while exchanging data over a public Network using the elliptic Curve to generate points and get the secrete key using parameters.

• For the sake of simplicity and practical implementation of the Algorithm, we will consider only 4 variables, one prime p and G (a primitive root of p) and two private values a and b.

• P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secrete key to encrypt.

Example.

1. User1 and user2 get public keys p=33 and G=8
2. User selects as a private key i.e 3, and user2 selects b as a private key, i.e 2.
3. User1 calculate the public value:

$$x = (8^3 \bmod 33) = 512 \bmod 33 = 17$$

4. User2 calculate the public value

RAISONI GROUP
a vision beyond

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 key can be typically 1024 or 2048 bits long, but experts believe that 1024 bits keys could be ~~[illegible]~~ broken in the near future. But still now it seems to be an infeasible task.

program :-
```
                import java.util.*;
                import java.math.*;
        class main
     {
        public static void main (String args[])
{
   int p, q, n, z, d = 0, e, i;
   int msg = 12;
   double c;
   BigInteger msgback;
     p = 3;
   System.out.println ("RSA Algorithm:");
       q = 11;
      n = p * q;
      z = (p-1) * (q-1)
System.out.println(" The value of c = "+ z);
  for ( e=z; i<z; i++)
{
   if (gcd (e,z) == 1 )
   { break;
   }
}
System.out.println("The value of e = " +e);
for ( i=0; i<=9, i++)
int x = 1 + (i*z);
if (x%e == 0)
{
  d = x/e;
  break;
}
} System.out.println("The value of d = "+d);
c = (math.pow (msg.e)) %n;
System.out.println ("Encrypted Message is : "+c);
BigInteger N = BigInteger.ValueOf(n);
BigInteger C = BigDecimal.valueOf (c).toBigInteger
 ();
msgback = (C.pow (d)).mod(N);
 System.out.println ("Decrypted Message is : "+msgback);
}
static int gcd (int e, int z)
{ if (e==0)
  return z;
  else
  return gcd (z%e, e);
}
}
```

Result: Thus a Program to implement RSA algorithm is executed Successfully.