```
import javax.swing.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.security.SecureRandom;
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 SymmetricKey=" + 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();
  }
}
```

AES.java

```
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);
  }
}
```

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>
```

DIFFIE-HELLMAN.java

```
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 lessthan "+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.");
 }
}
```

DSS.java

```
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 digitalsignature\n ");
    }
  }
}
```