DES:

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class DES {
    public static void main(String[] argv) {
        try {
            System.out.println("Message Encryption Using DES
Algorithm\n-------");
            KeyGenerator keygenerator =
KeyGenerator.getInstance("DES");
            SecretKey myDesKey = keygenerator.generateKey();
            Cipher desCipher;
            desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            desCipher.init(Cipher.ENCRYPT_MODE, myDesKey);
            byte[] text = "Secret Information ".getBytes();
            System.out.println("Message [Byte Format] : " + text);
            System.out.println("Message : " + new String(text));
            byte[] textEncrypted = desCipher.doFinal(text);
            System.out.println("Encrypted Message: " +
textEncrypted);
            desCipher.init(Cipher.DECRYPT_MODE, myDesKey);
            byte[] textDecrypted = desCipher.doFinal(textEncrypted);
            System.out.println("Decrypted Message: " + new
String(textDecrypted));
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        }
    }
}
```

AES:

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
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.getBy
tes("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) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the key ");
        final String secretKey = scan.nextLine();
        System.out.println("Enter the URL to encrypt");
        String originalString = scan.nextLine();
        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</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 />
      <form onsubmit="event.preventDefault()">
        <div>
          <p>
            Enter First Prime Number (p):<input
              type="number"
              placeholder=" Prime Number"
              id="p"
              required
            />
          </p>
          <p>
            Enter First Second Number (q):<input
              type="number"
              placeholder=" Prime Number"
              id="q"
              required
            />
          </p>
          <p>
            Enter the Message(Cipher text):[A=1, B=2,...]<input
              type="number"
              placeholder="message"
              id="msg"
            />
          </p>
          <input type="submit" id="apply" value="Apply RSA"
onclick="RSA()" />
        </div>
      </form>
      <table>
        <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>Decrypted text:</td>
          <td>
            <p id="decryptedtext"></p>
          </td>
        </tr>
      </table>
    </center>
  </body>
  <script type="text/javascript">
    function power(x, y, n) {
      let res = 1;
      while (y > 0) {
        if (y & 1) res = res * x;
        y = y >> 1;
        x = x * x;
        x = x % n;
      }
      return res;
    }
    function test_prime(n) {
      if (n == 1) {
        return false;
      } else if (n == 2) {
        return true;
      } else {
        for (var x = 2; x < n; x++) {
          if (n % x == 0) {
            return false;
          }
        }
        return true;
      }
    }
    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;
      if (test_prime(p) == false || test_prime(q) == false) {
        document.getElementById("publickey").innerHTML = null;
        document.getElementById("exponent").innerHTML = null;
        document.getElementById("privatekey").innerHTML = null;
        document.getElementById("ciphertext").innerHTML = null;
        document.getElementById("decryptedtext").innerHTML = null;
        alert("enter a prime no");
        return false;
      }
      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 = power(no, e, n);
      ct = ctt % n;
      dtt = power(ct, d, n);
      dt = dtt % n;
      document.getElementById("publickey").innerHTML = n;
      document.getElementById("exponent").innerHTML = e;
      document.getElementById("privatekey").innerHTML = d;
      document.getElementById("ciphertext").innerHTML = ct;
      document.getElementById("decryptedtext").innerHTML = dt;
    }
  </script>
</html>
```

Diffie:

```
import java.io.*;
import java.math.BigInteger;

class Diffie {
    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("deffie hellman secret key Encryption has
Taken");
    }
}
```

SHA1:

```
import java.security.*;
import java.io.*;

public class SHA1 {
    public static void main(String[] a) {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        try {
            MessageDigest md = MessageDigest.getInstance("SHA1");
            System.out.println("Enter the string: ");
            String input = in.readLine();
            md.update(input.getBytes());
            byte[] output = md.digest();
            System.out.println();
            System.out.println("SHA1(\"" + input + "\") = " +
bytesToHex(output));
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }

    public static String bytesToHex(byte[] b) {
        // char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6',
'7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
        StringBuffer buf = new StringBuffer();
        // for (int j = 0; j < b.length; j++) {
        //     buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
        //     buf.append(hexDigit[b[j] & 0x0f]);
        // }
        for(byte by : b){
            buf.append(String.format("%X", by));
        }

        return buf.toString();
    }
}
```

DSS:

```
import java.util.*;
import java.math.BigInteger;

public class DSS {
    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)) {
            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("Digital Signature Algorithm");
        System.out.println("global public key components are:");
        System.out.println("p is: " + p);
        System.out.println("q is: " + q);
        System.out.println("g 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("secret information are:");
        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("Generating digital signature:");
        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("verifying digital signature
(checkpoints):");
        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("success: digital signature is
verified! " + r);
        } else {
            System.out.println("error: incorrect digital
signature");
        }
    }
}
```