**EXP-1**

import java.util.\*;

import javax.crypto.BadPaddingException;

import javax.crypto.Cipher;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.DESKeySpec;

import java.io.\*;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.spec.InvalidKeySpecException;

class DES{

public static void main(String[] args) throws IOException, NoSuchAlgorithmException, InvalidKeyException, InvalidKeySpecException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {

String message="DES";

byte[] myMessage =message.getBytes();

KeyGenerator Mygenerator = KeyGenerator.getInstance("DES");

SecretKey myDesKey = Mygenerator.generateKey();

Cipher myCipher = Cipher.getInstance("DES");

myCipher.init(Cipher.ENCRYPT\_MODE, myDesKey);

byte[] myEncryptedBytes=myCipher.doFinal(myMessage);

myCipher.init(Cipher.DECRYPT\_MODE, myDesKey);

byte[] myDecryptedBytes=myCipher.doFinal(myEncryptedBytes);

String encrypteddata=new String(myEncryptedBytes);

String decrypteddata=new String(myDecryptedBytes);

System.out.println("Message : "+ message);

System.out.println("Encrypted - "+ encrypteddata);

System.out.println("Decrypted Message - "+ decrypteddata);

}

}

**EXP-2**

import java.util.\*;  
public class SubstitutionCipher  
{  
static Scanner sc=new Scanner(System.in);  
static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));  
public static void main(String args[])throws IOException  
{  
String a="abcdefghijklmnopqrstuvwxyz";  
String b="zyxwvutsrqponmlkjihgfedcba";  
System.out.println("Enter any string:");  
String str=br.readLine();  
String decrypt="";  
char c;  
for(int i=0;i<str.length();i++)  
{  
c=str.charAt(i);  
int j=a.indexOf(c);  
decrypt=decrypt+b.charAt(j);  
}  
System.out.println("The encrypted data is :"+decrypt);  
}  
}

**EXP-3**

**Caesar cipher**

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.BufferedReader;

import java.util.Scanner;

public class CaesarCipher1

{

static Scanner sc=new Scanner(System.in);

static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

public static void main(String[]args)throws IOException

{

System.out.println("enter any string:");

String str=br.readLine();

System.out.println("enter the key:");

int key=sc.nextInt();

String encrypted=encrypt(str,key);

System.out.println("Encrypted String is:"+encrypted);

String decrypted=decrypt(encrypted,key);

System.out.println("Decrypted String is:"+decrypted);

System.out.print("\n");

}

public static String encrypt(String str,int key)

{

String encrypted="";

for(int i=0;i<str.length();i++)

{

int c=str.charAt(i);

if(Character.isUpperCase(c))

{

c=c+(key%26);

if(c>'Z')

c=c-26;

}

else if(Character.isLowerCase(c))

{

c=c+(key%26);

if(c>'z')

c=c-26;

}

encrypted+=(char)c;

}

return encrypted;

}

public static String decrypt(String str,int key)

{

String decrypted="";

for(int i=0;i<str.length();i++)

{

int c=str.charAt(i);

if(Character.isUpperCase(c))

{

c=c-(key%26);

if(c<'A')

c=c+26;

}

else if(Character.isLowerCase(c))

{

c=c-(key%26);

if(c<'a')

c=c+26;

}

decrypted+=(char)c;

}

return decrypted;

} }

**Rjindael algorithm**

import java.security.\*;

import javax.crypto.\*;

import javax.crypto.spec.\*;

import java.io.\*;

public class AES {

public static String asHex (byte buf[]) {

StringBuffer strbuf = new StringBuffer(buf.length \* 2);

int i;

for (i = 0; i < buf.length; i++) {

if (((int) buf[i] & 0xff) < 0x10)

strbuf.append("0");

strbuf.append(Long.toString((int) buf[i] & 0xff, 16)); }

return strbuf.toString(); }

public static void main(String[] args) throws Exception {

String message="AES";

KeyGenerator kgen = KeyGenerator.getInstance("AES");

kgen.init(128);

SecretKey skey = kgen.generateKey();

byte[] raw = skey.getEncoded();

SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

Cipher = Cipher.getInstance("AES");

cipher.init(Cipher.ENCRYPT\_MODE, skeySpec);

byte[] encrypted = cipher.doFinal((args.length == 0 ? message :args[0]).getBytes());

