**Shift chipher**

import java.io.\*;

import java.util.\*;

class ShiftCipher

{

public static StringBuffer encrypt(String text, int s)

{

StringBuffer result= new StringBuffer();

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

{

if(Character.isUpperCase(text.charAt(i)))

{

char ch=(char)(((int)text.charAt(i)+s-65)%26+65);

result.append(ch);

}

else

{

char ch=(char)(((int)text.charAt(i)+s-97)%26+97);

result.append(ch);

}

}

return result;

}

public static void main(String[] args)

{

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

System.out.println("Enter Text:");

Scanner sc =new Scanner(System.in);

String text=sc.nextLine();

System.out.println("Enter shift:");

int s=sc.nextInt();

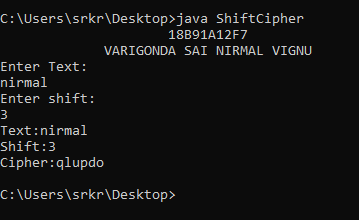
System.out.println("Text:"+text);

System.out.println("Shift:"+s);

System.out.println("Cipher:"+encrypt(text,s));

}

}



**MonoCIPHER**

import java.io.\*;

import java.util.\*;

class MonoCipher

{

static String encoder(char[] key)

{

String encoded="";

boolean[]arr=new boolean[26];

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

{

if(arr[key[i]-65]==false)

{

encoded+=(char)key[i];

arr[key[i]-65]=true;

}

}

for(int i=0;i<26;i++)

{

if(arr[i]==false)

{

arr[i]=true;

encoded+=(char)(i+65);

}

}

return encoded;

}

static String cipheredIt(String msg,String encoded)

{

String cipher="";

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

{

if(msg.charAt(i)>='A' && msg.charAt(i)<='Z')

{

int pos=msg.charAt(i)-65;

cipher+=encoded.charAt(pos);

}

else

{

cipher+=msg.charAt(i);

}

}

return cipher;

}

public static void main(String[] args)

{

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

String key;

key="KRYPTOS";

System.out.println("keyword:"+key);

String encoded=encoder(key.toCharArray());

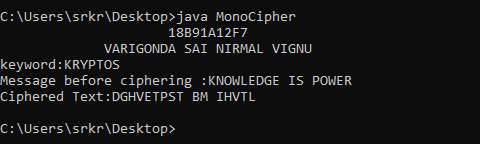
String message="KNOWLEDGE IS POWER";

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

System.out.println("Ciphered Text:"+cipheredIt(message,encoded));

}

}



OTPCIPHER

import java.util.\*;

import java.io.\*;

public class OTPCipher

{

public static void main(String[] args)

{

Scanner sc =new Scanner(System.in);

String text="HELLO";

String key=RandomAlpha(text.length());

String enc=OTPEncryption(text,key);

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

System.out.println("plaintext:"+text);

System.out.println("Encrypted:"+enc);

}

public static String RandomAlpha(int len)

{

Random r=new Random();

String key="XMCKL";

for(int x=0;x<len;x++)

key=key+(char)(r.nextInt(26) + 'A');

return key;

}

public static String OTPEncryption(String text,String key)

{

String alphaU = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int len=text.length();

String sb="";

for(int x=0;x<len;x++)

{

char get =text.charAt(x);

char keyget = key.charAt(x);

if(Character.isUpperCase(get))

{

int index=alphaU.indexOf(get);

int keydex=alphaU.indexOf(Character.toUpperCase(keyget));

int total=(index + keydex) % 26;

sb = sb+ alphaU.charAt(total);

}

else

{

sb=sb+get;

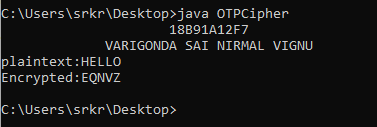
}

}

return sb;

}

}



MacSample

import java.security.Key;

import java.security.SecureRandom;

import javax.crypto.KeyGenerator;

import javax.crypto.Mac;

public class MacSample{

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

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

SecureRandom secRandom = new SecureRandom();

keyGen.init(secRandom);

Key key = keyGen.generateKey();

