**CAESAR**

import java.util.Scanner;

public class Main {

static Boolean givenInput;

static String inputString;

static int key;

static StringBuffer encryptedString;

static StringBuffer decryptedString;

public Main() {

givenInput = false;

encryptedString = new StringBuffer();

decryptedString = new StringBuffer();

}

public static void Cipher() {

encryptedString = new StringBuffer();

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

if (Character.isUpperCase(inputString.charAt(i))) {

char temp = (char)(Math.floorMod(inputString.charAt(i) + key - 65, 26) + 65);

encryptedString.append(temp);

} else {

char temp = (char) (Math.floorMod(inputString.charAt(i) + key - 97, 26) + 97);

encryptedString.append(temp);

}

}

}

public static void De\_Cipher() {

decryptedString = new StringBuffer();

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

if (Character.isUpperCase(encryptedString.charAt(i))) {

char temp = (char)(Math.floorMod(encryptedString.charAt(i) - key - 65, 26) + 65);

decryptedString.append(temp);

} else {

char temp = (char) (Math.floorMod(encryptedString.charAt(i) - key - 97, 26) + 97);

decryptedString.append(temp);

}

}

}

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("-------------------CAESAR CIPHER-------------------");

int choice = 0;

while (choice != -1) {

System.out.println("1. Input the plain text");

System.out.println("2. Input the key");

System.out.println("3. Encrypt");

System.out.println("4. Decrypt");

System.out.println("5. Exit");

choice = scan.nextInt();

switch (choice) {

case 3:

if (givenInput) {

Cipher();

System.out.println(encryptedString);

} else {

choice = 3;

}

break;

case 4:

if (givenInput) {

De\_Cipher();

System.out.println(decryptedString);

} else {

choice = 3;

}

break;

case 1:

System.out.println("Enter the text - ");

inputString = scan.next();

givenInput = true;

// System.out.println(inputString);

break;

case 2:

System.out.println("Enter the Key - ");

key = scan.nextInt();

givenInput = true;

// System.out.println(key);

break;

case 5:

choice = -1;

break;

default:

System.out.println("Please enter one of the above mentioned options :");

}

}

}

}

**B. PLAYFAIR**

import java.util.Scanner;

public class Main

{

private String KeyWord=new String();

private String Key=new String();

private char key\_matrix[][]= new char[5][5];

public void setKey(String k)

{

String K\_adjust=new String();

boolean flag = false;

K\_adjust = K\_adjust + k.charAt(0);

for(int i=1; i<k.length();i++)

{

for(int j=0;j<K\_adjust.length(); j++)

{

if(k.charAt(i)==K\_adjust.charAt(j))

{

flag = true;

}

}

if(flag == false)

K\_adjust = K\_adjust + k.charAt(i);

flag = false;

}

KeyWord=K\_adjust;

}

public void KeyGen()

{

boolean flag=true;

char current;

Key=KeyWord;

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

{

current=(char)(i+97);

if(current=='j')

continue;

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

{

if (current == KeyWord.charAt(j))

{

flag=false;

break;

}

}

if(flag)

Key=Key+current;

flag=true;

}

// System.out.println(Key);

matrix ();

}

private void matrix ()

{

System.out.println("MATRIX:");

int counter=0;

for (int i=0 ; i<5 ;i++)

{

for (int j=0 ; j<5 ; j++)

{

key\_matrix[i][j]=Key.charAt(counter);

System.out.printf("%s ",key\_matrix[i][j]);

counter++;

}

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

}

}

private String format(String old\_text)

{

int i = 0;

int j = 0;

int len = 0;

String text = new String();

len = old\_text.length();

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

{

if (old\_text.charAt(tmp) == 'j')

{

text = text + 'i';

}

else if(old\_text.charAt(tmp)==' '){

continue;

}

else

text = text+old\_text.charAt(tmp);

}

len = text.length();

for (i = 0; i < len-1; i = i + 2)

{

if (text.charAt(i+1) == text.charAt(i))

{

text = text.substring(0, i+1) + 'x' + text.substring(i+1);

}

else

{}

}

return text;

}

private String [] Divid2Pairs (String new\_string)

{

String Original = format(new\_string);

int size= Original.length();

if(size%2!=0)

{

size++;

Original = Original+'x';

}

String x[]= new String[size/2];

int counter=0;

for ( int i=0 ; i<size/2 ;i++)

{

x[i]=Original.substring(counter, counter+2);

System.out.print(x[i]+'\t');

counter=counter+2;

}

System.out.println();

return x;

}

public int[] GetDimensions(char letter)

{

int []key=new int[2];

if ( letter == 'j')

letter='i';

for (int i=0 ; i<5 ;i++)

{

for (int j=0 ; j<5 ; j++)

{

if(key\_matrix[i][j] == letter)

{

key[0]=i;

key[1]=j;

break;

}

}

}

return key;

}

public String Encrypt(String Source)

{

System.out.println("Encryption Start");

String src\_arr[]=Divid2Pairs(Source);

String Code=new String();

char one;

char two;

int part1[]=new int[2];

int part2[]=new int[2];

for (int i=0 ; i< src\_arr.length ;i++ )

{

one = src\_arr[i].charAt(0);

two = src\_arr[i].charAt(1);

part1 = GetDimensions(one);

part2 = GetDimensions(two);

if(part1[0]==part2[0])

{

if (part1[1]<4)

part1[1]++;

else

part1[1]=0;

if(part2[1]<4)

part2[1]++;

else

part2[1]=0;

}

else if (part1[1]==part2[1])

{

if (part1[0]<4)

part1[0]++;

else

part1[0]=0;

if(part2[0]<4)

part2[0]++;

else

part2[0]=0;

}

else

{

int temp=part1[1];

part1[1]=part2[1];

part2[1]=temp;

}

Code= Code + key\_matrix[part1[0]][part1[1]] + key\_matrix[part2[0]][part2[1]];

}

System.out.println("Encrypted Text: "+Code+'\n');

return Code;

}

public String Decrypt (String Code, int s)

{

System.out.println("Decryption Start");

String Original=new String();

String src\_arr[]=Divid2Pairs(Code);

char one;

char two;

int part1[]=new int[2];

int part2[]=new int[2];

for (int i=0 ; i< src\_arr.length ;i++ )

{

one = src\_arr[i].charAt(0);

two = src\_arr[i].charAt(1);

part1 = GetDimensions(one);

part2 = GetDimensions(two);

if(part1[0]==part2[0])

{

if (part1[1]>0)

part1[1]--;

else

part1[1]=4;

if(part2[1]>0)

part2[1]--;

else

part2[1]=4;

}

else if (part1[1]==part2[1])

{

if (part1[0]>0)

part1[0]--;

else

part1[0]=4;

if(part2[0]>0)

part2[0]--;

else

part2[0]=4;

}

else

{

int temp=part1[1];

part1[1]=part2[1];

part2[1]=temp;

}

Original =Original + key\_matrix[part1[0]][part1[1]] + key\_matrix[part2[0]][part2[1]];

}

String result=new String();

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

char c=Original.charAt(j);

result=result+c;

}

System.out.println("Decrypted Text: "+result+"\n");

return result;

}

public static void main(String[] args)

{

Main x=new Main();

int choice = 0;

Scanner sc = new Scanner(System.in);

String key\_input="";

String Encrypted="";

int s=0;

while(choice!=-1){

System.out.println("Enter your choice:\n1.Enter keyword\n2.Enter text to be encrypted\n3.Encrypt\n4.Decrypt\n5.Exit");

choice=sc.nextInt();

sc.nextLine();

switch (choice) {

case 1:

System.out.print("Enter keyword for Cipher: ");

String keyword = sc.nextLine();

x.setKey(keyword);

x.KeyGen();

break;

case 2:

System.out.print("Enter text: ");

key\_input = sc.nextLine();

s=key\_input.length();

break;

case 3:

Encrypted= x.Encrypt(key\_input);

break;

case 4:

x.Decrypt(Encrypted,s);

break;

case 5:

choice=-1;

break;

default:

System.out.println("Please enter one of the above mentioned options :");

}

}

}

}

**HILL**

import java.util.\*;

import java.io.\*;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

static int[] lm;

static int[][] keyMatrix;

static int[] rm;

static int choice;

static int [][] inverseKeyMatrix;

static int casevariable;

static String line="";

// Display function to print a ma1trix

public static void displayMatrix(int A[][],int len) {

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

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

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

System.out.println();

}

}

// Perform encryption/decryption

public static void performEncryptionOrDecryption(String temp, int s)

{

while (temp.length() > s)

{

String line = temp.substring(0, s);

temp = temp.substring(s, temp.length());

findColumnMatrix(line);

if(choice ==1){

multiplyColumnByKey(line.length());

showResult(line.length());

}

else if(choice==2){

multiplyColumnByInverseKey(line.length());

showResult(line.length());

}

}

if (temp.length() == s){

if(choice ==1){

findColumnMatrix(temp);

multiplyColumnByKey(temp.length());

showResult(temp.length());

}

else if(choice==2){

findColumnMatrix(temp);

multiplyColumnByInverseKey(temp.length());

showResult(temp.length());

}

}

else if (temp.length() < s)

