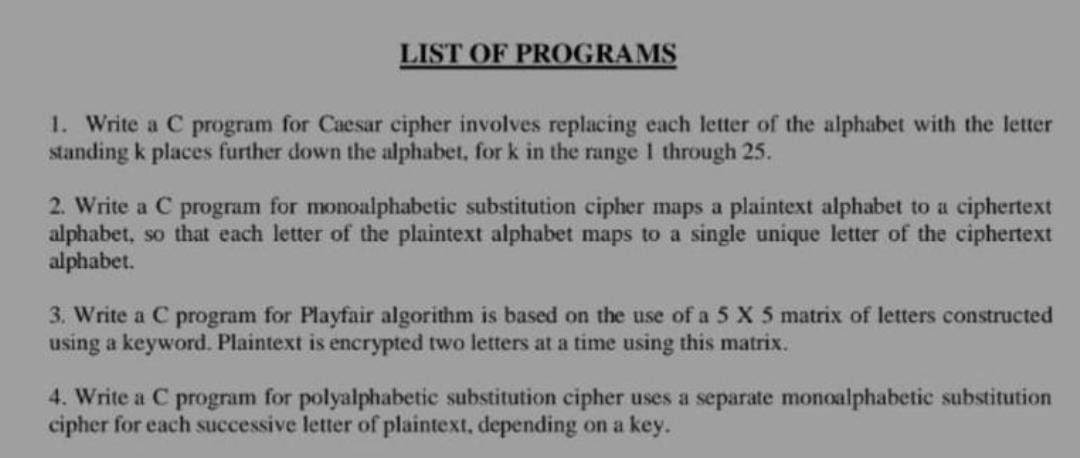
CSA5173-CRYPTOGRAPHY AND NETWORK SECURITY WITH CRYPTOGRAPHY ATTACKS

NAME:ASWINI L

REG NO:192124096

DATE:28\04\23



1.def caesar\_cipher(message, shift):

cipher = ''

for char in message:

if char.isalpha():

char\_code = ord(char) - shift

if char.isupper():

if char\_code > ord('Z'):

char\_code -= 26

elif char\_code < ord('A'):

char\_code += 26

elif char.islower():

if char\_code > ord('z'):

char\_code -= 26

elif char\_code < ord('a'):

char\_code += 26

cipher += chr(char\_code)

else:

cipher += char

return cipher

message = input("Enter the string to be decrypted")

shift = 3

decrypted\_message = caesar\_cipher(message, shift)

print(decrypted\_message)

2. cipher\_map = {'a': 'q', 'b': 'w', 'c': 'e', 'd': 'r', 'e': 't',

'f': 'y', 'g': 'u', 'h': 'i', 'i': 'o', 'j': 'p',

'k': 'a', 'l': 's', 'm': 'd', 'n': 'f', 'o': 'g',

'p': 'h', 'q': 'j', 'r': 'k', 's': 'l', 't': 'z',

'u': 'x', 'v': 'c', 'w': 'v', 'x': 'b', 'y': 'n', 'z': 'm'}

# Define the reverse mapping for decryption

decipher\_map = {v: k for k, v in cipher\_map.items()}

def encrypt(message):

"""Encrypts the given message using the cipher map."""

# Convert message to lowercase

message = message.lower()

# Initialize the encrypted message

encrypted\_message = ''

# Encrypt each character in the message

for char in message:

if char in string.ascii\_lowercase:

encrypted\_char = cipher\_map[char]

else:

encrypted\_char = char

encrypted\_message += encrypted\_char

return encrypted\_message

message = input("Enter the text:")

encrypted\_message = encrypt(message)

print(encrypted\_message)

3. def toLowerCase(text):

return text.lower()

def removeSpaces(text):

newText = ""

for i in text:

if i == " ":

continue

else:

newText = newText + i

return newText

def Diagraph(text):

Diagraph = []

group = 0

for i in range(2, len(text), 2):

