EXP NO: 1a
DATE: 27/1/24

CAESAR CIPHER

AIM:

To write a python program implementing caesar cipher algorithm

ALGORITHM:

- Input: Read the plaintext message and the shift value (an integer).
- Initialize: Create an empty string for the ciphertext.
- Iterate: Loop through each character in the plaintext.
- Check: For each character, check if it is a letter (ignore non-letter characters).
- Shift: Calculate the shifted position for each letter:
- For uppercase letters, use the formula: (ord(char) - ord('A') + shift) % 26 + ord('A')
- For lowercase letters, use the formula: (ord(char) - ord('a') + shift) % 26 + ord('a')
- Append: Append the shifted character to the ciphertext.
- Output: Print or return the final ciphertext.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <ctype.h>

int main()
{
    char message[500], c;
    int i;
    int key;

    printf("Enter a message to encrypt: ");
    scanf("%[^\n]", message); // Read the whole line including spaces

    printf("Enter key: ");
    scanf("%d", &key);

    for (i = 0; message[i] != '\0'; i++) {
        c = message[i];

        // Encrypt alphabets (both lowercase and uppercase)
        if (isalpha(c)) {
            if (islower(c)) {
                c = (c - 'a' + key) % 26 + 'a';
            } else {
                c = (c - 'A' + key) % 26 + 'A';
            }
        } else { // Encrypt special characters
            c = (c + key) % 128;
        }

        message[i] = c;
    }

    printf("Encrypted message: %s\n", message);

    printf("*****Decryption*****");
    char message[500], c;
```

```
    int i;
    int key;
    printf("Enter a message to decrypt: ");
    scanf("%[^\n]", message); // Read the whole line including spaces
    printf("Enter key: ");
    scanf("%d", &key);
    for (i = 0; message[i] != '\0'; i++) {
        c = message[i];
        // Decrypt alphabets (both lowercase and uppercase)
        if (isalpha(c)) {
            if (islower(c)) {
                c = (c - 'a' - key + 26) % 26 + 'a';
            } else {
                c = (c - 'A' - key + 26) % 26 + 'A';
            }
        } else { // Decrypt special characters
            c = (c - key + 128) % 128;
        }

        message[i] = c;
    }
    printf("Decrypted message: %s\n", message);

    return 0;
}
```

OUTPUT:

```
┌──(kali㉿kali)-[~/Documents/cnslab]
└─$ gcc caesar.c

┌──(kali㉿kali)-[~/Documents/cnslab]
└─$ ./a.out
Enter a message to encrypt: Cryptography and Network Security
Enter key: 3
Encrypted message: Fubswrjudskb#dqg#Qhwzrun#Vhfxulwb
```

RESULT:

Thus a C program was implemented to demonstrate Caesar Cipher.

EXP NO: 1b
DATE: 03/02/2024

PLAYFAIR CIPHER

AIM:

To write a python program implementing playfair cipher algorithm

ALGORITHM:

- Input: Read the plaintext message and the keyword.
- Generate Key Matrix: Create a 5x5 matrix using the keyword (eliminate duplicates, fill with remaining letters of the alphabet, treating I and J as the same).
- Preprocess: Prepare the plaintext by pairing letters, adding 'X' between duplicates and at the end if needed.
- Iterate Pairs: Loop through each pair of letters in the plaintext.
- Encrypt Pair: For each pair, apply Playfair rules:
- Same row: Replace each letter with the one to its right (wrap around to the beginning).
- Same column: Replace each letter with the one below it (wrap around to the top).
- Rectangle: Replace each letter with the one in its row but in the column of the other letter of the pair.
- Append: Construct the ciphertext by combining the encrypted pairs.
- Output: Print or return the final ciphertext.

PROGRAM:

```
def toLowerCase(text):
```

```
        return text.lower()

# Function to remove all spaces in a string


def removeSpaces(text):
        newText = ""
        for i in text:
                if i == " ":
                        continue
                else:
                        newText = newText + i
        return newText