{

for (int i = temp.length(); i < s; i++)

temp = temp + 'x';

if(choice ==1){

findColumnMatrix(temp);

multiplyColumnByKey(temp.length());

showResult(temp.length());

}

else if(choice==2){

findColumnMatrix(temp);

multiplyColumnByInverseKey(temp.length());

showResult(temp.length());

}

}

}

// Compute the key matrix

public static void findKeyMatrix(String key, int len)

{

keyMatrix = new int[len][len];

int k = 0;

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

{

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

{

keyMatrix[i][j] = ((int) key.charAt(k)) - casevariable;

k++;

}

}

if(choice==1)

{

System.out.println("\nKEY MATRIX");

System.out.println("----------");

displayMatrix(keyMatrix,len);

System.out.print("\nCipher Text : ");

}

}

// Take each group of input variables and put them into a col matrix

public static void findColumnMatrix(String line)

{

lm = new int[line.length()];

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

{

lm[i] = ((int) line.charAt(i)) - casevariable;

}

}

public static void multiplyColumnByKey(int len)

{

rm = new int[len];

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

{

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

{

rm[i] += keyMatrix[i][j] \* lm[j];

}

rm[i] %= 26;

}

}

public static void multiplyColumnByInverseKey(int len)

{

rm = new int[len];

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

{

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

{

rm[i] += inverseKeyMatrix[i][j] \* lm[j];

}

rm[i] %= 26;

}

}

public static void showResult(int len)

{

String result = "";

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

{

result += (char) (rm[i] + casevariable);

}

System.out.print(result);

}

public static int findDeterminant(int A[][], int N)

{

int resultOfDet;

switch (N) {

case 1:

resultOfDet = A[0][0];

break;

case 2:

resultOfDet = A[0][0] \* A[1][1] - A[1][0] \* A[0][1];

break;

default:

resultOfDet = 0;

for (int j1 = 0; j1 < N; j1++)

{

int m[][] = new int[N - 1][N - 1];

for (int i = 1; i < N; i++)

{

int j2 = 0;

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

{

if (j == j1)

continue;

m[i - 1][j2] = A[i][j];

j2++;

}

}

resultOfDet += Math.pow(-1.0, 1.0 + j1 + 1.0) \* A[0][j1]

\* findDeterminant(m, N - 1);

}

break;

}

return resultOfDet;

}

public static void findCoFactor(int num[][], int f)

{

int b[][], fac[][];

b = new int[f][f];

fac = new int[f][f];

int p, q, m, n, i, j;

for (q = 0; q < f; q++)

{

for (p = 0; p < f; p++)

{

m = 0;

n = 0;

for (i = 0; i < f; i++)

{

for (j = 0; j < f; j++)

{

b[i][j] = 0;

if (i != q && j != p)

{

b[m][n] = num[i][j];

if (n < (f - 2))

n++;

else

{

n = 0;

m++;

}

}

}

}

fac[q][p] = (int) Math.pow(-1, q + p) \* findDeterminant(b, f - 1);

}

}

findTranspose(fac, f);

}

static void findTranspose(int fac[][], int r)

{

int i, j;

int b[][], inv[][];

b = new int[r][r];

inv = new int[r][r];

int d = findDeterminant(keyMatrix, r);

int mi = mi(d % 26);

mi %= 26;

if (mi < 0)

mi += 26;

for (i = 0; i < r; i++)

{

for (j = 0; j < r; j++)

{

b[i][j] = fac[j][i];

}

}

for (i = 0; i < r; i++)

{

for (j = 0; j < r; j++)

{

inv[i][j] = b[i][j] % 26;

if (inv[i][j] < 0)

inv[i][j] += 26;

inv[i][j] \*= mi;

inv[i][j] %= 26;

}

}

//System.out.println("\nInverse key:");

//matrixtoinverseKeyMatrixey(inv, r);

inverseKeyMatrix = inv;

if(choice==2)

{

System.out.println("\nINVERSE KEY MATRIX");

System.out.println("------------------");

displayMatrix(inverseKeyMatrix,r);

System.out.print("\nOriginal Text : ");

}

}

public static int mi(int d)

{

int q, r1, r2, r, t1, t2, t;

r1 = 26;

r2 = d;

t1 = 0;

t2 = 1;

while (r1 != 1 && r2 != 0)

{

q = r1 / r2;

r = r1 % r2;

t = t1 - (t2 \* q);

r1 = r2;

r2 = r;

t1 = t2;

t2 = t;

}

return (t1 + t2);

}

// Check if key matrix is invertible

public static boolean check(String key, int len)

{

findKeyMatrix(key, len);

int d = findDeterminant(keyMatrix, len);

d = d % 26;

if (d == 0)

{

System.out.println("Key is not invertible");

return false;

}

else if (d % 2 == 0 || d % 13 == 0)

{

System.out.println("Key is not invertible");

return false;

}

else

{

return true;

}

}

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

{

String key="";

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

System.out.println("\nOPTIONS");

System.out.println("-------");

System.out.println("1. Encrypt\n2. Decrypt\n3. Exit\n");

choice = -1;

while(choice!=3)

{

System.out.print("Enter option : ");

choice = Integer.parseInt(in.readLine());

if(choice==1)

{

System.out.print("Enter the Plain Text to Encrypt : ");

line = in.readLine();

System.out.print("Enter the Key : ");

key = in.readLine();

}

else if(choice==2)

{

System.out.print("Enter the Cipher Text to Decrypt : ");

line = in.readLine();

System.out.print("Enter the Key : ");

key = in.readLine();

}

line = line.replaceAll("\\s+","");

if(Character.isUpperCase(line.charAt(0)))

{

casevariable = 65;

}

else

{

casevariable = 97;

}

double sq = Math.sqrt(key.length());

if (sq != (long) sq)

System.out.println("Cannot Form a Square Matrix !\n");

else

{

int size = (int) sq;

if (check(key, size))

{

findCoFactor(keyMatrix, size);

performEncryptionOrDecryption(line, size);

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

}

}

}

}

}

**VIGNERE**

import java.util.Scanner;

class Main

{

static String generateKey(String str, String key)

{

int x = str.length();

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

{

if (x == i)

i = 0;

if (key.length() == str.length())

break;

key+=(key.charAt(i));

}

return key;

}

// This function returns the encrypted text

// generated with the help of the key

static String cipherText(String str, String key)

{

String cipher\_text="";

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

{

// converting in range 0-25

int x = (str.charAt(i) + key.charAt(i)) %26;

// convert into alphabets(ASCII)

x += 'A';

cipher\_text+=(char)(x);

}

return cipher\_text;

}

// This function decrypts the encrypted text

// and returns the original text

static String originalText(String cipher\_text, String key)

{

String orig\_text="";

for (int i = 0 ; i < cipher\_text.length() &&

i < key.length(); i++)

{

// converting in range 0-25

int x = (cipher\_text.charAt(i) -

key.charAt(i) + 26) %26;

// convert into alphabets(ASCII)

x += 'A';

orig\_text+=(char)(x);

}

return orig\_text;

}

static void print\_matrix ()

{

char c,d;

for (c = 'A' ; c <= 'Z' ; c++)

{

for ( d=c; d <= 'Z' ; d++)

{

System.out.print(d + " ");

}

for ( d= 'A'; d < c ; d++)

{

System.out.print(d + " ");

}

System.out.println();

}

}

// Driver code

public static void main(String[] args)

{

int choice = 0;

Scanner sc = new Scanner(System.in);

String key\_input="";

String key = "";

String Encrypted="";

int n =0;

while(true){

System.out.println("Enter your choice:\n1.Input PlainText \n2.Input keyword \n3.Print Matrix \n4.Encrypt \n5.Decrypt \n6.Exit");

choice=sc.nextInt();

sc.nextLine();

if(choice == 6){

break;

}

switch (choice) {

case 2:

System.out.print("Enter keyword : ");

String keyword = sc.nextLine();

key = generateKey(key\_input, keyword);

break;

case 1:

System.out.print("Enter PlainText: ");

key\_input = sc.nextLine();

break;

case 3:

print\_matrix();

break;

case 4:

Encrypted = cipherText(key\_input, key);

System.out.println("Ciphertext : "+ Encrypted + "\n");

break;

case 5:

System.out.println("Original/Decrypted Text :" + originalText(Encrypted, key));

break;

default:

System.out.println("Invalid Choice");

}

}

}

}

**RAIL**

import java.util.\*;

import java.io.\*;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String arg[]){

Scanner inn=new Scanner (System.in);

System.out.println("Rail Fence cipher:");

int choice = 0;

while (choice != -1) {

System.out.println("1. Encrypt");

System.out.println("2. Decrypt");

System.out.println("3. Exit");

choice = inn.nextInt();

switch (choice) {

case 1:

System.out.println("Enter the plaintext for encryption");

String plaintext=inn.next();

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

int rails=inn.nextInt();

encryption(plaintext,rails);

break;

case 2:

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

rails=inn.nextInt();

System.out.println("Enter the ciphertext for decryption:");

String ciphertext=inn.next();

decryption(ciphertext,rails);

System.out.println();

break;

case 3:

choice = -1;

break;

default:

System.out.println("Please enter one of the above mentioned options :");