Diagraph.append(text[group:i])

group = i

Diagraph.append(text[group:])

return Diagraph

def FillerLetter(text):

k = len(text)

if k % 2 == 0:

for i in range(0, k, 2):

if text[i] == text[i+1]:

new\_word = text[0:i+1] + str('x') + text[i+1:]

new\_word = FillerLetter(new\_word)

break

else:

new\_word = text

else:

for i in range(0, k-1, 2):

if text[i] == text[i+1]:

new\_word = text[0:i+1] + str('x') + text[i+1:]

new\_word = FillerLetter(new\_word)

break

else:

new\_word = text

return new\_word

list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',

'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

def generateKeyTable(word, list1):

key\_letters = []

for i in word:

if i not in key\_letters:

key\_letters.append(i)

compElements = []

for i in key\_letters:

if i not in compElements:

compElements.append(i)

for i in list1:

if i not in compElements:

compElements.append(i)

matrix = []

while compElements != []:

matrix.append(compElements[:5])

compElements = compElements[5:]

return matrix

def search(mat, element):

for i in range(5):

for j in range(5):

if(mat[i][j] == element):

return i, j

def encrypt\_RowRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

if e1c == 4:

char1 = matr[e1r][0]

else:

char1 = matr[e1r][e1c+1]

char2 = ''

if e2c == 4:

char2 = matr[e2r][0]

else:

char2 = matr[e2r][e2c+1]

return char1, char2

def encrypt\_ColumnRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

if e1r == 4:

char1 = matr[0][e1c]

else:

char1 = matr[e1r+1][e1c]

char2 = ''

if e2r == 4:

char2 = matr[0][e2c]

else:

char2 = matr[e2r+1][e2c]

return char1, char2

def encrypt\_RectangleRule(matr, e1r, e1c, e2r, e2c):

char1 = ''

char1 = matr[e1r][e2c]

char2 = ''

char2 = matr[e2r][e1c]

return char1, char2

def encryptByPlayfairCipher(Matrix, plainList):

CipherText = []

for i in range(0, len(plainList)):

c1 = 0

c2 = 0

ele1\_x, ele1\_y = search(Matrix, plainList[i][0])

ele2\_x, ele2\_y = search(Matrix, plainList[i][1])

if ele1\_x == ele2\_x:

c1, c2 = encrypt\_RowRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

# Get 2 letter cipherText

elif ele1\_y == ele2\_y:

c1, c2 = encrypt\_ColumnRule(Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

else:

c1, c2 = encrypt\_RectangleRule(

Matrix, ele1\_x, ele1\_y, ele2\_x, ele2\_y)

cipher = c1 + c2

CipherText.append(cipher)

return CipherText

text\_Plain = 'instruments'

text\_Plain = removeSpaces(toLowerCase(text\_Plain))

PlainTextList = Diagraph(FillerLetter(text\_Plain))

if len(PlainTextList[-1]) != 2:

PlainTextList[-1] = PlainTextList[-1]+'z'

key = "Monarchy"

print("Key text:", key)

key = toLowerCase(key)

Matrix = generateKeyTable(key, list1)

print("Plain Text:", text\_Plain)

CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

CipherText = ""

for i in CipherList:

CipherText += i

print("CipherText:", CipherText)

Output:

Key text: Monarchy

Plain Text: instruments

CipherText: gatlmzclrqtx

**DATE:03.05.2023**

**DAY:WEDNESDAY**

5. Write a C program for generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter C: C = E([a, b], p) = (ap + b) mod 26 A basic requirement of any encryption algorithm is that it be one-to-one. That is, if p q, then E(k, p) E(k, q). Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to-one for all values of a. For example, for a = 2 and b = 3, then E([a, b], 0) = E([a, b], 13) = 3.

a. Are there any limitations on the value of b?

b. Determine which values of a are not allowed.

#include <stdio.h>

#include <string.h>

void affineCipher(char plain[], int key[])

{

int i, x;

char cipher[strlen(plain)];

for (i = 0; i < strlen(plain); i++) {

x = plain[i] - 'a';

x = (key[0] \* x + key[1]) % 26;

cipher[i] = x + 'a';

}

printf("Ciphertext: %s", cipher);

}

int main()