# Function to group 2 elements of a string
# as a list element


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

# Function to fill a letter in a string element
# If 2 letters in the same string matches


def FillerLetter(text):
        k = len(text)
        if k % 2 == 0:
                for i in range(0, k, 2):
                        if text[i] == text[i+1]:
```

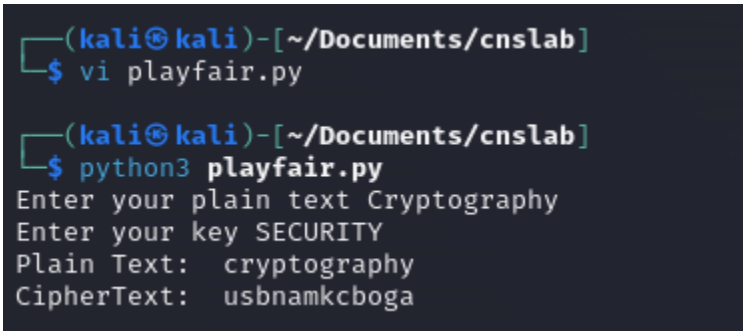
```
                        new_word = text[0:i+1] + str('x') + text[i+1:]
                        new_word = FillerLetter(new_word)
                        break
                else:
                        new_word = text
        else:
                for i in range(0, k-1, 2):
                        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']

# Function to generate the 5x5 key square matrix


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 = input("Enter your plain text ")
```

```
text_Plain = removeSpaces(toLowerCase(text_Plain))
PlainTextList = Diagraph(FillerLetter(text_Plain))
if len(PlainTextList[-1]) != 2:
        PlainTextList[-1] = PlainTextList[-1]+'z'
key = input("Enter your 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:



RESULT:

Thus a python program has been implemented to demonstrate Playfair Cipher.

EXP NO: 1c
DATE: 10/02/2024

RAIL FENCE CIPHER

AIM:

To write a python program implementing rail fence cipher algorithm

ALGORITHM:

- Input: Read the plaintext message and the number of rails (rows).
- Initialize Rails: Create a list of empty strings, one for each rail.
- Set Direction: Initialize variables for the current rail and direction (downward).
- Iterate Characters: Loop through each character in the plaintext.
- Place Characters: Append the character to the current rail and adjust the direction (change at top and bottom rails).
- Combine Rails: Concatenate all strings from the rails to form the ciphertext.
- Output: Print or return the final ciphertext.

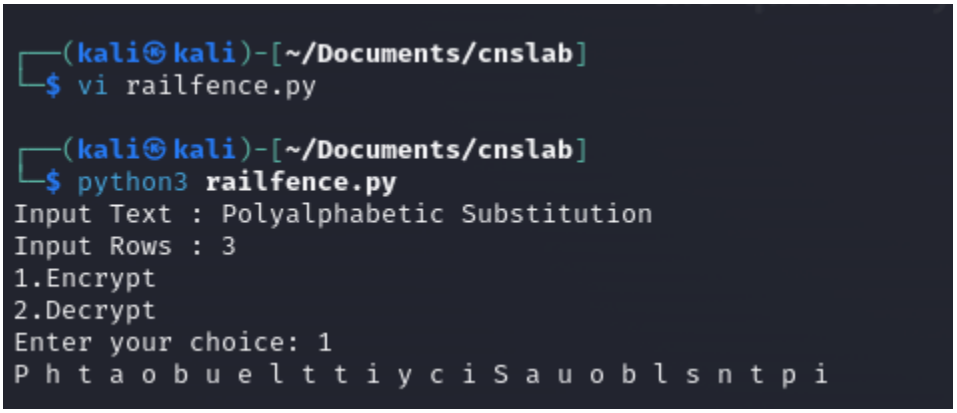
PROGRAM:

```
def main():
    text = input('Input Text : ')
    rows = int(input('Input Rows : '))
    text = text.replace(' ','')

    while True:
        chc = input('1.Encrypt\n2.Decrypt\nEnter your choice: ')
        if chc in ['0','1']:
            break
            print('Choose 0 / 1')

    #print(len(text))
    if int(chc):
        arr = [[ ' ' for y in range(len(text))] for x in range(rows)]
        #[ print(row) for row in arr ]

        dir_down = None
        row, col = 0 , 0
        for i in range(len(text)):
            if row == 0: dir_down = True
            if row == rows - 1: dir_down = False

            arr[row][col] = '*'
            col += 1

            if dir_down: row += 1
            else: row -= 1

        #print('\n\n')
        #[ print(row) for row in arr ]
        count = 0
        for row in arr:
            for i in range(len(row)):
                if row[i] == '*':
                    row[i] = text[count]
```

```
                count += 1

        #print('\n\n')
        #[ print(row) for row in arr ]

        result = []
        row, col = 0, 0
        for i in range(len(text)):

            if row == 0: dir_down = True
            if row == rows-1: dir_down = False

            if (arr[row][col] != '*'):
                result.append(arr[row][col])
                col += 1

            if dir_down: row += 1
            else: row -= 1

        print(" ".join(result).strip())
else:
        arr = [ [] for x in range(rows)]
        #print(arr)
        count = 0
        finish = False

        while True:
            for j in range(0,rows-1):
                arr[j].append(text[count])
                count += 1

                if count >= len(text):
                    finish = True
                    break

            if finish :
                break
```

```
            for k in range(rows - 1 ,0,-1):
                arr[k].append(text[count])
                count += 1

                if count >= len(text):
                    finish = True
                    break

            if finish :
                break
        print(arr)

main()
```