}

}

// System.out.println("Enter the plaintext for encryption");

// Scanner inn=new Scanner (System.in);

// String plaintext=inn.next();

// plaintext = plaintext.replaceAll("\\s+", "");

// System.out.println(plaintext);

// System.out.println("Decryption process start:");

}

public static void encryption(String str,int rails){

boolean checkdown=false; //check whether it is moving downward or upward

int j=0;

int row=rails; // no of row is the no of rails entered by user

int col=str.length(); //column length is the size of string

char[][] a=new char[row][col];

//we create a matrix of a of row \*col size

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

//matrix visiting in rails order and putting the character of plaintext

if(j==0||j==row-1)

checkdown=!checkdown;

a[j][i]=str.charAt(i);

if(checkdown){

j++;

}

else

j--;

}

//visiting the matrix in usual order to get ciphertext

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

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

System.out.print(a[i][k]+" ");

}

System.out.println();

}

String en="";

System.out.println("----------------------");

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

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

if(a[i][k]!=0)

en=en+a[i][k];

}

}

System.out.println(en);//printing the ciphertext

}

public static void decryption(String str,int rails){

boolean checkdown=false;

int j=0;

int row=rails;

int col=str.length();

char[][] a=new char[row][col];

//first of all mark the rails position by \* in the matrix

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

if(j==0||j==row-1)

checkdown=!checkdown;

a[j][i]='\*';

if(checkdown)j++;

else j--;

}

//now enter the character of cipheetext in the matrix positon that have \* symbol

int index=0;

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

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

if(a[i][k]=='\*'&&index<str.length()){

a[i][k]=str.charAt(index++);

}

}

}

// visit each character in rails order as character are put in the encryption function

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

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

System.out.print(a[i][k]+ "\t");

}

System.out.println();

}

checkdown=false;

String s="";

j=0;

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

if( j==0||j==row-1)

checkdown=!checkdown;

s+=a[j][i];

if(checkdown)j++;

else j--;

}

System.out.print(s);//print the plaintext that was decrypted by rail fence cipher

}

}

**ROW:**

import java.util.\*;

class Main{

static String message;

static String key;

static int columnCount;

static int rowCount;

static int plainText[][];

static int cipherText[][];

static String ct;

public static void main(String sap[]){

Scanner sc = new Scanner(System.in);

System.out.println("Row Transposition Cipher:");

int choice = 0;

while (choice != -1) {

System.out.println("1. Input plaintext");

System.out.println("2. Input key ");

System.out.println("3. Encrypt");

System.out.println("4. Decrypt");

System.out.println("5. Exit");

choice = sc.nextInt();

switch (choice) {

case 1:

System.out.print("\nEnter plaintext(enter in lower case): ");

message = sc.next();

break;

case 2:

System.out.print("\nEnter key in numbers: ");

key = sc.next();

break;

case 3:

columnCount = key.length();

rowCount = (message.length()+columnCount)/columnCount;

plainText = new int[rowCount][columnCount];

cipherText =new int[rowCount][columnCount];

cipherText = encrypt(plainText, cipherText, message, rowCount, columnCount, key);

ct = "";

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

{

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

{

if(cipherText[j][i] == 0)

ct = ct + 'x';

else{

ct = ct + (char)cipherText[j][i];

}

}

}

System.out.print("\nCipher Text: " + ct);

System.out.println();

break;

case 4:

plainText = decrypt(plainText, cipherText, ct, rowCount, columnCount, key);

String pt = "";

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

{

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

{

if(plainText[i][j] == 0)

pt = pt + "";

else{

pt = pt + (char)plainText[i][j];

}

}

}

System.out.print("\nPlain Text: " + pt);

System.out.println();

break;

case 5:

choice = -1;

break;

default:

System.out.println("Please enter one of the above mentioned options :");

}

}

}

static int[][] encrypt(int plainText[][], int cipherText[][], String message, int rowCount, int columnCount, String key){

int i,j;

int k=0;

for(i=0; i<rowCount; i++)

{

for(j=0; j<columnCount; j++)

{

if(k < message.length())

{

plainText[i][j] = (int)message.charAt(k);

k++;

}

else

{

break;

}

}

}

System.out.print("PlainText Array: \n");

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

for(j=0;j<columnCount;j++){

System.out.print((char)plainText[i][j]+"\t");

}

System.out.println();

}

for(i=0; i<columnCount; i++)

{

int currentCol= ( (int)key.charAt(i) - 48 ) -1;

for(j=0; j<rowCount; j++)

{

cipherText[j][i] = plainText[j][currentCol];

}

}

System.out.print("Cipher Array(read column by column): \n");

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

for(j=0;j<columnCount;j++){

System.out.print((char)cipherText[i][j]+"\t");

}

System.out.println();

}

return cipherText;

}

static int[][] decrypt(int plainText[][], int cipherText[][], String message, int rowCount, int columnCount, String key){

int i,j;

int k=0;

System.out.print("Cipher Array: \n");

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

for(j=0;j<columnCount;j++){

System.out.print((char)cipherText[i][j]+"\t");

}

System.out.println();

}

for(i=0; i<columnCount; i++)

{

int currentCol= ( (int)key.charAt(i) - 48 ) -1;

for(j=0; j<rowCount; j++)

{

plainText[j][currentCol] = cipherText[j][i];

}

}

System.out.print("Plain Array(read row by row): \n");

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

for(j=0;j<columnCount;j++){

System.out.print((char)plainText[i][j]+"\t");

}

System.out.println();

}

return plainText;

}

}

**3. AES:**

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 Main {

byte[] skey = new byte[1000];

String skeyString;

static byte[] raw;

String inputMessage,encryptedData,decryptedMessage,keyMessage;

public Main() {

try {

// generateSymmetricKey();

inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");

byte[] ibyte = inputMessage.getBytes();

keyMessage=JOptionPane.showInputDialog(null,"Enter key for encryption");

raw = keyMessage.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("AES Symmetric key = "+skeyString);

}

catch(Exception e) {

System.out.println(e);

}

}

private static byte[] getRawKey(byte[] seed) throws Exception {

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

SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");

sr.setSeed(seed);

kgen.init(128, sr); // 192 and 256 bits may not be available

SecretKey skey = kgen.generateKey();

raw = skey.getEncoded();

return raw;

}

private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {

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

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

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, "AES");

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

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

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

return decrypted;

}

public static void main(String args[]) {

Main aes = new Main();

}}

AES.java---

AES\_LIB

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 Main {

byte[] skey = new byte[1000];

String skeyString;

static byte[] raw;

String inputMessage, encryptedData, decryptedMessage;

public Main() {

try {

generateSymmetricKey();

inputMessage = JOptionPane.showInputDialog(null, &quot;Enter message to encrypt&quot;);

byte[] ibyte = inputMessage.getBytes();

byte[] ebyte = encrypt(raw, ibyte);

String encryptedData = new String(ebyte);

System.out.println(&quot;Encrypted message &quot; + encryptedData);

JOptionPane.showMessageDialog(null, &quot;Encrypted Data &quot; + &quot;\n&quot; + encryptedData);

byte[] dbyte = decrypt(raw, ebyte);

String decryptedMessage = new String(dbyte);

System.out.println(&quot;Decrypted message &quot; + decryptedMessage);

JOptionPane.showMessageDialog(null, &quot;Decrypted Data &quot; + &quot;\n&quot; + 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(&quot;AES Symmetric key = &quot; + skeyString);

} catch (Exception e) {

System.out.println(e);

}

}

private static byte[] getRawKey(byte[] seed) throws Exception {

KeyGenerator kgen = KeyGenerator.getInstance(&quot;AES&quot;);

SecureRandom sr = SecureRandom.getInstance(&quot;SHA1PRNG&quot;);

sr.setSeed(seed);

kgen.init(128, sr); // 192 and 256 bits may not be available

SecretKey skey = kgen.generateKey();

raw = skey.getEncoded();

return raw;

}

private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {

SecretKeySpec skeySpec = new SecretKeySpec(raw, &quot;AES&quot;);

Cipher cipher = Cipher.getInstance(&quot;AES&quot;);

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, &quot;AES&quot;);

Cipher cipher = Cipher.getInstance(&quot;AES&quot;);

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

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

return decrypted;

}

public static void main(String args[]) {

new Main();}}

AES

import java.util.Scanner;

public class AESKEYGEN {

static char sbox[] = {

//0 1 2 3 4 5 6 7 8 9 A

B C D E F

0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,

0x2b, 0xfe, 0xd7, 0xab, 0x76, //0

0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,

0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1

0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,

0xf1, 0x71, 0xd8, 0x31, 0x15, //2

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,

0xe2, 0xeb, 0x27, 0xb2, 0x75, //3

0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,

0xb3, 0x29, 0xe3, 0x2f, 0x84, //4

0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,

0x39, 0x4a, 0x4c, 0x58, 0xcf, //5

0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,

0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6

0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,

0x21, 0x10, 0xff, 0xf3, 0xd2, //7

0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,

0x3d, 0x64, 0x5d, 0x19, 0x73, //8

0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,

0x14, 0xde, 0x5e, 0x0b, 0xdb, //9

0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,

0x62, 0x91, 0x95, 0xe4, 0x79, //A

0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,

0xea, 0x65, 0x7a, 0xae, 0x08, //B

0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,

0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C

0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,

0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D

0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,

0xe9, 0xce, 0x55, 0x28, 0xdf, //E

0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,

0x0f, 0xb0, 0x54, 0xbb, 0x16}; //F

static char rsbox[]

