Sub cipher

import java.io.\*;

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);

}

}

}

}

Trans cipher

import java.util.\*;

import java.io.\*;

class TranspositionCipher

{

public static void main(String [] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("enter a string");

String s=sc.next();

String a="",b="";

int c;

for(c=0;c<s.length();c++)

{

if(c%2==0)

{

a=a+s.charAt(c);

}

else

{

b=b+s.charAt(c);

}

}

System.out.println(a+b);

}

}

Md5

import java.security.\*;

public class MD5

 {

public static void main(String[] a)

{

try

{

MessageDigest md = MessageDigest.getInstance("MD5");

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("MD5(\""+input+"\") = " +bytesToHex(output));

input = "abc";

md.update(input.getBytes());

output = md.digest();

System.out.println();

System.out.println("MD5(\""+input+"\") = "  +bytesToHex(output));

input = "abcdefghijklmnopqrstuvwxyz";

md.update(input.getBytes());

output = md.digest();

System.out.println();

System.out.println("MD5(\"" +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'};

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]);

 }

return buf.toString();

 }

 }

Deffehellman

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));

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

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

createKey();

int bitLength = 512;

SecureRandom rnd = new SecureRandom();

p = BigInteger.probablePrime(bitLength, rnd);

g = BigInteger.probablePrime(bitLength, rnd);

createSpecificKey(p, g);

}

public static void createKey() throws Exception

{

KeyPairGenerator kpg = KeyPairGenerator.getInstance("DiffieHellman");

kpg.initialize(512);

KeyPair kp = kpg.generateKeyPair();

KeyFactory kfactory = KeyFactory.getInstance("DiffieHellman");

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

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

}

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

 {

KeyPairGenerator kpg = KeyPairGenerator.getInstance("DiffieHellman");

DHParameterSpec param = new DHParameterSpec(p, g);

kpg.initialize(param);

KeyPair kp = kpg.generateKeyPair();

KeyFactory kfactory = KeyFactory.getInstance("DiffieHellman");

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

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

}

 }

DES

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,encryptedData,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 Symmetric key = "+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 {

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

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

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

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

return encrypted;

}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {

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

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

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

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

return decrypted;

}

public static void main(String args[]) {

DES des = new DES();

}

}

DigSign

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.Signature;

import java.util.Scanner;

public class DigSign{

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

      Scanner sc = new Scanner(System.in);

System.out.println("Enter some text");

      String msg = sc.nextLine();

KeyPairGenerator KeyPairGen = KeyPairGenerator.getInstance("DSA");

KeyPairGen.initialize(2048);

KeyPair pair = KeyPairGen.generateKeyPair();

PrivateKey privKey = pair.getPrivate();

      Signature sign = Signature.getInstance("SHA256withDSA");

sign.initSign(privKey);

byte[] bytes = "msg".getBytes();

sign.update(bytes);

byte[] signature = sign.sign();

System.out.println("Digital signature for given text: "+new String(signature, "UTF8"));

   }

}

RSA1

import java.util.\*;

import java.math.\*;

class RSA1

{

    public static void main(String args[])

    {

        Scanner sc=new Scanner(System.in);

        int p,q,n,z,d=0,e,i;

        System.out.println("Enter the number to be encrypted and decrypted");

        int msg=sc.nextInt();

        double c;

        BigInteger msgback;

        System.out.println("Enter 1st prime number p");

        p=sc.nextInt();

        System.out.println("Enter 2nd prime number q");

        q=sc.nextInt();

        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)            // e is for public key exponent

            {

                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 is for private key exponent

            {

                d=x/e;

                break;

            }

        }

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

        c=(Math.pow(msg,e))%n;

        System.out.println("Encrypted message is : -");

        System.out.println(c);

                //converting int value of n to BigInteger

        BigInteger N = BigInteger.valueOf(n);

        //converting float value of c to BigInteger

        BigInteger C = BigDecimal.valueOf(c).toBigInteger();

        msgback = (C.pow(d)).mod(N);

        System.out.println("Derypted message is : -");

        System.out.println(msgback);

    }

    static int gcd(int e, int z)

    {

        if(e==0)

            return z;

        else

            return gcd(z%e,e);

    }

}