System.out.println("encrypted string: " + asHex(encrypted));

cipher.init(Cipher.DECRYPT\_MODE, skeySpec);

byte[] original = cipher.doFinal(encrypted);

String originalString = new String(original);

System.out.println("Original string: " + originalString + " " + asHex(original));

}

}

**Exp-4**

**HILL CIPHER**

import java.io.\*;

import java.util.\*;

import java.io.\*;

public class HillCipher {

static float[][] decrypt = new float[3][1];

static float[][] a = new float[3][3];

static float[][] b = new float[3][3];

static float[][] mes = new float[3][1];

static float[][] res = new float[3][1];

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

static Scanner sc = new Scanner(System.in);

public static void main(String[] args) throws IOException {

getkeymes();

for(int i=0;i<3;i++)

for(int j=0;j<1;j++)

for(int k=0;k<3;k++) {

res[i][j]=res[i][j]+a[i][k]\*mes[k][j]; }

System.out.print("\nEncrypted string is :");

for(int i=0;i<3;i++) {

System.out.print((char)(res[i][0]%26+97));

res[i][0]=res[i][0];}

inverse();

for(int i=0;i<3;i++)

for(int j=0;j<1;j++)

for(int k=0;k<3;k++) {

decrypt[i][j] = decrypt[i][j]+b[i][k]\*res[k][j]; }

System.out.print("\nDecrypted string is : ");

for(int i=0;i<3;i++){

System.out.print((char)(decrypt[i][0]%26+97));}

System.out.print("\n");}

public static void getkeymes() throws IOException {

System.out.println("Enter 3x3 matrix for key (It should be inversible): ");

for(int i=0;i<3;i++)

for(int j=0;j<3;j++)

a[i][j] = sc.nextInt();

System.out.print("\nEnter a 3 letter string: ");

String msg = br.readLine();

for(int i=0;i<3;i++)

mes[i][0] = msg.charAt(i)-97;}

public static void inverse() {

float p,q;

float[][] c = a;

for(int i=0;i<3;i++)

for(int j=0;j<3;j++) {

//a[i][j]=sc.nextFloat();

if(i==j)

b[i][j]=1;

else b[i][j]=0;}

for(int k=0;k<3;k++) {

for(int i=0;i<3;i++) {

p = c[i][k];

q = c[k][k];

for(int j=0;j<3;j++) {

if(i!=k) {

c[i][j] = c[i][j]\*q-p\*c[k][j];

b[i][j] = b[i][j]\*q-p\*b[k][j];} } } }

for(int i=0;i<3;i++)

for(int j=0;j<3;j++) {

b[i][j] = b[i][j]/c[i][i]; }

System.out.println("");

System.out.println("\nInverse Matrix is : ");

for(int i=0;i<3;i++) {

for(int j=0;j<3;j++)

System.out.print(b[i][j] + " ");

System.out.print("\n");

}}

}

**Exp-5**

**A)BLOWFISH**

import java.io.\*;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.security.Key;

import javax.crypto.Cipher;

import javax.crypto.CipherOutputStream;

import javax.crypto.KeyGenerator;

import java.util.Base64;

public class BlowFish {

public static void main(String[] args) throws Exception

{

KeyGeneratorkeyGenerator = KeyGenerator.getInstance("Blowfish");

keyGenerator.init(128);

Key secretKey = keyGenerator.generateKey();

Cipher cipherOut = Cipher.getInstance("Blowfish/CFB/NoPadding");

cipherOut.init(Cipher.ENCRYPT\_MODE, secretKey);

BASE64Encoder encoder = new BASE64Encoder();

byte iv[] = cipherOut.getIV();

if (iv != null) {

System.out.println("Initialization Vector of the Cipher: " + encoder.encode(iv));

} FileInputStream fin = new FileInputStream("inputFile.txt");

FileOutputStreamfout = new FileOutputStream("outputFile.txt");

CipherOutputStreamcout = new CipherOutputStream(fout, cipherOut);

int input = 0;

while ((input = fin.read()) != -1) {

fin.close();

cout.close();

}

}

**B)RSA**

import java.math.\*;

import java.util.\*;

class RSA {

public static void main(String args[]){

int p, q, n, z, d = 0, e, i;

int msg = 12;

double c;