= {0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40,

0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f,

0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, 0x54, 0x7b,

0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa,

0xc3, 0x4e, 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b,

0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25, 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68,

0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, 0x6c, 0x70,

0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d,

0x9d, 0x84, 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4,

0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06, 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f,

0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, 0x3a, 0x91,

0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4,

0xe6, 0x73, 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9,

0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e, 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29,

0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b, 0xfc, 0x56,

0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd,

0x5a, 0xf4, 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12,

0x10, 0x59, 0x27, 0x80, 0xec, 0x5f, 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5,

0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, 0xa0, 0xe0,

0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53,

0x99, 0x61, 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69,

0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d};

static char Rcon[] = {

0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,

0x6c, 0xd8,

0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,

0xd4, 0xb3,

0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,

0xc2, 0x9f,

0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,

0xcb, 0x8d,

0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,

0xd8, 0xab,

0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,

0xb3, 0x7d,

0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2,

0x9f, 0x25,

0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb,

0x8d, 0x01,

0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,

0xab, 0x4d,

0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,

0x7d, 0xfa,

0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,

0x25, 0x4a,

0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d,

0x01, 0x02,

0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,

0x4d, 0x9a,

0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,

0xfa, 0xef,

0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,

0x4a, 0x94,

0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,

0x02, 0x04,

0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,

0x9a, 0x2f,

0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,

0xef, 0xc5,

0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,

0x94, 0x33,

0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb};

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

char plainText[][] = new char[4][4]; //

&quot;00000000000000000000000000000000&quot;;

char cipherKey[][] = new char[4][8]; //

&quot;00000000000000000000000000000000&quot;;

System.out.println(&quot;Enter the key: &quot;);

String line3 = new Scanner(System.in).nextLine();

while(line3.length() &lt; 32)

line3 += &quot;0&quot;;

if(line3.length() &gt; 32)

line3 = line3.substring(0, 32);

char temp3[] = line3.toCharArray();

int c4 = 0;

for (int c2 = 0; c2 &lt; 32; c2 += 2) {

cipherKey[c4 % 4][c4++ / 4] = (char)Integer.parseInt(&quot;&quot; +

temp3[c2] + temp3[c2+1], 16);

}

System.out.println(&quot;Cipher key:&quot;);

printMatrix(cipherKey, 4, 4);

// print 2D array for debugging

// printMatrix(cipherKey, 4, 8);

// System.out.println(&quot;Rcon &quot; + Integer.toHexString((int)

getRconValue((char) 5)));

// System.out.println(&quot;Plain Text: &quot;);

char extendedKey[][] = new char[4][60];

extendedKey = extendKey(cipherKey);

System.out.println(&quot;Extended key:&quot;);

printMatrix(extendedKey, 4, 40);

}

public static char[][] extendKey(char key[][]) {

char extended[][] = new char[4][60];

for (int c = 0; c &lt; 4; c++) {

for (int c2 = 0; c2 &lt; 8; c2++) {

extended[c][c2] = key[c][c2];

}

}

char c = 1;

for (int i = 7; i &lt; 55; i++) {

// STEP 1

// rotate

// System.out.println(&quot;c: &quot; + (int) c + &quot; i: &quot; + i + &quot; rot 4

byte: &quot;);

// System.out.println(&quot;&quot; + String.format(&quot;%02X&quot;, (int)

extended[0][i]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) extended[1][i]) + &quot; &quot;

+ String.format(&quot;%02X&quot;, (int) extended[2][i]) + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) extended[3][i]));

char tempArr[] = new char[4];

tempArr[0] = extended[1][i];

tempArr[1] = extended[2][i];

tempArr[2] = extended[3][i];

tempArr[3] = extended[0][i];

// System.out.println(&quot;After rotation&quot; + String.format(&quot;%02X&quot;,

(int) tempArr[0]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) tempArr[1]) + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) tempArr[2]) + &quot; &quot; + String.format(&quot;%02X&quot;,

(int) tempArr[3]));

//substitute

tempArr[0] = getSBoxValue(tempArr[0]);

tempArr[1] = getSBoxValue(tempArr[1]);

tempArr[2] = getSBoxValue(tempArr[2]);

tempArr[3] = getSBoxValue(tempArr[3]);

// System.out.println(&quot;After sub&quot; + String.format(&quot;%02X&quot;, (int)

tempArr[0]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) tempArr[1]) + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) tempArr[2]) + &quot; &quot; + String.format(&quot;%02X&quot;,

(int) tempArr[3]));

char rCon = getRconValue(c);

c = (char) (c + 1);

tempArr[0] ^= rCon;

extended[0][i + 1] = (char) (extended[0][i - 3] ^ tempArr[0]);

extended[1][i + 1] = (char) (extended[1][i - 3] ^ tempArr[1]);

extended[2][i + 1] = (char) (extended[2][i - 3] ^ tempArr[2]);

extended[3][i + 1] = (char) (extended[3][i - 3] ^ tempArr[3]);

//debugging point

// System.out.println(&quot;c: &quot; + (int)c );

// System.out.println(&quot;new 4 byte: &quot;);

// System.out.println(&quot;&quot; + String.format(&quot;%02X&quot;, (int)

extended[0][i + 1]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) extended[1][i +

1]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) extended[2][i + 1]) + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) extended[3][i + 1]));

// STEP 2

for (int c2 = 0; c2 &lt; 3; c2++) {

i++;

extended[0][i + 1] = (char) (extended[0][i - 3] ^

extended[0][i]);

extended[1][i + 1] = (char) (extended[1][i - 3] ^

extended[1][i]);

extended[2][i + 1] = (char) (extended[2][i - 3] ^

extended[2][i]);

extended[3][i + 1] = (char) (extended[3][i - 3] ^

extended[3][i]);

}

//STEP 3

i++;

tempArr[0] = extended[0][i];

tempArr[1] = extended[1][i];

tempArr[2] = extended[2][i];

tempArr[3] = extended[3][i];

// System.out.println(&quot; step3 begins&quot; + String.format(&quot;%02X&quot;,

(int) tempArr[0]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) tempArr[1]) + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) tempArr[2]) + &quot; &quot; + String.format(&quot;%02X&quot;,

(int) tempArr[3]));

//substitute

tempArr[0] = getSBoxValue(tempArr[0]);

tempArr[1] = getSBoxValue(tempArr[1]);

tempArr[2] = getSBoxValue(tempArr[2]);

tempArr[3] = getSBoxValue(tempArr[3]);

// System.out.println(&quot;After step3 sub- i: &quot; + i + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) tempArr[0]) + &quot; &quot; + String.format(&quot;%02X&quot;,

(int) tempArr[1]) + &quot; &quot; + String.format(&quot;%02X&quot;, (int) tempArr[2]) + &quot; &quot; +

String.format(&quot;%02X&quot;, (int) tempArr[3]));

//

// extended[0][i + 1] = (char) (extended[0][i - 3] ^

tempArr[0]);

// extended[1][i + 1] = (char) (extended[1][i - 3] ^

tempArr[1]);

// extended[2][i + 1] = (char) (extended[2][i - 3] ^

tempArr[2]);

// extended[3][i + 1] = (char) (extended[3][i - 3] ^

tempArr[3]);

extended[0][i + 1] = (char) tempArr[0];

extended[1][i + 1] = (char) tempArr[1];

extended[2][i + 1] = (char) tempArr[2];

extended[3][i + 1] = (char) tempArr[3];

// STEP 4

for (int c2 = 0; c2 &lt; 3; c2++) {

i++;

extended[0][i + 1] = (char) (extended[0][i - 3] ^

extended[0][i]);

extended[1][i + 1] = (char) (extended[1][i - 3] ^

extended[1][i]);

extended[2][i + 1] = (char) (extended[2][i - 3] ^

extended[2][i]);

extended[3][i + 1] = (char) (extended[3][i - 3] ^

extended[3][i]);

}

}

return extended;

}

public static void printMatrix(char[][] matrix, int row, int column) {

for (int c = 0; c &lt; row; c++) {

for (int c2 = 0; c2 &lt; column; c2++) {

System.out.print(&quot;&quot; + String.format(&quot;%02X&quot;, (int)

matrix[c][c2]) + &quot; &quot;);

}

System.out.println();

}

}

public static char getSBoxValue(char num) {

return sbox[num];

}

public static char getSBoxInvert(char num) {

return rsbox[num];

}

public static char getRconValue(char num) {

return Rcon[num];

}

}

**AESKeyGEN**

import java.util.Scanner;

public class Main {

static char sbox[] = {

//0 1 2 3 4 5 6 7 8 9 A B C D E F

0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0

0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1

0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3

0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4

0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5

0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6

0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7

0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8

0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9

0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A

0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B

0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C

0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D

0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E

0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16}; //F

static char rsbox[]