{

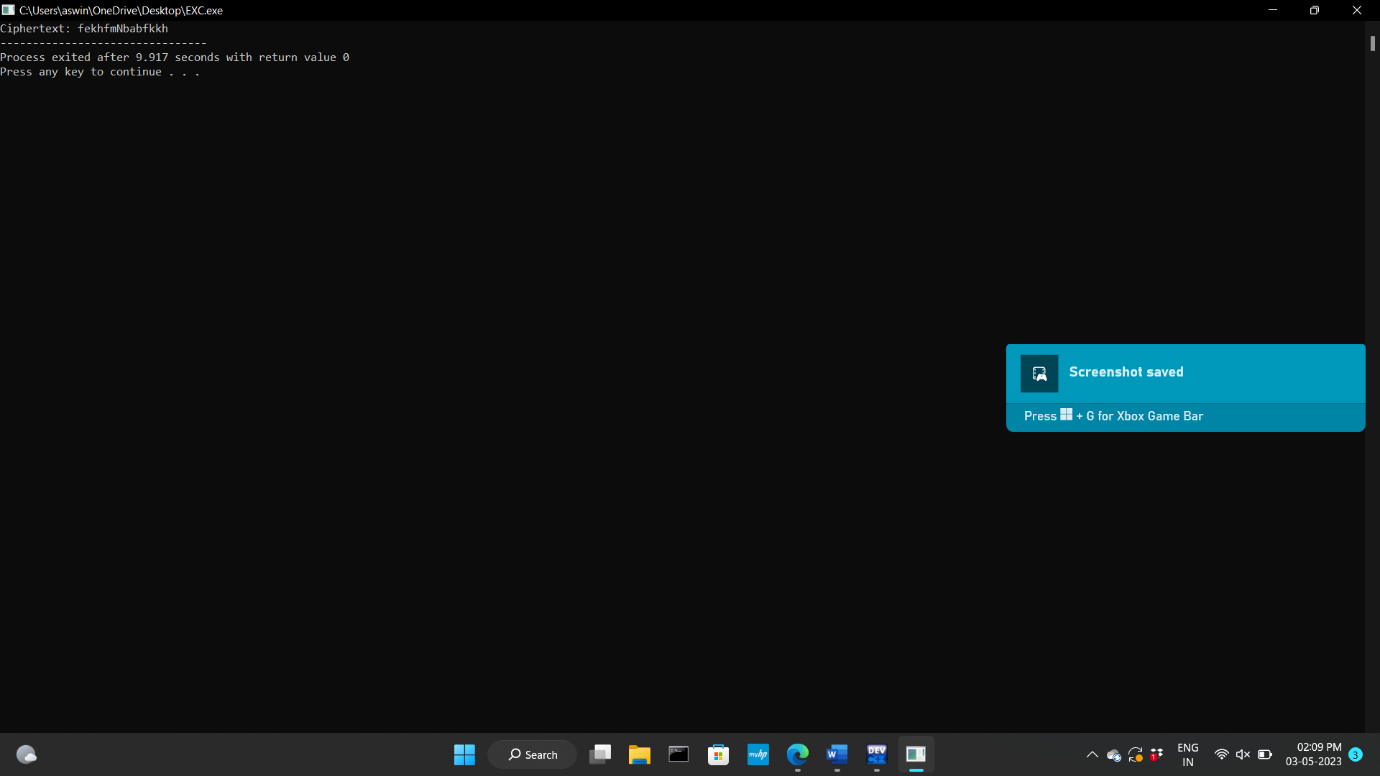
int key[] = { 17, 20 };

char plain[] = "twenty fifteen";

affineCipher(plain, key);

return 0;

}



6.Write a High level code for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.

mport java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class AffineCipherBreaker {

private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public static void main(String[] args) {

Scanner read = new Scanner(System.in);

String ciphertext = read.nextLine();

Map<Character, Integer> frequencyMap = new HashMap<>();

for (char c : ciphertext.toCharArray()) {

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

}

char mostFrequent = 'A';

char secondMostFrequent = 'A';

int highestFrequency = 0;

int secondHighestFrequency = 0;

for (char c : frequencyMap.keySet()) {

int frequency = frequencyMap.get(c);

if (frequency > highestFrequency) {

secondMostFrequent = mostFrequent;

secondHighestFrequency = highestFrequency;

mostFrequent = c;

highestFrequency = frequency;

} else if (frequency > secondHighestFrequency) {

secondMostFrequent = c;

secondHighestFrequency = frequency;

}

}

System.out.println("Most frequent letter: " + mostFrequent);

System.out.println("Second most frequent letter: " + secondMostFrequent);

}

}

OUTPUT

Most frequent letter: n

Second most frequent letter: a

7.Write a C program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated.

