Ex.No. : 1 (a) DATA ENCRYPTION STANDARD (DES)
Date :

AIM:

To apply Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption.

ALGORITHM:

1. Create a DES Key.
2. Create a Cipher instance from Cipher class, specify the following information and separated by a slash (/).
   - Algorithm name
   - Mode (optional)
   - Padding scheme (optional)
3. Convert String into Byte[] array format.
4. Make Cipher in encrypt mode, and encrypt it with Cipher.doFinal() method.
5. Make Cipher in decrypt mode, and decrypt it with Cipher.doFinal() method.

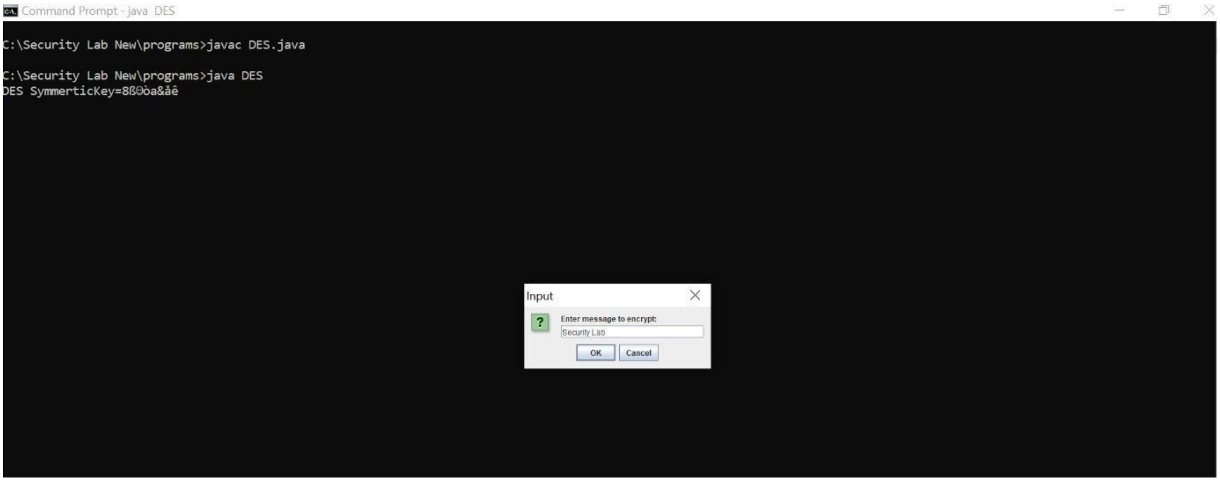
PROGRAM:

```
import javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random;
class DES
{
	byte[] skey=new byte[1000];
	String skeystring;
	static byte[] raw;
	String inputmessage,encryptedata,decryptedmessage;
	public DES()
	{
		try
		{
			generatesymmetrickey();
			inputmessage=JOptionPane.showInputDialog(null,"Enter message to
			encrypt:");
			byte[] ibyte =inputmessage.getBytes();
			byte[] ebyte=encrypt(raw, ibyte);
			String encrypteddata=new String(ebyte);
			System.out.println("Encrypted message:"+encrypteddata);
			JOptionPane.showMessageDialog(null,"Encrypted
			Data"+"\n"+encrypteddata);
			byte[] dbyte=decrypt(raw,ebyte);
			String decryptedmessage=new String(dbyte);
			System.out.println("Decrypted message:"+decryptedmessage);
```

```
                    JOptionPane.showMessageDialog(null,"Decrypted Data
                    "+"\n"+decryptedmessage);
                }
                catch(Exception e)
                {
                    System.out.println(e);
                }
            }
    void generatesymmetrickey()
    {
         try
        {
                Random r = new Random();
                int num=r.nextInt(10000);
                String knum=String.valueOf(num);
                byte[] knumb=knum.getBytes();
                skey=getRawKey(knumb);
                skeystring=new String(skey);
                System.out.println("DES
                SymmerticKey="+skeystring);
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
    }

    private static byte[] getRawKey(byte[] seed) throws Exception
    {
        KeyGenerator kgen=KeyGenerator.getInstance("DES ");
        SecureRandom sr =SecureRandom.getInstance("SHA1PRNG");
        sr.setSeed(seed);
        kgen.init(56,sr);
        SecretKey skey=kgen.generateKey();
        raw=skey.getEncoded();
        return raw;
    }

    private static byte[] encrypt(byte[] raw,byte[] clear) throws Exception
    {
        SecretKey seckey = new SecretKeySpec(raw, "DES");
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.ENCRYPT_MODE,seckey);
        byte[] encrypted=cipher.doFinal(clear);
        return encrypted;
    }

    private static byte[] decrypt(byte[] raw,byte[] encrypted) throws Exception
    {
        SecretKey seckey = new SecretKeySpec(raw, "DES");
```

```
		Cipher cipher = Cipher.getInstance("DES");
		cipher.init(Cipher.DECRYPT_MODE,seckey);
		byte[] decrypted = cipher.doFinal(encrypted);
		return decrypted;
	}
	public static void main(String args[])
	{
		DES des=new DES();
	}

}
```