= {0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e, 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25, 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92, 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84, 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06, 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73, 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e, 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b, 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4, 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f, 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61, 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d};

static char Rcon[] = {

0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,

0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,

0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,

0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d,

0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,

0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,

0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,

0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,

0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,

0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,

0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,

0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,

0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,

0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,

0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,

0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,

0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f,

0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,

0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,

0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb};

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

char plainText[][] = new char[4][4]; // "00000000000000000000000000000000";

char cipherKey[][] = new char[4][8]; // "00000000000000000000000000000000";

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

String line3 = new Scanner(System.in).nextLine();

while(line3.length() < 32)

line3 += "0";

if(line3.length() > 32)

line3 = line3.substring(0, 32);

char temp3[] = line3.toCharArray();

int c4 = 0;

for (int c2 = 0; c2 < 32; c2 += 2) {

cipherKey[c4 % 4][c4++ / 4] = (char)Integer.parseInt("" + temp3[c2] + temp3[c2+1], 16);

}

System.out.println("Cipher key:");

printMatrix(cipherKey, 4, 4);

// print 2D array for debugging

// printMatrix(cipherKey, 4, 8);

// System.out.println("Rcon " + Integer.toHexString((int) getRconValue((char) 5)));

// System.out.println("Plain Text: ");

char extendedKey[][] = new char[4][60];

extendedKey = extendKey(cipherKey);

System.out.println("Extended key:");

printMatrix(extendedKey, 4, 40);

}

public static char[][] extendKey(char key[][]) {

char extended[][] = new char[4][60];

for (int c = 0; c < 4; c++) {

for (int c2 = 0; c2 < 8; c2++) {

extended[c][c2] = key[c][c2];

}

}

char c = 1;

for (int i = 7; i < 55; i++) {

// STEP 1

// rotate

// System.out.println("c: " + (int) c + " i: " + i + " rot 4 byte: ");

// System.out.println("" + String.format("%02X", (int) extended[0][i]) + " " + String.format("%02X", (int) extended[1][i]) + " " + String.format("%02X", (int) extended[2][i]) + " " + String.format("%02X", (int) extended[3][i]));

char tempArr[] = new char[4];

tempArr[0] = extended[1][i];

tempArr[1] = extended[2][i];

tempArr[2] = extended[3][i];

tempArr[3] = extended[0][i];

// System.out.println("After rotation" + String.format("%02X", (int) tempArr[0]) + " " + String.format("%02X", (int) tempArr[1]) + " " + String.format("%02X", (int) tempArr[2]) + " " + String.format("%02X", (int) tempArr[3]));

//substitute

tempArr[0] = getSBoxValue(tempArr[0]);

tempArr[1] = getSBoxValue(tempArr[1]);

tempArr[2] = getSBoxValue(tempArr[2]);

tempArr[3] = getSBoxValue(tempArr[3]);

// System.out.println("After sub" + String.format("%02X", (int) tempArr[0]) + " " + String.format("%02X", (int) tempArr[1]) + " " + String.format("%02X", (int) tempArr[2]) + " " + String.format("%02X", (int) tempArr[3]));

char rCon = getRconValue(c);

c = (char) (c + 1);

tempArr[0] ^= rCon;

extended[0][i + 1] = (char) (extended[0][i - 3] ^ tempArr[0]);

extended[1][i + 1] = (char) (extended[1][i - 3] ^ tempArr[1]);

extended[2][i + 1] = (char) (extended[2][i - 3] ^ tempArr[2]);

extended[3][i + 1] = (char) (extended[3][i - 3] ^ tempArr[3]);

//debugging point

// System.out.println("c: " + (int)c );

// System.out.println("new 4 byte: ");

// System.out.println("" + String.format("%02X", (int) extended[0][i + 1]) + " " + String.format("%02X", (int) extended[1][i + 1]) + " " + String.format("%02X", (int) extended[2][i + 1]) + " " + String.format("%02X", (int) extended[3][i + 1]));

// STEP 2

for (int c2 = 0; c2 < 3; c2++) {

i++;

extended[0][i + 1] = (char) (extended[0][i - 3] ^ extended[0][i]);

extended[1][i + 1] = (char) (extended[1][i - 3] ^ extended[1][i]);

extended[2][i + 1] = (char) (extended[2][i - 3] ^ extended[2][i]);

extended[3][i + 1] = (char) (extended[3][i - 3] ^ extended[3][i]);

}

//STEP 3

i++;

tempArr[0] = extended[0][i];

tempArr[1] = extended[1][i];

tempArr[2] = extended[2][i];

tempArr[3] = extended[3][i];

// System.out.println(" step3 begins" + String.format("%02X", (int) tempArr[0]) + " " + String.format("%02X", (int) tempArr[1]) + " " + String.format("%02X", (int) tempArr[2]) + " " + String.format("%02X", (int) tempArr[3]));

//substitute

tempArr[0] = getSBoxValue(tempArr[0]);

tempArr[1] = getSBoxValue(tempArr[1]);

tempArr[2] = getSBoxValue(tempArr[2]);

tempArr[3] = getSBoxValue(tempArr[3]);

// System.out.println("After step3 sub- i: " + i + " " + String.format("%02X", (int) tempArr[0]) + " " + String.format("%02X", (int) tempArr[1]) + " " + String.format("%02X", (int) tempArr[2]) + " " + String.format("%02X", (int) tempArr[3]));

//

// extended[0][i + 1] = (char) (extended[0][i - 3] ^ tempArr[0]);

// extended[1][i + 1] = (char) (extended[1][i - 3] ^ tempArr[1]);

// extended[2][i + 1] = (char) (extended[2][i - 3] ^ tempArr[2]);

// extended[3][i + 1] = (char) (extended[3][i - 3] ^ tempArr[3]);

extended[0][i + 1] = (char) tempArr[0];

extended[1][i + 1] = (char) tempArr[1];

extended[2][i + 1] = (char) tempArr[2];

extended[3][i + 1] = (char) tempArr[3];

// STEP 4

for (int c2 = 0; c2 < 3; c2++) {

i++;

extended[0][i + 1] = (char) (extended[0][i - 3] ^ extended[0][i]);

extended[1][i + 1] = (char) (extended[1][i - 3] ^ extended[1][i]);

extended[2][i + 1] = (char) (extended[2][i - 3] ^ extended[2][i]);

extended[3][i + 1] = (char) (extended[3][i - 3] ^ extended[3][i]);

}

}

return extended;

}

public static void printMatrix(char[][] matrix, int row, int column) {

for (int c = 0; c < row; c++) {

for (int c2 = 0; c2 < column; c2++) {

System.out.print("" + String.format("%02X", (int) matrix[c][c2]) + " ");

}

System.out.println();

}

}

public static char getSBoxValue(char num) {

return sbox[num];

}

public static char getSBoxInvert(char num) {

return rsbox[num];

}

public static char getRconValue(char num) {

return Rcon[num];

}

}

**DES**

import java.util.\*;

import java.io.\*;

class IncorrectInputFormatException extends Exception {

private static final long serialVersionUID = 1L;

private final String message;

IncorrectInputFormatException(String message) {

super();

this.message = message;

}

@Override

public String toString() {

return this.message;

}

}

class EncryptionException extends Exception {

private static final long serialVersionUID = 1L;

private int issueCode;

EncryptionException(int code) {

super();

this.issueCode = code;

}

@Override

public String toString() {

if (this.issueCode == 0){

return "Number of allowed rounds exceeded!";

}

else{

return "unknown";

}

}

}

class StringParser {

private String text;

private String keyString;

StringParser(String text, String keyString) {

this.text = text;

this.keyString = keyString;

}

private void convertHex (int x, ArrayList<Integer> seq) {

int[] binary = new int[4];

int k = 0;

while (x!=0) {

binary[k++] = x%2;

x = x/2;

}

for (int i=3; i>=0; i--) {

seq.add(binary[i]);

}

}

ArrayList<ArrayList<Integer>> evaluate () throws IncorrectInputFormatException { //returns list of blocks

if (this.keyString.length() != 16) {

throw new IncorrectInputFormatException("invalid key format.");

}

if (this.text.length() == 0) {

throw new IncorrectInputFormatException("empty input.");

}

for (int i=0; i<this.keyString.length(); i++) {

int encoding = (int) this.keyString.charAt(i);

if ((encoding>=48) && (encoding<=57)) {

continue;

}

else if ((encoding>=65) && (encoding<=70)) {

continue;

}

else if ((encoding>=97) && (encoding<=102)) {

continue;

}

else {

throw new IncorrectInputFormatException("invalid key format.");

}

}

ArrayList<ArrayList<Integer>> blocks = new ArrayList<>();

ArrayList<Integer> binaryInput = new ArrayList<>();

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

int encoding = (int) this.text.charAt(i);

if ((encoding>=48) && (encoding<=57)) {

this.convertHex(encoding-48, binaryInput);

}

else if ((encoding>=65) && (encoding<=70)) {

this.convertHex(encoding-55, binaryInput);

}

else if ((encoding>=97) && (encoding<=102)) {

this.convertHex(encoding-87, binaryInput);

}

else {

if (encoding != 32) {

throw new IncorrectInputFormatException("invalid input format.");

