|  |  |
| --- | --- |
| **ASSIGNMENT** | |
| **Course** | CRYPTOGRAPHY & DATA SECURITY |
| **Due Date** | Thursday, December 24, 2020 |
| **Submitted By:** | M. Iftikhar Uddin Khan Sami (022-16-113275) |
| Omer Ashraf Khan (022-16-113393) |
| KamranAhmed (022-16-113392) |
| INDEX | |
| Contents  [Github Link: 2](#_Toc59716021)  [Class: CipherFramework 2](#_Toc59716022)  [Interface: Cipher\_Interface 5](#_Toc59716023)  [Class: Hill\_cipher implements Cipher\_Interface 5](#_Toc59716024)  [Class: Playfair\_cipher implements Cipher\_Interface 14](#_Toc59716025) | |

# Github Link:

<https://github.com/miks98/CryptographyAssignment>

# Class: CipherFramework

package cipher.framework;

// @author Omer & MIKS

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

/\*\*

\*

\* @author Omer and MIKS

\*/

public class CipherFramework {

/\*\*

\* @param args the command line arguments

\*/

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

// TODO code application logic here

Scanner sc = new Scanner(System.in);

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

System.out.println("Please Select the Cipher technique you want to run");

System.out.println("Press 1 for PlayFair Cipher");

System.out.println("Press 2 for Hill Cipher");

System.out.println("\n\n Enter your option: ");

int val = sc.nextInt();

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

if (val == 1) {

Playfair\_cipher pf = new Playfair\_cipher();

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

System.out.println("Enter Keyword");

String pk =sc.next();

pf.setKey(pk);

pf.generate\_key();

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

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

String text\_pe =sc.next();

String pe =pf.encrypt(text\_pe);

System.out.println("The Encrypted Text is "+pe);

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

System.out.println("Enter Text to Decrypt");

String text\_pd =sc.next();

String pd = pf.decrypt(text\_pd);

System.out.println("The Decrypted Text is "+pd);

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

}

else if (val == 2) {

Hill\_cipher obj = new Hill\_cipher() {};

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

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

String line = in.readLine();

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

String key = in.readLine();

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

if (sq != (long) sq)

System.out

.println("Invalid key length!!! Does not form a square matrix...");

else

{

int s = (int) sq;

if (obj.check(key, s))

{

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

obj.divide(line, s);

obj.cofact(obj.keymatrix, s);

}

}

}

else{

System.out.println("Error Encountered");

}

}

}

# Interface: Cipher\_Interface

package cipher.framework;

// @author Omer & MIKS

public interface Cipher\_Interface {

public void generate\_key();

public String encrypt(String Source);

public String decrypt(String Code);

public void cryptoanalysis();

}

# Class: Hill\_cipher implements Cipher\_Interface

package cipher.framework;

// @author Omer & MIKS

public abstract class Hill\_cipher implements Cipher\_Interface {

int keymatrix[][];

int linematrix[];

int resultmatrix[];

public Hill\_cipher() {

}

public void divide(String temp, int s)

{

while (temp.length() > s)

{

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

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

perform(sub);

}

if (temp.length() == s)

perform(temp);

else if (temp.length() < s)

{

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

temp = temp + 'x';

perform(temp);

}

}

public void perform(String line)

{

linetomatrix(line);

linemultiplykey(line.length());

result(line.length());

}

public void keytomatrix(String key, int len)

{

keymatrix = new int[len][len];

int c = 0;

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

{

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

{

keymatrix[i][j] = ((int) key.charAt(c)) - 97;

c++;

}

}

}

public void linetomatrix(String line)

{

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

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

{

linematrix[i] = ((int) line.charAt(i)) - 97;

}

}

public void linemultiplykey(int len)

{

resultmatrix = new int[len];

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

{

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

{

resultmatrix[i] += keymatrix[i][j] \* linematrix[j];

}

resultmatrix[i] %= 26;

}

}

public void result(int len)

{

String result = "";

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

{

result += (char) (resultmatrix[i] + 97);

}

System.out.print(result);

}

public boolean check(String key, int len)

{

keytomatrix(key, len);

int d = determinant(keymatrix, len);

d = d % 26;

if (d == 0)

{

System.out

.println("Invalid key!!! Key is not invertible because determinant=0...");

return false;

}

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

{

System.out

.println("Invalid key!!! Key is not invertible because determinant has common factor with 26...");

return false;

}

else

{

return true;

}

}

public int determinant(int A[][], int N)

{

int res;

if (N == 1)

res = A[0][0];

else if (N == 2)

{

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

}

else

{

res = 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++;

}

}

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

\* determinant(m, N - 1);

}

}

return res;

}

public void cofact(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) \* determinant(b, f - 1);

}

}

trans(fac, f);

}

void trans(int fac[][], int r)

{

int i, j;

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

b = new int[r][r];

inv = new int[r][r];

int d = determinant(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:");

matrixtoinvkey(inv, r);

}

public 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);

}

public void matrixtoinvkey(int inv[][], int n)

{

String invkey = "";

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

{

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

{

invkey += (char) (inv[i][j] + 97);

}

}

System.out.print(invkey);

}

@Override

public void generate\_key() { }

@Override

public String encrypt(String Source) { return null; }

@Override

public String decrypt(String Code) { return null; }

@Override

public void cryptoanalysis() { }

}

# Class: Playfair\_cipher implements Cipher\_Interface

package cipher.framework;

// @author Omer & MIKS

public class Playfair\_cipher implements Cipher\_Interface {

private char matrix\_arr[][] = new char[5][5];

String KeyWord = new String();

String Key = new String();

String Original = new String();

public Playfair\_cipher() { }

@Override

public void generate\_key() {

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();

}

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;

}

private void matrix(){

int counter = 0;

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

{

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

{

matrix\_arr[i][j] = Key.charAt(counter);

System.out.print(matrix\_arr[i][j] + " ");

counter++;

}

System.out.println();

}

}

private String format(String old\_text){

int i = 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

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

}

len = text.length();

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

{

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

{

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

}

}

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);

counter = counter + 2;

}

return x;

}

public int[] GetDiminsions(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 (matrix\_arr[i][j] == letter)

{

key[0] = i;

key[1] = j;

break;

}

}

}

return key;

}

public String encrypt(String Source) {

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 = GetDiminsions(one);

part2 = GetDiminsions(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 + matrix\_arr[part1[0]][part1[1]]

+ matrix\_arr[part2[0]][part2[1]];

}

return Code;

}

public String decrypt(String Code) {

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 = GetDiminsions(one);

part2 = GetDiminsions(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 + matrix\_arr[part1[0]][part1[1]]

+ matrix\_arr[part2[0]][part2[1]];

}

return Original;

}

@Override

public void cryptoanalysis() {

}

}