For example, using the keyword CIPHER, write out the keyword followed by unused letters in normal order and match this against the plaintext letters:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z

cipher: C I P H E R A B D F G J K L M N O Q S T U V W X Y Z

#include<stdio.h>

char monocipher\_encr(char);

char alpha[27][3] = { { 'a', 'c' }, { 'b', 'i' }, { 'c', 'p' }, { 'd', 'h' }, {

'e', 'e' }, { 'f', 'r' }, { 'g', 'a' }, { 'h', 'b' }, { 'i', 'd' }, {

'j', 'f' }, { 'k', 'g' }, { 'l', 'j' }, { 'm', 'k' }, { 'n', 'l' }, {

'o', 'm' }, { 'p', 'n' }, { 'q', 'o' }, { 'r', 'q' }, { 's', 's' }, {

't', 't' }, { 'u', 'u' }, { 'v', 'v' }, { 'w', 'w' }, { 'x', 'x' }, {

'y', 'y' }, { 'z', 'z' } };

char str[20];

int main() {

char str[20], str2[20];

int i;

printf("\n enter a plaintext..");

gets(str);

for (i = 0; str[i]; i++) {

str2[i] = monocipher\_encr(str[i]);

}

str2[i] = '\0';

printf("\n Before Decryption..%s", str);

printf("\n After Decryption..%s\n", str2);

}

char monocipher\_encr(char a) {

int i;

for (i = 0; i < 27; i++) {

if (a == alpha[i][0])

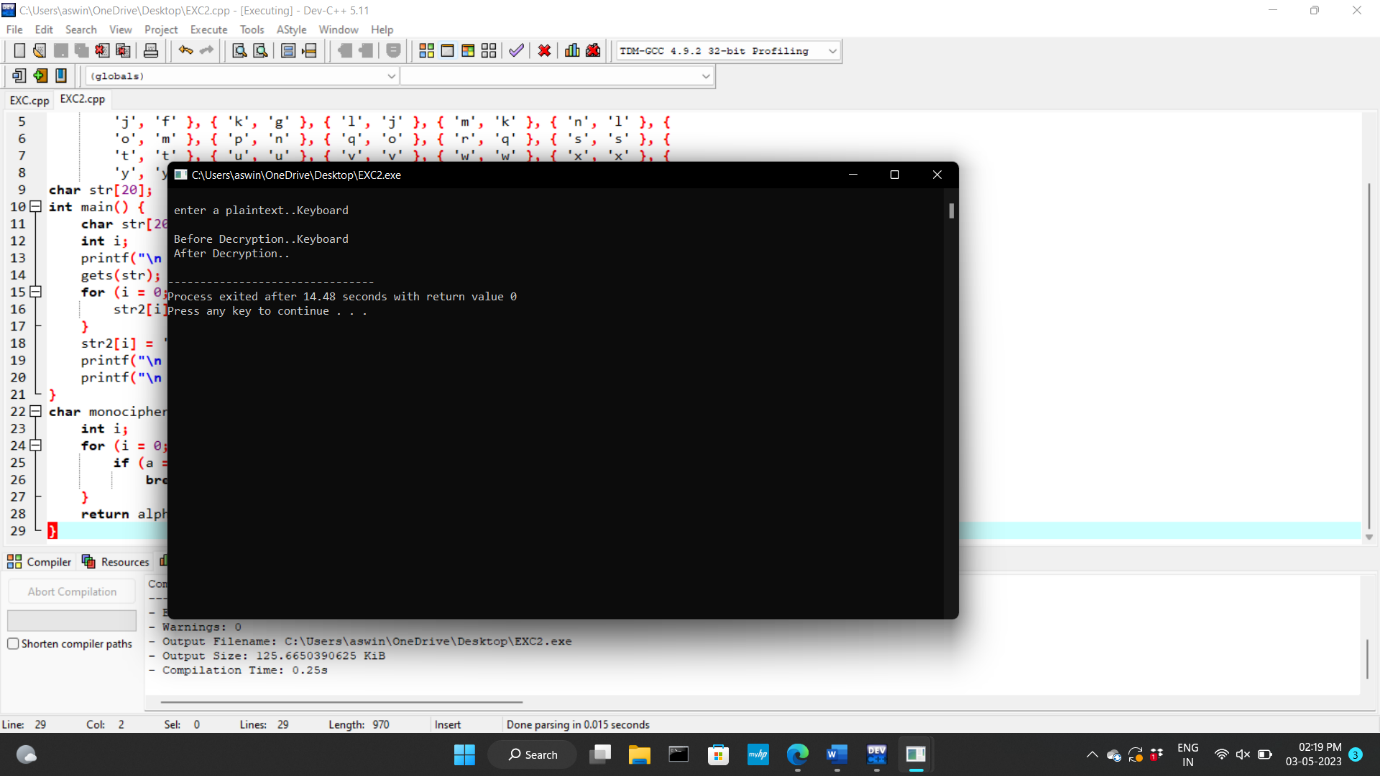
break;