BigInteger msgback;

p = 3;

q = 11;

n = p \* q;

z = (p - 1) \* (q - 1);

System.out.println("the value of z = " + z);

for (e = 2; e < z; e++) {

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);

}

}

**EXP-6**

**“Hello world using Blow fish”**

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.swing.JOptionPane;

public class BlowFishCipher {

public static void main(String[] args) throws Exception

{

KeyGenerator = KeyGenerator.getInstance("Blowfish");

SecretKey secretkey=keygenerator.generateKey();

Cipher = Cipher.getInstance("Blowfish");

cipher.init(Cipher.ENCRYPT\_MODE, secretkey);

String inputText = JOptionPane.showInputDialog("Input your message: ");

byte[] encrypted = cipher.doFinal(inputText.getBytes());

cipher.init(Cipher.DECRYPT\_MODE, secretkey);

byte[] decrypted = cipher.doFinal(encrypted);

JOptionPane.showMessageDialog(JOptionPane.getRootFrame(), "\nEncrypted text: " + new String(encrypted) + "\n" + "\nDecrypted text: " +new String(decrypted));

System.exit(0);

}

}

**EXP-7**

**DIFFIE HELLMAN**

import java.math.BigInteger;

import java.security.KeyFactory;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.SecureRandom; import

javax.crypto.spec.DHParameterSpec; import

javax.crypto.spec.DHPublicKeySpec; public

class DiffeHellman

{

public final static int

pValue = 47;

public final static int gValue = 71;

public final static int XaValue = 9;

public final static int XbValue = 14;

public static void main(String[] args) throws Exception

{

BigInteger p = new BigInteger(Integer.toString(pValue));

BigInteger g = new BigInteger(Integer.toString(gValue));

BigIntegerXa = new BigInteger(Integer.toString(XaValue));

BigIntegerXb = new BigInteger(Integer.toString(XbValue));

createKey();

intbitLength = 512; // 512 bits

SecureRandomrnd = new SecureRandom();

p = BigInteger.probablePrime(bitLength, rnd);

g = BigInteger.probablePrime(bitLength, rnd);

createSpecificKey(p, g);

}

public static void createKey() throws Exception

{

KeyPairGeneratorkpg = KeyPairGenerator.getInstance("DiffieHellman");

kpg.initialize(512);

KeyPairkp = kpg.generateKeyPair();

KeyFactorykfactory = KeyFactory.getInstance("DiffieHellman");

DHPublicKeySpeckspec = (DHPublicKeySpec) kfactory.getKeySpec(kp.getPublic(), DHPublicKeySpec.class);

System.out.println("Public key is: " +kspec); }

public static void createSpecificKey(BigInteger p, BigInteger g) throws Exception

{

KeyPairGeneratorkpg = KeyPairGenerator.getInstance("DiffieHellman");

DHParameterSpecparam = new DHParameterSpec(p, g);

kpg.initialize(param);

KeyPairkp = kpg.generateKeyPair();

KeyFactorykfactory = KeyFactory.getInstance("DiffieHellman");

DHPublicKeySpeckspec = (DHPublicKeySpec) kfactory.getKeySpec(kp.getPublic(),

DHPublicKeySpec.class);

System.out.println("\nPublic key is : " +kspec);

} }

**EXP-8**

**SHA-1**

import java.security.\*;

public class SHA1

{

public static void main(String[] a)

{

try

{

MessageDigest md = MessageDigest.getInstance("SHA1");

System.out.println("Message digest object info: ");

System.out.println(" Algorithm = " +md.getAlgorithm());

System.out.println(" Provider = " +md.getProvider());

System.out.println(" ToString = " +md.toString());

String input = "";

md.update(input.getBytes());

byte[] output = md.digest();

System.out.println();

System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));

input = "abc";

md.update(input.getBytes());

output = md.digest();

System.out.println();

System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));

input = "abcdefghijklmnopqrstuvwxyz";

md.update(input.getBytes());

output = md.digest();

System.out.println();

System.out.println("SHA1(\"" +input+"\") = " +bytesToHex(output));

System.out.println(""); }

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'};

StringBufferbuf = new StringBuffer();

for (int j=0; j<b.length; j++)

{

buf.append(hexDigit[(b[j] >> 4) & 0x0f]);

buf.append(hexDigit[b[j] & 0x0f]); }

returnbuf.toString(); }

}