Mac mac = Mac.getInstance("HmacSHA256");

mac.init(key);

String msg = new String("am i good at cns");

byte[] bytes = msg.getBytes();

byte[] macResult = mac.doFinal(bytes);

System.out.println(" 18B91A12F7");

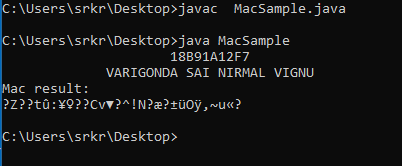
System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

System.out.println("Mac result:");

System.out.println(new String(macResult));

}

}



**Digitalsignature**

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.Signature;

import java.util.Scanner;

public class CreatingDigitalSignature {

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

Scanner sc = new Scanner(System.in);

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

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();

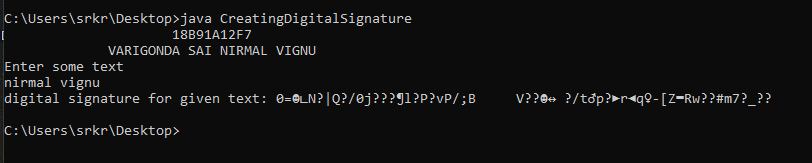
sign.update(bytes);

byte[] signature = sign.sign();

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

}

}



**HASH FUNCTION**

import java.math.BigInteger;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

class Hash{

public static byte[] getSHA(String input) throws NoSuchAlgorithmException

{

MessageDigest md = MessageDigest.getInstance("SHA-256");

return md.digest(input.getBytes(StandardCharsets.UTF\_8));

}

public static String toHexString(byte[] hash)

{

BigInteger number = new BigInteger(1, hash);

StringBuilder hexString = new StringBuilder(number.toString(16));

while (hexString.length() < 32)

{

hexString.insert(0, '0');

}

return hexString.toString();

}

public static void main(String args[])

{

try

{System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

System.out.println("HashCode Generated by SHA-256 for:");

String s2 = "hello world";

System.out.println("\n" + s2 + " : " + toHexString(getSHA(s2)));

}

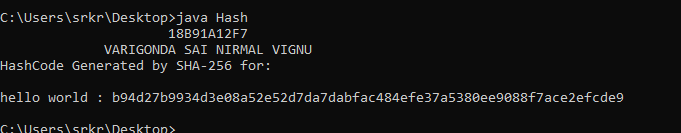
catch (NoSuchAlgorithmException e) {

System.out.println("Exception thrown for incorrect algorithm: " + e);

}

}

}



**RSA**

import java.util.\*;

import java.math.\*;

class RSA

{

public static void main(String args[])

{

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

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)

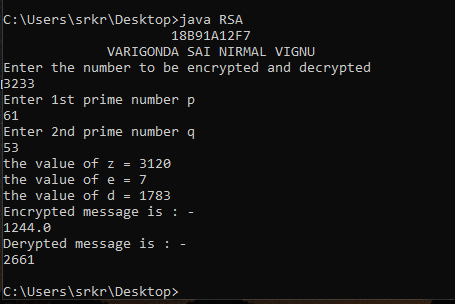
return z;

else

return gcd(z%e,e);

}

}



Diffie-Hellman

class DiffieHellman{

private static long power(long a, long b, long p)

{

if (b == 1)

return a;

else

return (((long)Math.pow(a, b)) % p);

}

public static void main(String[] args)

{

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

long P, G, x, a, y, b, ka, kb;

P = 23;

System.out.println("The value of P:" + P);

G = 9;

System.out.println("The value of G:" + G);

a = 4;

System.out.println("The private key a for Alice:" + a);

x = power(G, a, P);

b = 3;

System.out.println("The private key b for Bob:" + b);

y = power(G, b, P);

ka = power(y, a, P);

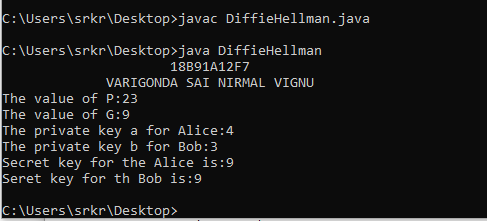
kb = power(x, b, P);