}

}

}

for (int i=0; i<binaryInput.size(); i+=64) {

ArrayList<Integer> block = new ArrayList<>();

for (int j=0; j<64; j++) {

if (i+j < binaryInput.size()) {

block.add (binaryInput.get(i+j));

}

}

int suffix = 64 - block.size();

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

block.add(0);

}

blocks.add(block);

}

return blocks;

}

ArrayList<ArrayList<Integer>> evaluateDecryption () throws IncorrectInputFormatException { //returns list of blocks

if (this.keyString.length() != 16) {

throw new IncorrectInputFormatException("invalid key format.");

}

if (this.text.length() == 0) {

throw new IncorrectInputFormatException("empty input.");

}

for (int i=0; i<this.keyString.length(); i++) {

int encoding = (int) this.keyString.charAt(i);

if ((encoding>=48) && (encoding<=57)) {

continue;

}

else if ((encoding>=65) && (encoding<=70)) {

continue;

}

else if ((encoding>=97) && (encoding<=102)) {

continue;

}

else {

throw new IncorrectInputFormatException("invalid key format.");

}

}

ArrayList<ArrayList<Integer>> blocks = new ArrayList<>();

ArrayList<Integer> binaryInput = new ArrayList<>();

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

int encoding = (int) this.text.charAt(i);

if ((encoding>=48) && (encoding<=57)) {

this.convertHex(encoding-48, binaryInput);

}

else if ((encoding>=65) && (encoding<=70)) {

this.convertHex(encoding-55, binaryInput);

}

else if ((encoding>=97) && (encoding<=102)) {

this.convertHex(encoding-87, binaryInput);

}

else {

if (encoding != 32) {

throw new IncorrectInputFormatException("invalid input format.");

}

}

}

for (int i=0; i<binaryInput.size(); i+=64) {

ArrayList<Integer> block = new ArrayList<>();

for (int j=0; j<64; j++) {

if (i+j < binaryInput.size()) {

block.add (binaryInput.get(i+j));

}

}

if (block.size() < 64) {

throw new IncorrectInputFormatException("missing cipher text.");

}

blocks.add(block);

}

return blocks;

}

}

class DESKeyOperations {

// PERMUTATION PC1

private ArrayList<Integer> leftKey;

private ArrayList<Integer> rightKey;

private int roundsCompleted = 0;

private static 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}

};

// PERMUTATION PC2

private static 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}

};

// LEFT SHIFTS

private int[] shifts = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

DESKeyOperations (String keyString) {

ArrayList<Integer> initialKey = new ArrayList<>();

this.leftKey = new ArrayList<>();

this.rightKey = new ArrayList<>();

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

initialKey.add (Character.getNumericValue(keyString.charAt(i)));

}

int count = 0;

int half = (pc1.length\*pc1[0].length)/2;

for (int i=0; i<pc1.length; i++) {

for (int j=0; j<pc1[i].length; j++) {

if (count < half) {

this.leftKey.add (initialKey.get(pc1[i][j]-1));

}

else {

this.rightKey.add (initialKey.get(pc1[i][j]-1));

}

count++;

}

}

}

private void leftShift (ArrayList<Integer> sequence) {

int first = sequence.get(0);

for (int i=1; i<sequence.size(); i++) {

sequence.set(i-1, sequence.get(i));

}

sequence.set(sequence.size()-1, first);

}

public ArrayList<Integer> getKey() throws EncryptionException {

if (this.roundsCompleted > this.shifts.length) {

throw new EncryptionException(0);

}

int keyShifts = this.shifts[this.roundsCompleted];

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

this.leftShift(this.leftKey);

this.leftShift(this.rightKey);

}

ArrayList<Integer> combined = new ArrayList<>();

ArrayList<Integer> permuted = new ArrayList<>();

for (Integer value: this.leftKey) {

combined.add (value);

}

for (Integer value: this.rightKey) {

combined.add (value);

}

for (int i=0; i<pc2.length; i++) {

for (int j=0; j<pc2[i].length; j++) {

permuted.add(combined.get(pc2[i][j]-1));

}

}

this.roundsCompleted++;

return permuted;

}

}

class Cipher {

// INITIAL PERMUTATION

private static 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}

};

// FINAL INVERSE PERMUTATION

private static int[][] ipinverse = {

{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}

};

// E BIT SELECTION TABLE

private static int[][] e = {

{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}

};

// PERMUTATION P

private static 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 BOXES

private static int[][][] S = {

{ // S1

{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}

},

{ // S2

{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}

},

{ // S3

{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}

},

{ // S4

{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}

},

{ // S5

{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}

},

{ // S6

{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}

},

{ // S7

{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}

},

{ // S8

{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}

}

};

private ArrayList<ArrayList<Integer>> keys;

Cipher (String keyString) throws EncryptionException {

this.keys = new ArrayList<>();

DESKeyOperations generator = new DESKeyOperations(keyString);

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

this.keys.add (generator.getKey());

}

System.out.println ("Round keys: ");

for (int i=0; i<this.keys.size(); i++) {

System.out.print ("Round " + i + ": ");

for (int j=0; j<this.keys.get(i).size(); j++) {

System.out.print(this.keys.get(i).get(j));

}

System.out.println();

}

}

private int[] sBox (int[] expanded) {

int[] contracted = new int[32];

int filled = 0;

int count = 0;

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

int row = expanded[i]\*2 + expanded[i+5];

int col = expanded[i+1]\*8 + expanded[i+2]\*4 + expanded[i+3]\*2 + expanded[i+4];

int val = S[count][row][col];

int[] fourBits = new int[4];

int k = 0;

while (val != 0) {

fourBits[k++] = val%2;

val = val/2;

}

for (int j=3; j>=0; j--) {

contracted[filled] = fourBits[j];

filled++;

}

count++;

}

return contracted;

}

private ArrayList<Integer> round (ArrayList<Integer> input, ArrayList<Integer> roundKey) {

ArrayList<Integer> output = new ArrayList<>();

int[] left = new int[32];

int[] right = new int[32];

for (int i=0; i<32; i++) {

left[i] = input.get(i);

right[i] = input.get(i+32);

output.add (right[i]);

}

int[] expansionPermutation = new int[48];

int k = 0;

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

for (int j=0; j<e[i].length; j++) {

expansionPermutation[k] = right[e[i][j]-1];

k++;

}

}

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

expansionPermutation[i] = expansionPermutation[i]^roundKey.get(i);

}

int[] postS = this.sBox(expansionPermutation);

int[] postSPermutation = new int[32];

k=0;

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

for (int j=0; j<p[i].length; j++) {

postSPermutation[k] = postS[p[i][j]-1];

k++;

}

}

for (int i=0; i<32; i++) {

output.add (left[i]^postSPermutation[i]);

}

return output;

}

private ArrayList<Integer> encryptBlock (ArrayList<Integer> block) {

//Input permutation

ArrayList<Integer> input = new ArrayList<>();

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

for (int j=0; j<ip[i].length; j++) {

input.add (block.get(ip[i][j]-1));

}

}

//round operations

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

ArrayList<Integer> nextInput = this.round(input, this.keys.get(i));

input = nextInput;

}

//32 bit swap

int half = input.size()/2;

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

int temp = input.get (i);

input.set (i, input.get(i+half));

input.set (i+half, temp);

}

//ipinverse

ArrayList<Integer> cipher = new ArrayList<>();

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

for (int j=0; j<ipinverse[i].length; j++) {

cipher.add (input.get(ipinverse[i][j]-1));

}

}

return cipher;

}

private ArrayList<Integer> decryptBlock (ArrayList<Integer> block) {

//Input permutation

ArrayList<Integer> input = new ArrayList<>();

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

for (int j=0; j<ip[i].length; j++) {

input.add (block.get(ip[i][j]-1));

}

}

//round operations

for (int i=15; i>=0; i--) {

ArrayList<Integer> nextInput = this.round(input, this.keys.get(i));

input = nextInput;

}

//32 bit swap

int half = input.size()/2;

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

int temp = input.get (i);

input.set (i, input.get(i+half));

input.set (i+half, temp);

}

//ipinverse

ArrayList<Integer> plain = new ArrayList<>();

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

for (int j=0; j<ipinverse[i].length; j++) {

plain.add (input.get(ipinverse[i][j]-1));

}

}

return plain;

}

private String convertToHex (ArrayList<ArrayList<Integer>> ans) {

StringBuilder hexCipher = new StringBuilder();

for (ArrayList<Integer> blockCipher : ans){

for (int i=0; i<blockCipher.size(); i+=4){

int val = blockCipher.get(i)\*8 + blockCipher.get(i+1)\*4 + blockCipher.get(i+2)\*2 + blockCipher.get(i+3);

//conversion to Hex

if (val < 10) {

hexCipher.append(val);

}

else{

int ten = (int) 'A';

int newVal = ten + (val-10);

hexCipher.append((char) newVal);

}

}

// hexCipher.append(' ');

}

return hexCipher.toString();

}

String encrypt (ArrayList<ArrayList<Integer>> input) {

ArrayList<ArrayList<Integer>> ciphers = new ArrayList<>();

for (ArrayList<Integer> block: input) {

ciphers.add(this.encryptBlock(block));

}

return this.convertToHex(ciphers);

