EXP NO: 1a
DATE: 23/1/24

# CAESAR CIPHER

**AIM:**

To write a python program implementing caesar cipher algorithm

**ALGORITHM:**

1. Get the plaintext from the user
2. Get the secret key from the user
3. If the character is uppercase take the ascii value of it and add with the key and subtract with original ascii value modulus with total number of characters.
4. If it is lowercase alphabet take its ascii value and do necessary operation modulus with total.
5. For digits and special characters take its ascii value and process it in its range.
6. Print the encrypted text.
7. Subtract the key from encrypted text to get original text.

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: 30/01/2024

# PLAYFAIR CIPHER

**AIM:**

To write a python program implementing playfair cipher algorithm

**ALGORITHM:**

1. Get the plaintext from the user
2. Get the key from the user
3. Plaintext is encrypted two letters at a time
4. If a pair is a repeated letter, insert filler like 'X'
5. If both letters fall in the same row, replace each with letter to right (wrapping back to start from end)
6. If both letters fall in the same column, replace each with the letter below it (again wrapping to top from bottom)
7. Otherwise each letter is replaced by the letter in the same row and in the column of the other letter of the pair.

**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:
```

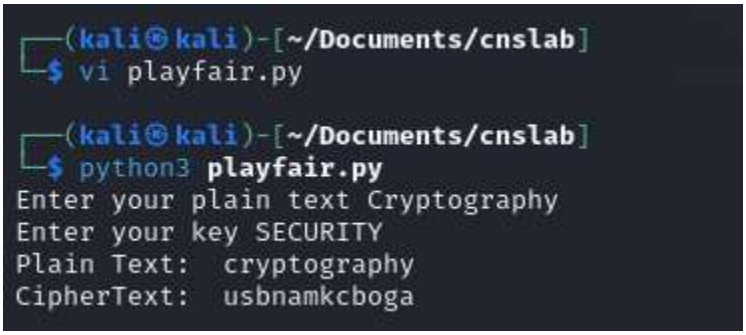
```
                for i in range(0, k, 2):
                        if text[i] == text[i+1]:
                                new_word = text[0:i+1] + str('x') + text[i+1:]
                                new_word = FillerLetter(new_word)
                                break
                        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: 06/02/2024

# RAIL FENCE CIPHER

**AIM:**

To write a python program implementing rail fence cipher algorithm

**ALGORITHM:**

1. Get the plain text from the user
2. Set the key as 2 by default.
3. Arrange the plaintext in two rows in a zig-zag manner.
4. Derive the cipher text by adding the first row of arrangement with the second row of arrangement.
5. Get the original text by using the cipher text and arranging it in zigzag manner and repeat the same process.

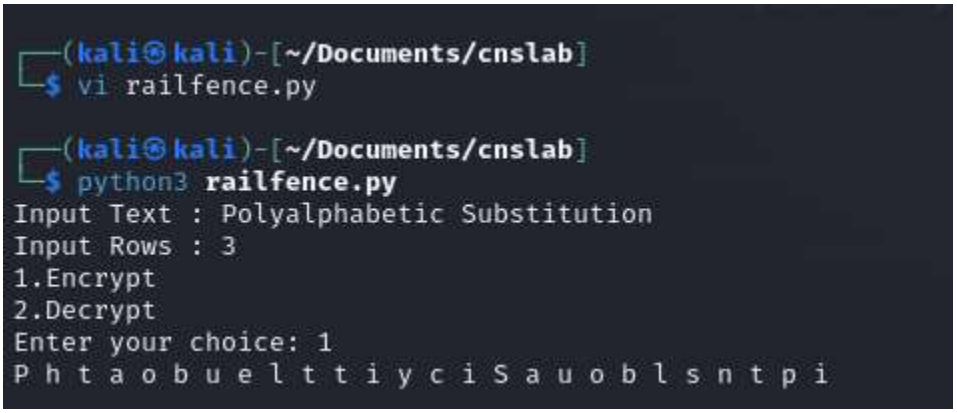
**PROGRAM:**

```
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: 13/03/2024

# COLUMNAR TRANSPOSITION TECHNIQUES

**AIM:**

To write a python program implementing columnar transposition techniques.

**ALGORITHM:**

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. The permutation is defined by the alphabetical order of the letters in the keyword.
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
5. Finally, the message is printed off in columns, in the order specified by the keyword.

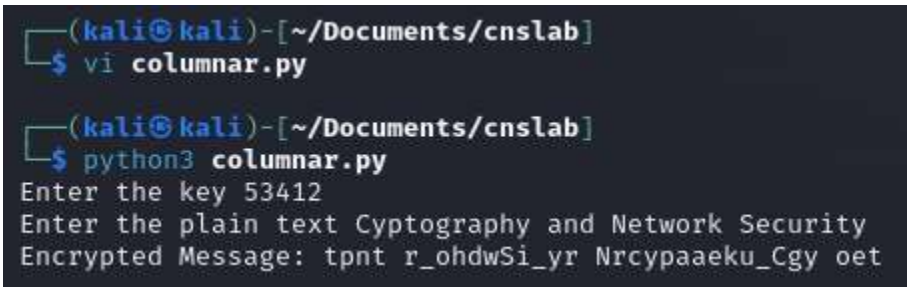
**PROGRAM:**

```
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:**




**RESULT:**

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