System.out.println("Secret key for the Alice is:" + ka);

System.out.println("Seret key for th Bob is:"+kb);

}

}



AES

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(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

final String secretKey="welcome";

String originalString="helloworld";

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

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

System.out.println(originalString);

System.out.println(encryptedString);

System.out.println(decryptedString);

}

}

Text

Description automatically generated

DES Algo

class DesAlgo {

private static class DES {

// CONSTANTS

// Initial Permutation Table

int[] IP = { 58, 50, 42, 34, 26, 18,

10, 2, 60, 52, 44, 36, 28, 20,

12, 4, 62, 54, 46, 38,

30, 22, 14, 6, 64, 56,

48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17,

9, 1, 59, 51, 43, 35, 27,

19, 11, 3, 61, 53, 45,

37, 29, 21, 13, 5, 63, 55,

47, 39, 31, 23, 15, 7 };

// Inverse Initial Permutation Table

int[] IP1 = { 40, 8, 48, 16, 56, 24, 64,

32, 39, 7, 47, 15, 55,

23, 63, 31, 38, 6, 46,

14, 54, 22, 62, 30, 37,

5, 45, 13, 53, 21, 61,

29, 36, 4, 44, 12, 52,

20, 60, 28, 35, 3, 43,

11, 51, 19, 59, 27, 34,

2, 42, 10, 50, 18, 58,

26, 33, 1, 41, 9, 49,

17, 57, 25 };

// first key-hePermutation Table

int[] PC1 = { 57, 49, 41, 33, 25,

17, 9, 1, 58, 50, 42, 34, 26,

18, 10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36, 63,

55, 47, 39, 31, 23, 15, 7, 62,

54, 46, 38, 30, 22, 14, 6, 61,

53, 45, 37, 29, 21, 13, 5, 28,

20, 12, 4 };

// second key-Permutation Table

int[] PC2 = { 14, 17, 11, 24, 1, 5, 3,

28, 15, 6, 21, 10, 23, 19, 12,

4, 26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40,

51, 45, 33, 48, 44, 49, 39, 56,

34, 53, 46, 42, 50, 36, 29, 32 };

// Expansion D-box Table

int[] EP = { 32, 1, 2, 3, 4, 5, 4,

5, 6, 7, 8, 9, 8, 9, 10,

11, 12, 13, 12, 13, 14, 15,

16, 17, 16, 17, 18, 19, 20,

21, 20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29, 28,

29, 30, 31, 32, 1 };

// Straight Permutation Table

int[] P = { 16, 7, 20, 21, 29, 12, 28,

17, 1, 15, 23, 26, 5, 18,

31, 10, 2, 8, 24, 14, 32,

27, 3, 9, 19, 13, 30, 6,

22, 11, 4, 25 };

// S-box Table

int[][][] sbox = {

{ { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },

{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },

{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },

{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } },

{ { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },

{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },

{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },

{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } },

{ { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },

{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },

{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },

{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } },

{ { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },

{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },

{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },

{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } },

{ { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },

{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },

{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },

{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } },

{ { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },

{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },

{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },

{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } },

{ { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },

{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },

{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },

{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } },

{ { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },

{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },

{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },

{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } }

};

int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1 };

// hexadecimal to binary conversion

String hextoBin(String input)

{

int n = input.length() \* 4;

input = Long.toBinaryString(

Long.parseUnsignedLong(input, 16));

while (input.length() < n)

input = "0" + input;

return input;

}

// binary to hexadecimal conversion

String binToHex(String input)

{

int n = (int)input.length() / 4;

input = Long.toHexString(

Long.parseUnsignedLong(input, 2));

while (input.length() < n)

input = "0" + input;

return input;

}

// per-mutate input hexadecimal

// according to specified sequence

String permutation(int[] sequence, String input)

{

String output = "";

input = hextoBin(input);

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

output += input.charAt(sequence[i] - 1);

output = binToHex(output);

return output;

}

// xor 2 hexadecimal strings

String xor(String a, String b)

{

// hexadecimal to decimal(base 10)

long t\_a = Long.parseUnsignedLong(a, 16);