}

String decrypt (ArrayList<ArrayList<Integer>> input) {

ArrayList<ArrayList<Integer>> plainTexts = new ArrayList<>();

for (ArrayList<Integer> block: input) {

plainTexts.add(this.decryptBlock(block));

}

return this.convertToHex(plainTexts);

}

}

class Main {

private static String ASCIItoHEX(String ascii)

{

StringBuilder hex = new StringBuilder();

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

char ch = ascii.charAt(i);

int in = (int)ch;

String part = Integer.toHexString(in);

hex.append (part);

}

return hex.toString();

}

private static String hexToASCII(String hex)

{

StringBuilder ascii = new StringBuilder();

for (int i = 0; i < hex.length(); i += 2) {

String part = hex.substring(i, i + 2);

char ch = (char)Integer.parseInt(part, 16);

ascii.append(ch);

}

return ascii.toString();

}

private static 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;

}

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

BufferedReader scanner = new BufferedReader(new InputStreamReader(System.in));

String c="";

String keyString="" ;

while (true) {

try {

System.out.println ("\n" + "1.Encrypt\n2.Decrypt\n3.Exit\nEnter your choice : ");

int option = Integer.parseInt(scanner.readLine());

if ((option>3) || (option<=0)) {

System.out.println("Enter valid input!");

continue;

}

if (option == 3) {

break;

}

if (option == 1) {

String temp;

System.out.println ("Input your plaintext : ");

temp = scanner.readLine();

String inputString;

inputString = ASCIItoHEX(temp);

System.out.println ("Input your key : ");

keyString = scanner.readLine();

if (keyString.length() == 0) {

keyString = "133457799BBCDFF1";

}

System.out.println ("Key used: " + keyString);

StringParser checker = new StringParser(inputString, keyString);

ArrayList<ArrayList<Integer>> inputBlocks = checker.evaluate(); //Returns list of 64 bit blocks

String keyStringBinary = hextoBin(keyString);

Cipher cipher = new Cipher(keyStringBinary);

System.out.println ("Cipher: " + cipher.encrypt(inputBlocks));

c= cipher.encrypt(inputBlocks);

}

else{

String inputString;

inputString = c;

if (keyString.length() == 0) {

keyString = "133457799BBCDFF1";

}

System.out.println ("Key used: " + keyString);

StringParser checker = new StringParser(inputString, keyString);

ArrayList<ArrayList<Integer>> inputBlocks = checker.evaluateDecryption(); //Returns list of 64 bit blocks

String keyStringBinary;

keyStringBinary = hextoBin(keyString);

Cipher cipher = new Cipher(keyStringBinary);

System.out.println ("Plain text: " + hexToASCII(cipher.decrypt(inputBlocks)));

}

} catch (IncorrectInputFormatException|NumberFormatException|EncryptionException e) {

System.out.println("Error " + e);

}

}

}

}

import java.util.\*;

public class Main {

// 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 };

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 };

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

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

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;

}

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;

}

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;

}

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;

}

// Returns modulo inverse of a

// with respect to m using extended

// Euclid Algorithm. Refer below post for details:

// https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

public static void main(String args[]){

Main object = new Main();

Scanner in = new Scanner(System.in);

System.out.println("Enter a key ");

String key = in.next();

String[] result = object.getKeys(key);

for(String s: result){

System.out.print(s+" ");

}

}

}

Ip: hexadec key(16)

Op: hexa key for 4 lines

**RSA**

import java.util.List;

import java.util.ArrayList;

import java.util.Scanner;

import java.math.BigInteger;

import java.util.Random;

class Main {

String byteToString(byte [] sample) {

String result = "";

for (byte b : sample) {

result += Byte.toString(b);

}

return result;

}

byte[] encrypt(String plainText, BigInteger E, BigInteger N){

byte [] toEncrypt = plainText.getBytes();

BigInteger m = new BigInteger(toEncrypt);

BigInteger cipher;

cipher = m.modPow(E, N);

System.out.println("The Cipher is: " + cipher);

return cipher.toByteArray();

}

byte[] decrypt (byte[] encrypted, BigInteger D, BigInteger N) {

BigInteger m = new BigInteger(encrypted);

BigInteger plain;

plain = m.modPow(D, N);

return plain.toByteArray();

}

public static void main(String[] args) {

Random random = new Random();

RSA rsa = new RSA();

Scanner sc = new Scanner(System.in);

BigInteger P, Q, N, Phi, E, D;

byte [] encryptedBytes = new byte[10000];

byte [] decryptedBytes = new byte[10000];

P = BigInteger.probablePrime(1024, random);

Q = BigInteger.probablePrime(1024, random);

while (Q.compareTo(P) == 0) {

Q = BigInteger.probablePrime(1024, random);

}

N = P.multiply(Q);

Phi = (P.subtract(BigInteger.ONE)).multiply(Q.subtract(BigInteger.ONE));

E = BigInteger.probablePrime(5, random);

while (Phi.gcd(E).compareTo(BigInteger.ONE) > 0 && E.compareTo(Phi) < 0) {

E.add(BigInteger.ONE);

}

D = E.modInverse(Phi);

while (D.compareTo(Phi) >= 0) {

D = E.modInverse(Phi);

}

String plainText = "";

System.out.println("Random Key Generated!");

System.out.println("Public Key: (\nN: " + N + ", \nE: " + E + "\n)");

System.out.println("Private Key: (\nD: " + D + ",\nP: " + P + ",\nQ: " + Q + "\n)");

System.out.println("\nMENU: ");

System.out.println("1. Get Plain Text");

System.out.println("2. Encrypt");

System.out.println("3. Decrypt");

while (true) {

char ch;

System.out.print("Enter the choice: ");

ch = sc.next().charAt(0);

if (ch == '1') {

// get input

System.out.print("Enter the plain Text: ");

sc.nextLine();

plainText = sc.nextLine();

}

else if ( ch == '2') {

// encrypt

if (plainText.equals("")) {

System.out.println("Enter the plain text first.");

continue;

}

else {

encryptedBytes = rsa.encrypt(plainText, E, N);

}

}

else if (ch == '3') {

decryptedBytes = rsa.decrypt(encryptedBytes, D, N);

System.out.println("The decrypted String is: " + new

String(decryptedBytes));

}

else {

break;

}

}

}

}

**DIFFIE**

import java.math.BigInteger;

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int P = BigInteger.probablePrime(30, new Random()).intValue();

int G = primitiveRoot(P);

System.out.println("P = "+P+" G = "+G);

BigInteger g = new BigInteger(""+G);

BigInteger p = new BigInteger(""+P);

System.out.println("Enter A's secret key ");

BigInteger xa = new BigInteger(in.next());

BigInteger ya = g.modPow(xa, p);

System.out.println("A's public key = " + ya);

System.out.println("Enter B's secret key ");

BigInteger xb = new BigInteger(in.next());

BigInteger yb = g.modPow(xb, p);

System.out.println("B's public key = " + yb);

BigInteger A\_sh = yb.modPow(xa, p);

BigInteger B\_sh = ya.modPow(xb, p);

System.out.println("A's shared secret = "+A\_sh);

System.out.println("B's shared secret = "+B\_sh);

in.close();

}

public static int primitiveRoot(int p)

{

if (!(new BigInteger(""+p).isProbablePrime(1))) return -1;

int phi = p-1;

Set<Integer> factors = primeFactors(phi);

ArrayList<Integer> ans = new ArrayList<>();

for (int i=2;i<=phi;i++)

{

boolean flag = false;

for(int x : factors)

{

if (power(i,phi/x,p) == 1)

{

flag = true;

break;

}

}

if (!flag) return i;

// ans.add(i);

}

if (ans.size() == 0) return -1;

int ind = (int)(Math.random()%ans.size());

return ans.get(ind);

// return -1;

}

public static Set<Integer> primeFactors(int p)

{

Set<Integer> facts = new HashSet<>();

while (p%2==0){

facts.add(2);

p/=2;

}

for(int i=3;i<=(int)Math.sqrt(p);i+=2){

while(p%i==0)

{

p/=i;

facts.add(i);

}

}

if (p>2)

facts.add(p);

return facts;

}

public static int power(int a,int b,int mod)

{

if (b==0)

return 1;

if (b==1)

return a%mod;

int temp = power(a,b/2,mod);

temp = (temp\*temp);

if (b%2!=0)

temp\*= a;

return temp%mod;

}

static boolean millerTest(int d, int n) {

int a = 2 + (int)(Math.random() % (n - 4));

int x = power(a, d, n);

if (x == 1 || x == n - 1)

return true;

while (d != n - 1) {

x = (x \* x) % n;

d \*= 2;

if (x == 1)

return false;

if (x == n - 1)

return true;

}

return false;

}

static boolean isPrime(int n) {

if (n <= 1 || n == 4)

return false;

if (n == 3)

return true;

int d = n - 1;

int k=n-2;

while (d % 2 == 0)

{

d /= 2;

// k++;

}

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

if (!millerTest(d, n))

return false;

return true;

}

}

**DSA**

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.Signature;

import java.util.Scanner;

import java.io.UnsupportedEncodingException;

import java.util.\*;

import java.security.\*;

public class Main {

public static void main(String[] args) throws NoSuchAlgorithmException, InvalidKeyException, SignatureException, UnsupportedEncodingException {

