EXP NO: 1d

DATE:

COLUMNAR TRANSPOSITION TECHNIQUES

AIM:

To write a C Program to perform Columnar Transposition Techniques for both encryption and decryption process

ALGORITHM:

Step 1 : Start.

Step 2 : Include necessary header files.

Step 3 : Create a function for encryption process.

Step 4 : Define encryption function.

Step 5 : Create a function for decryption process.

Step 6 : Define decryption function.

Step 7 : Call both encryption and decryption function inside main function.

Step 8 : End.

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.