OUTPUT:



RESULT:

Thus the java program for applying Data Encryption Standard (DES) Algorithm for a practical application of User Message Encryption is written and executed successfully.

Ex.No. : 1(b) **AES ALGORITHM**

Date :

AIM:

To apply Advanced Encryption Standard (AES) Algorithm for a practical application like URL Encryption.

ALGORITHM:

1. AES is based on a design principle known as a substitution–permutation.
2. AES does not use a Feistel network like DES, it uses variant of Rijndael.
3. It has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits.
4. AES operates on a 4 × 4 column- major order array of bytes, termed the state

PROGRAM:

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AES
{
   private static SecretKeySpec secretKey;
   private static byte[] key;
   public static void setKey(String myKey) {
      MessageDigest sha = null;
      try {
         key = myKey.getBytes("UTF-8");
         sha = MessageDigest.getInstance("SHA-1");
         key = sha.digest(key);
         key = Arrays.copyOf(key, 16);
         secretKey = new SecretKeySpec(key, "AES");
      } catch (NoSuchAlgorithmException e) {
         e.printStackTrace();
      } catch (UnsupportedEncodingException e) {
         e.printStackTrace();
      }
   }
   public static String encrypt(String strToEncrypt, String secret) {
      try {
         setKey(secret);
         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
         cipher.init(Cipher.ENCRYPT_MODE, secretKey);
         return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes
("UTF-8")));
      } catch (Exception e) {
         System.out.println("Error while encrypting: " + e.toString());
      }
```

```
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret) {
        try {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        } catch (Exception e) {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }

    public static void main(String[] args) {

            System.out.println("Enter the secret key: ");
        String secretKey = System.console().readLine();

            System.out.println("Enter the original URL: ");
        String originalString = System.console().readLine();

        String encryptedString = AES.encrypt(originalString, secretKey);
        String decryptedString = AES.decrypt(encryptedString, secretKey);

        System.out.println("URL Encryption Using AES Algorithm\n -----------");
        System.out.println("Original URL : " + originalString);
        System.out.println("Encrypted URL : " + encryptedString);
        System.out.println("Decrypted URL : " + decryptedString);
    }
}
```

OUTPUT:

```
C:\Security Lab New\programs>java AES
Enter the secret key:
annaUniversity
Enter the original URL:
www.annauniv.edu
URL Encryption Using AES Algorithm
..........................
Original URL : www.annauniv.edu
Encrypted URL : vibpFJW6Cvs5Y+L7t4N6YWWe07+JzS1d3CU2h3mEvEg=
Decrypted URL : www.annauniv.edu
```

RESULT:

Thus the java program for applying Advanced Encryption Standard (AES) Algorithm for a practical application of URL encryption is written and executed successfully.

Ex.No. : 2(a) **RSA ALGORITHM**
Date :

AIM:

To implement a RSA algorithm using HTML and Javascript.

ALGORITHM:

1. Choose two prime number p and q.
2. Compute the value of n and t.
3. Find the value of public key e.
4. Compute the value of private key d.
5. Do the encryption and decryption
    a. Encryption is given as,
       $c = t^e \bmod n$
    b. Decryption is given as,
       $t = c^d \bmod n$

PROGRAM:

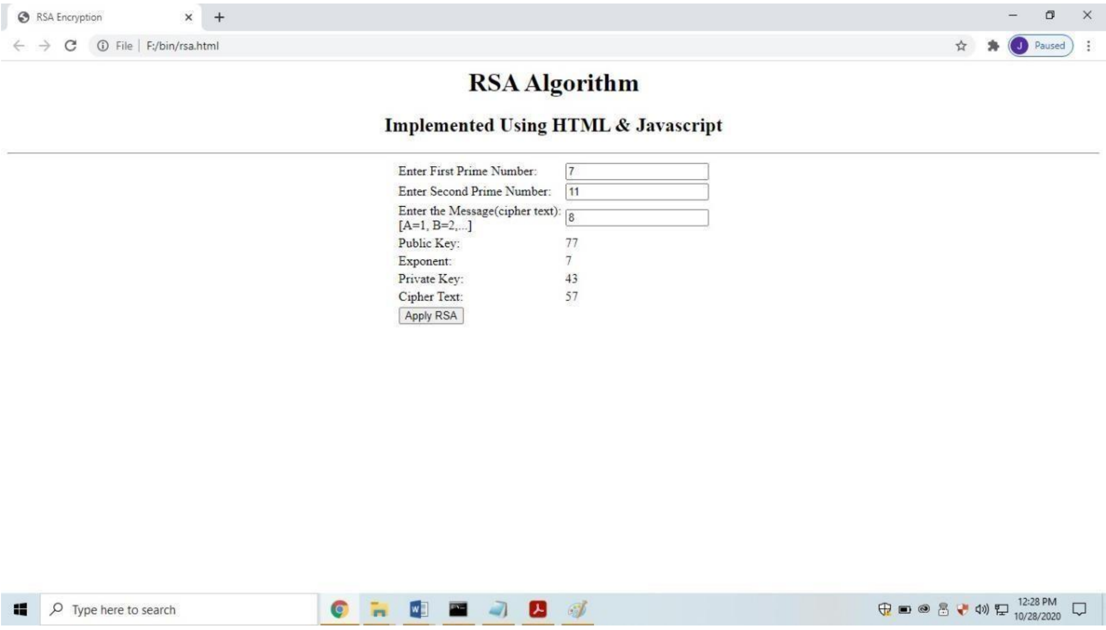
rsa.html

```
<html>
<head>
   <title>RSA Encryption</title>
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
   <center>
      <h1>RSA Algorithm</h1>
      <h2>Implemented Using HTML & Javascript</h2>
      <hr>
      <table>
         <tr>
            <td>Enter First Prime Number:</td>
            <td><input type="number" value="53" id="p"></td>
         </tr>
         <tr>
            <td>Enter Second Prime Number:</td>
            <td><input type="number" value="59" id="q"></p> </td>
         </tr>
         <tr>
            <td>Enter the Message(cipher text):<br>[A=1, B=2,...]</td>
            <td><input type="number" value="89" id="msg"></p> </td>
         </tr>
         <tr>
            <td>Public Key:</td>
            <td><p id="publickey"></p> </td>
         </tr>
         <tr>
            <td>Exponent:</td>
            <td><p id="exponent"></p> </td>
         </tr>
```