// hexadecimal to decimal(base 10)

long t\_b = Long.parseUnsignedLong(b, 16);

// xor

t\_a = t\_a ^ t\_b;

// decimal to hexadecimal

a = Long.toHexString(t\_a);

// prepend 0's to maintain length

while (a.length() < b.length())

a = "0" + a;

return a;

}

// left Circular Shifting bits

String leftCircularShift(String input, int numBits)

{

int n = input.length() \* 4;

int perm[] = new int[n];

for (int i = 0; i < n - 1; i++)

perm[i] = (i + 2);

perm[n - 1] = 1;

while (numBits-- > 0)

input = permutation(perm, input);

return input;

}

// preparing 16 keys for 16 rounds

String[] getKeys(String key)

{

String keys[] = new String[16];

// first key permutation

key = permutation(PC1, key);

for (int i = 0; i < 16; i++) {

key = leftCircularShift(

key.substring(0, 7), shiftBits[i])

+ leftCircularShift(key.substring(7, 14),

shiftBits[i]);

// second key permutation

keys[i] = permutation(PC2, key);

}

return keys;

}

// s-box lookup

String sBox(String input)

{

String output = "";

input = hextoBin(input);

for (int i = 0; i < 48; i += 6) {

String temp = input.substring(i, i + 6);

int num = i / 6;

int row = Integer.parseInt(

temp.charAt(0) + "" + temp.charAt(5), 2);

int col = Integer.parseInt(

temp.substring(1, 5), 2);

output += Integer.toHexString(

sbox[num][row][col]);

}

return output;

}

String round(String input, String key, int num)

{

// fk

String left = input.substring(0, 8);

String temp = input.substring(8, 16);

String right = temp;

// Expansion permutation

temp = permutation(EP, temp);

// xor temp and round key

temp = xor(temp, key);

// lookup in s-box table

temp = sBox(temp);

// Straight D-box

temp = permutation(P, temp);

// xor

left = xor(left, temp);

System.out.println("Round "

+ (num + 1) + " "

+ right.toUpperCase()

+ " " + left.toUpperCase() + " "

+ key.toUpperCase());

// swapper

return right + left;

}

String encrypt(String plainText, String key)

{

int i;

// get round keys

String keys[] = getKeys(key);

// initial permutation

plainText = permutation(IP, plainText);

System.out.println(

"After initial permutation: "

+ plainText.toUpperCase());

System.out.println(

"After splitting: L0="

+ plainText.substring(0, 8).toUpperCase()

+ " R0="

+ plainText.substring(8, 16).toUpperCase() + "\n");

// 16 rounds

for (i = 0; i < 16; i++) {

plainText = round(plainText, keys[i], i);

}

// 32-bit swap

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

// final permutation

plainText = permutation(IP1, plainText);

return plainText;

}

String decrypt(String plainText, String key)

{

int i;

// get round keys

String keys[] = getKeys(key);

// initial permutation

plainText = permutation(IP, plainText);

System.out.println(

"After initial permutation: "

+ plainText.toUpperCase());

System.out.println(

"After splitting: L0="

+ plainText.substring(0, 8).toUpperCase()

+ " R0=" + plainText.substring(8, 16).toUpperCase()

+ "\n");

// 16-rounds

for (i = 15; i > -1; i--) {

plainText = round(plainText, keys[i], 15 - i);

}

// 32-bit swap

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

plainText = permutation(IP1, plainText);

return plainText;

}

}

public static void main(String args[])

{

System.out.println(" 18B91A12F7");

System.out.println(" VARIGONDA SAI NIRMAL VIGNU");

String text = "123456ABCD132536";

String key = "AABB09182736CCDD";

DES cipher = new DES();

System.out.println("Encryption:\n");

text = cipher.encrypt(text, key);

System.out.println(

"\nCipher Text: " + text.toUpperCase() + "\n");

System.out.println("Decryption\n");

text = cipher.decrypt(text, key);

System.out.println(

"\nPlain Text: "

+ text.toUpperCase());

}

}

Graphical user interface, text

Description automatically generated