OUTPUT:



RESULT:

Thus, a python program has been implemented to demonstrate Rail Fence Cipher.

EXP NO: 1d
DATE:

COLUMNAR TRANSPOSITION TECHNIQUES

AIM:

To write a python program implementing columnar transposition techniques.

ALGORITHM:

- Input: Read the plaintext message and the keyword.
- Generate Key Order: Determine the numerical order of the keyword letters (e.g., "CIPHER" → [3, 1, 4, 5, 2, 6]).
- Pad Message: Pad the plaintext with spaces to fit into a rectangle where the number of columns is the length of the keyword.
- Fill Grid: Write the plaintext into a grid, row by row, with columns equal to the length of the keyword.
- Read Columns: Read the columns in the order specified by the keyword.
- Construct Ciphertext: Concatenate the characters read from each column to form the ciphertext.
- Output: Print or return the final ciphertext.

PROGRAM:

```
import math

key = input("Enter the key ")

# Encryption
def encryptMessage(msg):
	cipher = ""

	# track key indices
	k_indx = 0

	msg_len = float(len(msg))
	msg_lst = list(msg)
	key_lst = sorted(list(key))

	# calculate column of the matrix
	col = len(key)

	# calculate maximum row of the matrix
	row = int(math.ceil(msg_len / col))

	# add the padding character '_' in empty
	# the empty cell of the matix
	fill_null = int((row * col) - msg_len)
	msg_lst.extend('_' * fill_null)

	# create Matrix and insert message and
	# padding characters row-wise
	matrix = [msg_lst[i: i + col]
			for i in range(0, len(msg_lst), col)]

	# read matrix column-wise using key
	for _ in range(col):
		curr_idx = key.index(key_lst[k_indx])
		cipher += ''.join([row[curr_idx]
```

```
                                        for row in matrix])
                k_indx += 1

        return cipher

# Decryption
def decryptMessage(cipher):
        msg = ""

        # track key indices
        k_indx = 0

        # track msg indices
        msg_indx = 0
        msg_len = float(len(cipher))
        msg_lst = list(cipher)

        # calculate column of the matrix
        col = len(key)

        # calculate maximum row of the matrix
        row = int(math.ceil(msg_len / col))

        # convert key into list and sort
        # alphabetically so we can access
        # each character by its alphabetical position.
        key_lst = sorted(list(key))

        # create an empty matrix to
        # store deciphered message
        dec_cipher = []
        for _ in range(row):
                dec_cipher += [[None] * col]

        # Arrange the matrix column wise according
        # to permutation order by adding into new matrix
        for _ in range(col):
```

```
                curr_idx = key.index(key_lst[k_indx])

                for j in range(row):
                        dec_cipher[j][curr_idx] = msg_lst[msg_indx]
                        msg_indx += 1
                k_indx += 1

        # convert decrypted msg matrix into a string
        try:
                msg = ''.join(sum(dec_cipher, []))
        except TypeError:
                raise TypeError("This program cannot","handle repeating words.")
        null_count = msg.count('_')
        if null_count > 0:
                return msg[: -null_count]
        return msg
msg = input("Enter the plain text ")
cipher = encryptMessage(msg)
print("Encrypted Message: {}".
                        format(cipher))

print("Decryped Message: {}".
        format(decryptMessage(cipher)))
```

OUTPUT:

```
┌──(kali㉿kali)-[~/Documents/cnslab]
└─$ vi columnar.py

┌──(kali㉿kali)-[~/Documents/cnslab]
└─$ python3 columnar.py
Enter the key 53412
Enter the plain text Cyptography and Network Security
Encrypted Message: tpnt r_ohdwSi_yr Nrcypaaeku_Cgy oet
```

RESULT:

Thus, a python program has been implemented to demonstrate Columnar Transposition techniques.