Scanner sc = new Scanner(System.in);

System.out.print("Text : ");

String msg = sc.nextLine();

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

keys.initialize(1024);

KeyPair key = keys.generateKeyPair();

PrivateKey pk = key.getPrivate();

PublicKey puk = key.getPublic();

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

sign.initSign(pk);

byte[] text = msg.getBytes();

sign.update(text);

byte[] signature = sign.sign();

sign.initVerify(puk);

System.out.print("Data : ");

String data = sc.nextLine();

sign.update(data.getBytes());

if(sign.verify(signature))

System.out.println("Signature Verified !");

else

System.out.println("Signature Invalid !");

}

}

**MD5\_WITH LIBRARIES**

**MD5\_WITH\_LIBRARY**

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;

public class Main {

public static String getMd5(String input)

{

try {

MessageDigest md = MessageDigest.getInstance("MD5") ; //;change to SHA-1 for sha

byte[] messageDigest = md.digest(input.getBytes());

// Convert byte array into signum representation

BigInteger no = new BigInteger(1, messageDigest);

// Convert message digest into hex value

String hashtext = no.toString(16);

while (hashtext.length() < 32) {

hashtext = "0" + hashtext;

}

return hashtext;

}

// For specifying wrong message digest algorithms

catch (NoSuchAlgorithmException e) {

throw new RuntimeException(e);

}

}

// Driver code

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

{

System.out.println("Enter the text:");

Scanner s=new Scanner(System.in);

String txt\_ip=s.nextLine();

System.out.println("Your HashCode Generated by MD5 for "+txt\_ip+ "is: " + getMd5(txt\_ip));

}

}

**SHA\_LIB**

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.\*;

public class Main {

public static String encryptThisString(String input) {

try {

MessageDigest md = MessageDigest.getInstance("SHA-1");

byte[] messageDigest = md.digest(input.getBytes());

BigInteger no = new BigInteger(1, messageDigest);

String hashtext = no.toString(16);

while (hashtext.length() < 32) {

hashtext = "0" + hashtext;

}

return hashtext;

} catch (NoSuchAlgorithmException e) {

throw new RuntimeException(e);

}

}

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

System.out.println("----SHA-1----");

System.out.println("Insert a word a phrase to be hashed");

Scanner sc = new Scanner(System.in);

String word = sc.nextLine();

System.out.println("Plain Text: " + word);

System.out.println("Encrypted Text: " + encryptThisString(word));

}

}

**SHA:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Main {

private List<byte[]> inputDataList = new ArrayList<byte[]>();

// Bitwise rotate a 32-bit number to the left

private static int rol(int num, int cnt) {

return (num << cnt) | (num >>> (32 - cnt));

}

public byte[] digest(byte[] x) {

// Append padding bits and the length

int[] blks = new int[(((x.length + 8) >> 6) + 1) \* 16];

int i;

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

blks[i >> 2] |= x[i] << (24 - (i % 4) \* 8);

}

blks[i >> 2] |= 0x80 << (24 - (i % 4) \* 8);

blks[blks.length - 1] = x.length \* 8;

// calculate 160 bit SHA1 hash of the sequence of blocks

int[] w = new int[80];

int a = 1732584193;

int b = -271733879;

int c = -1732584194;

int d = 271733878;

int e = -1009589776;

for(i = 0; i < blks.length; i += 16) {

int olda = a;

int oldb = b;

int oldc = c;

int oldd = d;

int olde = e;

for(int j = 0; j < 80; j++) {

w[j] = (j < 16) ? blks[i + j] :

( rol(w[j-3] ^ w[j-8] ^ w[j-14] ^ w[j-16], 1) );

int t = rol(a, 5) + e + w[j] +

( (j < 20) ? 1518500249 + ((b & c) | ((~b) & d))

: (j < 40) ? 1859775393 + (b ^ c ^ d)

: (j < 60) ? -1894007588 + ((b & c) | (b & d) | (c & d))

: -899497514 + (b ^ c ^ d) );

e = d;

d = c;

c = rol(b, 30);

b = a;

a = t;

}

a = a + olda;

b = b + oldb;

c = c + oldc;

d = d + oldd;

e = e + olde;

}

// Convert result to a byte array

byte[] digest = new byte[20];

fill(a, digest, 0);

fill(b, digest, 4);

fill(c, digest, 8);

fill(d, digest, 12);

fill(e, digest, 16);

return digest;

}

private void fill(int value, byte[] arr, int off) {

arr[off + 0] = (byte) ((value >> 24) & 0xff);

arr[off + 1] = (byte) ((value >> 16) & 0xff);

arr[off + 2] = (byte) ((value >> 8) & 0xff);

arr[off + 3] = (byte) ((value >> 0) & 0xff);

}

private static final char[] HEX\_ARRAY = "0123456789ABCDEF".toCharArray();

public static String bytesToHex(byte[] bytes) {

char[] hexChars = new char[bytes.length \* 2];

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

int v = bytes[j] & 0xFF;

hexChars[j \* 2] = HEX\_ARRAY[v >>> 4];

hexChars[j \* 2 + 1] = HEX\_ARRAY[v & 0x0F];

}

return new String(hexChars);

}

public static void main(String[] args) {

Main Main = new Main();

System.out.println("Enter the string: ");

Scanner scanner = new Scanner(System.in);

String p = scanner.nextLine();

byte[] digest = Main.digest(p.getBytes());

System.out.println("The SHA1 hash is "+bytesToHex(digest));

}

}

**MILLER**

import java.io.\*;

import java.math.\*;

import java.util.\*;

class Main {

static int power(int x, int y, int p) {

int res = 1;

x = x % p;

while (y > 0) {

if ((y & 1) == 1)

res = (res \* x) % p;

y = y >> 1; // y = y/2

x = (x \* x) % p;

}

return res;

}

static boolean millerTest(int d, int n) {

int a = 2 + (int) (Math.random() % (n - 4));

int x = power(a, d, n);

if (x == 1 || x == n - 1)

return true;

while (d != n - 1) {

x = (x \* x) % n;

d \*= 2;

if (x == 1)

return false;

if (x == n - 1)

return true;

}

return false;

}

static boolean isPrime(int n, int k) {

if (n <= 1 || n == 4)

return false;

if (n <= 3)

return true;

int d = n - 1;

while (d % 2 == 0)

d /= 2;

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

if (!millerTest(d, n))

return false;

return true;

}

public static void main(String args[]) {

int k = 4;

System.out.println("Enter the number : ");

Scanner scan = new Scanner(System.in);

int num = scan.nextInt();

System.out.println(isPrime(num, k));

// System.out.println("All primes smaller " + "than 100: ");

// for (int n = 1; n < 100; n++)

// if (isPrime(n, k))

// System.out.print(n + " ");

}

}

**INVERSE& EUCLID**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

System.out.println("----Mod Inverse----");

Scanner scan = new Scanner(System.in);

System.out.println("Enter the value : ");

int a = scan.nextInt();

System.out.println("Enter the mod : ");

int m = scan.nextInt();

a = a % m;

for (int i = 1; i < m; i++) {

if ((a \* i) % m == 1) {

System.out.println("FOUND : " + i);

}

}

}

public static int modInverseExtendedEuclid(int a, int m) {

int m0 = m;

int y = 0;

int x = 1;

if (m == 1)

return 0;

while (a > 1) {

int q = a / m;

int t = m;

m = a % m;

a = t;

t = y;

y = x - q \* y;

x = t;

}

if (x < 0)

x += m0;

return x;

}

}

**CRT**

import java.util.\*;

class Main {

// Returns modulo inverse of a

// with respect to m using extended

// Euclid Algorithm. Refer below post for details:

// https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/

static int inv(int a, int m)

{

int m0 = m, t, q;

int x0 = 0, x1 = 1;

if (m == 1)

return 0;

// Apply extended Euclid Algorithm

while (a > 1)

{

// q is quotient

q = a / m;

t = m;

// m is remainder now, process

// same as euclid's algo

m = a % m;a = t;

t = x0;

x0 = x1 - q \* x0;

x1 = t;

}

// Make x1 positive

if (x1 < 0)

x1 += m0;

return x1;

}

// k is size of num[] and rem[].

// Returns the smallest number

// x such that:

// x % num[0] = rem[0],

// x % num[1] = rem[1],

// ..................

// x % num[k-2] = rem[k-1]

// Assumption: Numbers in num[] are pairwise

// coprime (gcd for every pair is 1)

static int findMinX(int num[], int rem[], int k)

{

// Compute product of all numbers

int prod = 1;

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

prod \*= num[i];

// Initialize result

int result = 0;

// Apply above formula

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

{

int pp = prod / num[i];

result += rem[i] \* inv(pp, num[i]) \* pp;

}

return result % prod;

}

// Driver method

public static void main(String args[])

{

System.out.println("Enter the number of remainders : ");

Scanner scan = new Scanner(System.in);

int len = scan.nextInt();

int[] num = new int[len];

int[] rem = new int[len];

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

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

rem[i] = scan.nextInt();

}

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

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

num[i] = scan.nextInt();

}

System.out.println("x is " +findMinX(num, rem, len));

}

}