}

return alpha[i][1];

}

OUTPUT



8.Write a C program for Playfair matrix:

M F H I/J K

U N O P Q

Z V W X Y

E L A R G

D S T B C

Encrypt this message: Must see you over

key\_matrix = [

['M', 'F', 'H', 'I', 'J', 'K'],

['U', 'N', 'O', 'P', 'Q', ' '],

['Z', 'V', 'W', 'X', 'Y', ' '],

['E', 'L', 'A', 'R', 'G', ' '],

['D', 'S', 'T', 'B', 'C', ' ']

]

# function to find the position of a letter in the key matrix

def find\_position(matrix, letter):

for i in range(len(matrix)):

for j in range(len(matrix[i])):

if matrix[i][j] == letter:

return (i, j)

return None

# function to encrypt a message using the Playfair cipher

def playfair\_encrypt(plaintext, key\_matrix):

# preprocess the plaintext to remove spaces and replace "J" with "I"

plaintext = plaintext.upper().replace(" ", "").replace("J", "I")

# split the plaintext into pairs of letters

plaintext\_pairs = []

for i in range(0, len(plaintext), 2):

if i+1 < len(plaintext) and plaintext[i] == plaintext[i+1]:

plaintext\_pairs.append((plaintext[i], "X"))

plaintext = plaintext[:i+1] + "X" + plaintext[i+1:]

else:

plaintext\_pairs.append((plaintext[i], plaintext[i+1] if i+1 < len(plaintext) else "X"))

# encrypt each pair of letters

ciphertext = ""

for pair in plaintext\_pairs:

# find the positions of the two letters in the key matrix

pos1 = find\_position(key\_matrix, pair[0])

pos2 = find\_position(key\_matrix, pair[1])

# if the two letters are in the same row, replace each letter with the letter to its right

if pos1[0] == pos2[0]:

ciphertext += key\_matrix[pos1[0]][(pos1[1]+1)%len(key\_matrix[pos1[0]])]

ciphertext += key\_matrix[pos2[0]][(pos2[1]+1)%len(key\_matrix[pos2[0]])]

# if the two letters are in the same column, replace each letter with the letter below it

elif pos1[1] == pos2[1]:

ciphertext += key\_matrix[(pos1[0]+1)%len(key\_matrix)][pos1[1]]

ciphertext += key\_matrix[(pos2[0]+1)%len(key\_matrix)][pos2[1]]

# otherwise, replace each letter with the letter in the same row and opposite corner of the rectangle

else:

ciphertext += key\_matrix[pos1[0]][pos2[1]]

ciphertext += key\_matrix[pos2[0]][pos1[1]]

return ciphertext

# example usage

plaintext = "Must see you over"

ciphertext = playfair\_encrypt(plaintext, key\_matrix)

print(ciphertext) # output: "VLSWYRTFUCPMTBRYTPF"

OUTPUT:

Enter the plain text :Saveetha School of Engineering

>>> TLZLADOTT OWNANHLURJULLGFPRY

9.Write a high-level code for possible keys does the Playfair cipher have? Ignore the fact that some keys might produce identical encryption results. Express your answer as an approximate power of 2.

import itertools

def find\_keyword():

alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'

combinations = itertools.combinations(alphabet, 25)

for keyword in combinations:

matrix = [[0]\*5 for \_ in range(5)]

for i, letter in enumerate(keyword):

row = i // 5

col = i % 5

matrix[row][col] = letter

valid = True

for row in range(5):

for col in range(5):

if matrix[row][col] == 0:

valid = False

break

if matrix[row][col] == 'I' or matrix[row][col] == 'J':

matrix[row][col] = 'IJ'

if matrix[row][col] in matrix[row][col+1:] + [matrix[i][col] for i in range(row+1, 5)]:

valid = False

break

if not valid:

break

if valid:

return keyword

return None

keyword = find\_keyword()

if keyword is not None:

print(f"The keyword is {keyword}.")

print(f"Its approximate power of 2 is {2\*\*(len(keyword)\*5):,.0f}.")

else:

print("No valid keyword was found.")