```
            <tr>
                <td>Private Key:</td>
                <td><p id="privatekey"></p></td>
            </tr>
            <tr>
                <td>Cipher Text:</td>
                <td><p id="ciphertext"></p> </td>
            </tr>
            <tr>
                <td><button onclick="RSA();">Apply RSA</button></td>
            </tr>
        </table> </center>
</body>
<script type="text/javascript">

function RSA()
{
        var gcd, p, q, no, n, t, e, i, x;
        gcd = function (a, b) { return (!b) ? a : gcd(b, a % b); };
        p = document.getElementById('p').value;
        q = document.getElementById('q').value;
        no = document.getElementById('msg').value;
        n = p * q;
        t = (p - 1) * (q - 1);
        for (e = 2; e < t; e++)
        {
            if (gcd(e, t) == 1)
            {
                break;
            }
        }
        for (i = 0; i < 10; i++)
        {
            x = 1 + i * t
            if (x % e == 0)
            {
                d = x / e;
                break;
            }
        }
        ctt = Math.pow(no, e).toFixed(0);
        ct = ctt % n;
        dtt = Math.pow(ct, d).toFixed(0);
        dt = dtt % n;
        document.getElementById('publickey').innerHTML = n;
        document.getElementById('exponent').innerHTML = e;
        document.getElementById('privatekey').innerHTML = d;
        document.getElementById('ciphertext').innerHTML = ct;
    }
</script>
```

</html>

OUTPUT:



RESULT:

Thus the RSA algorithm was implemented using HTML and Javascript and executed successfully.

Ex.No. : 2(b) DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM
Date :

AIM:

To implement a Diffie-Hellman Key Exchange algorithm.

ALGORITHM:

1. Sender and receiver publicly agree to use a modulus *p* and base *g* which is a primitive root modulo p.
2. Sender chooses a secret integer x then sends Bob R1 = g^x mod *p*
3. Receiver chooses a secret integer y, then sends Alice R2 = g^y mod *p*
4. Sender computes k1 = B^x mod *p*
5. Receiver computes k2 = A^y mod *p*
6. Sender and Receiver now share a secret key.

PROGRAM:

```
import java.io.*;
import java.math.BigInteger;
class dh
{
public static void main(String[]args)throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter prime number:");
BigInteger p=new BigInteger(br.readLine());

System.out.print("Enter primitive root of "+p+":");
BigInteger g=new BigInteger(br.readLine());

System.out.println("Enter value for x less than "+p+":");
BigInteger  x=new BigInteger(br.readLine());
BigInteger R1=g.modPow(x,p);
System.out.println("R1="+R1);

System.out.print("Enter value for y less than "+p+":");
BigInteger y=new BigInteger(br.readLine());
BigInteger R2=g.modPow(y,p);
System.out.println("R2="+R2);

BigInteger k1=R2.modPow(x,p);
System.out.println("Key calculated at Sender's side:"+k1);
BigInteger k2=R1.modPow(y,p);
System.out.println("Key calculated at Receiver's side:"+k2);
System.out.println("Diffie-Hellman secret key was calculated.");
}
}
```

OUTPUT

```
C:\Security Lab New\programs>javac dh.java

C:\Security Lab New\programs>java dh
Enter prime number:
11
Enter primitive root of 11:7
Enter value for x less than 11:
3
R1=2
Enter value for y less than 11:6
R2=4
Key calculated at Sender's side:9
Key calculated at Receiver's side:9
Diffie-Hellman secret key was calculated.
```

RESULT:

Thus the Diffie-Hellman key exchange algorithm was implemented and executed successfully.

Ex.No. : 3 **DIGITAL SIGNATURE SCHEME**
Date :

AIM:

To implement the signature scheme - Digital Signature Standard.

ALGORITHM:

1. Declare the class and required variables.
2. Create the object for the class in the main program.
3. Access the member functions using the objects.
4. Implement the SIGNATURE SCHEME - Digital Signature Standard.
5. It uses a hash function.
6. The hash code is provided as input to a signature function along with a random number K generated for the particular signature.
7. The signature function also depends on the sender„s private key.
8. The signature consists of two components.
9. The hash code of the incoming message is generated.
10. The hash code and signature are given as input to a verification function.

PROGRAM:

```
import java.util.*;
import java.math.BigInteger;
class dsaAlg {
final static BigInteger one = new BigInteger("1");
final static BigInteger zero = new BigInteger("0");
public static BigInteger getNextPrime(String ans)
{
BigInteger test = new BigInteger(ans);
while (!test.isProbablePrime(99))
e:
{
test = test.add(one);
}
return test;
}
public static BigInteger findQ(BigInteger n)
{
BigInteger start = new BigInteger("2");
while (!n.isProbablePrime(99))
{
while (!((n.mod(start)).equals(zero)))
{
start = start.add(one);
}
n = n.divide(start);
}
return n;
}
```

```
public static BigInteger getGen(BigInteger p, BigInteger q,
Random r)
{
BigInteger h = new BigInteger(p.bitLength(), r);
h = h.mod(p);
return h.modPow((p.subtract(one)).divide(q), p);
}
public static void main (String[] args) throws
java.lang.Exception
{
Random randObj = new Random();
BigInteger p = getNextPrime("10600"); /* approximate
prime */
BigInteger q = findQ(p.subtract(one));
BigInteger g = getGen(p,q,randObj);
System.out.println(" \n simulation of Digital Signature Algorithm \n");
System.out.println(" \n global public key components are:\n");
System.out.println("\np is: " + p);
System.out.println("\nq is: " + q);
System.out.println("\ng is: " + g);
BigInteger x = new BigInteger(q.bitLength(), randObj);
x = x.mod(q);
BigInteger y= g.modPow(x,p);
BigInteger k = new BigInteger(q.bitLength(), randObj);
k = k.mod(q);
BigInteger r = (g.modPow(k,p)).mod(q);
BigInteger hashVal = new BigInteger(p.bitLength(),
randObj);
BigInteger kInv = k.modInverse(q);
BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));
s = s.mod(q);
System.out.println("\nsecret information are:\n");
System.out.println("x (private) is:" + x);
System.out.println("k (secret) is: " + k);
System.out.println("y (public) is: " + y);
System.out.println("h (rndhash) is: " + hashVal);
System.out.println("\n generating digital signature:\n");
System.out.println("r is : " + r);
System.out.println("s is : " + s);
BigInteger w = s.modInverse(q);
BigInteger u1 = (hashVal.multiply(w)).mod(q);
BigInteger u2 = (r.multiply(w)).mod(q);
BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));
v = (v.mod(p)).mod(q);
System.out.println("\nverifying digital signature (checkpoints)\n:");
System.out.println("w is : " + w);
System.out.println("u1 is : " + u1);
System.out.println("u2 is : " + u2);
System.out.println("v is : " + v);
if (v.equals(r))
```

```
{
System.out.println("\nsuccess: digital signature is verified!\n " + r);
}
else
{
System.out.println("\n error: incorrect digital signature\n ");
}
}
}
```

OUTPUT:

C:\Security Lab
New\programs>javac dsaAlg.java
C:\Security Lab New\programs>java
dsaAlg simulation of Digital
Signature Algorithm
global public key components are:
p is: 10601
q is: 53
g is: 6089
secret information are:
x
(
p
ri
v
a
t
e
)
is
:
6
k
(
s
e
c
r
e
t)
i
s
:

3
y (public) is: 1356
h (rndhash) is:
12619
generating
digital
signature:
r

i
s

:

2

s

i
s

:

4
1
verifying digital signature (checkpoints):
w

i
s

:

2
2

u
1

i
s

:

4

u

2

i
s

:

4
4

v

i
s

:

2
success: digital
signature is verified!2

RESULT:

Thus the Digital Signature Standard Signature Scheme has been implemented andexecuted successfully.