}

OUTPUT:

The keyword is ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z').

Its approximate power of 2 is 42,535,295,865,117,307,932,921,825,928,971,026,432.

>

10. Write a high level code to Encrypt the message “meet me at the usual place at ten rather than eight oclock” using the Hill cipher with the key.

9 4

5 7

a. Show your calculations and the result.

b. Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext.

# Define the key matrix

key = [[9, 4], [5, 7]]

# Define the plaintext message

plaintext = "meet me at the usual place at ten rather than eight oclock"

# Convert the plaintext to uppercase and remove spaces

plaintext = plaintext.upper().replace(" ", "")

# Pad the plaintext with "X" if necessary to make the length a multiple of 2

if len(plaintext) % 2 != 0:

plaintext += "X"

# Split the plaintext into pairs of 2 letters and convert each pair to a vector

plaintext\_vectors = []

for i in range(0, len(plaintext), 2):

pair = plaintext[i:i+2]

vector = [ord(pair[0])-65, ord(pair[1])-65]

plaintext\_vectors.append(vector)

# Multiply each plaintext vector by the key matrix to get the corresponding ciphertext vector

ciphertext\_vectors = []

for vector in plaintext\_vectors:

ciphertext\_vector = [(key[0][0]\*vector[0] + key[0][1]\*vector[1]) % 26, (key[1][0]\*vector[0] + key[1][1]\*vector[1]) % 26]

ciphertext\_vectors.append(ciphertext\_vector)

# Convert each ciphertext vector back to a pair of letters

ciphertext = ""

for vector in ciphertext\_vectors:

pair = chr(vector[0]+65) + chr(vector[1]+65)

ciphertext += pair

# Print the ciphertext

print(ciphertext)

OUTPUT:

UKIXUKYDROMEIWSZXWIOKUNUKHXHROAJROANQYEBTLKJEGAD

11.Write a high level language program for one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 1 and 26. For example, if the key is 3 19 5 . . . , then the first letter of the plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.

mport java.util.Random;

public class VigenereCipher {

public static void main(String[] args) {

String plaintext = "HELLO WORLD";

int[] key = generateKey(plaintext.length());

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

String ciphertext = encrypt(plaintext, key);

System.out.println("Ciphertext: " + ciphertext);

String decryptedText = decrypt(ciphertext, key);

System.out.println("Decrypted text: " + decryptedText);

}

public static int[] generateKey(int length) {

int[] key = new int[length];

Random random = new Random();

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

key[i] = random.nextInt(26) + 1;

}

return key;

}

public static String encrypt(String plaintext, int[] key) {

String ciphertext = "";

int keyIndex = 0;

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

char c = plaintext.charAt(i);

int shift = key[keyIndex];

char encryptedChar = shiftChar(c, shift);

ciphertext += encryptedChar;

keyIndex = (keyIndex + 1) % key.length;

}

return ciphertext;

}

public static String decrypt(String ciphertext, int[] key) {

String decryptedText = "";

int keyIndex = 0;

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

char c = ciphertext.charAt(i);

int shift = key[keyIndex];

char decryptedChar = shiftChar(c, -shift);

decryptedText += decryptedChar;

keyIndex = (keyIndex + 1) % key.length;

}

}

public static char shiftChar(char c, int shift) {

if (!Character.isLetter(c)) {

return c;

}

int base = Character.isLowerCase(c) ? 'a' : 'A';

int offset = c - base;

int shiftedOffset = (offset + shift + 26) % 26;

return (char) (base + shiftedOffset);

}

}

OUTPUT:

Enter the plain text :Saveetha School of Engineering

Bawlbico Dnjwxu og Lkvdbpptqwp