Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

SUB: Information Security

AY 2023-24 (Semester-V)

Jhanvi Parekh
60009210033
D11

Experiment No: 5

Aim: To Implement Encryption and Decryption using Columnar Transposition Cipher.

Theory:

1. Transposition Cipher
2. Columnar Transposition Cipher/ Row Column Transposition Cipher.

Example:
Encryption and Decryption

1) **Plaintext : "Attack Postponed until two am"**
   Keyword: 4312567
   Ciphertext:

Conclusion:
In conclusion, the column transposition cipher is a simple yet effective encryption technique that relies on rearranging the characters of a plaintext message to create a cipher text.
This method offers a degree of security against casual attackers but is not suitable for protecting highly sensitive information, as it can be easily broken through various cryptographic attacks, especially with the availability of modern computing power. While column transposition ciphers can be fun to explore and use for educational purposes or as part of simple encryption games, they are not recommended for serious data protection or privacy needs. For robust security, it is advisable to employ more advanced encryption techniques, such as modern symmetric or asymmetric encryption algorithms, which provide a higher level of security and are widely used in today's digital communications and information security practices.

SUB: Information Security

LINK:
https://colab.research.google.com/drive/1OT0wQE_1gtZbQXmX_7Y2TobHktV7IEEA?usp=sharing

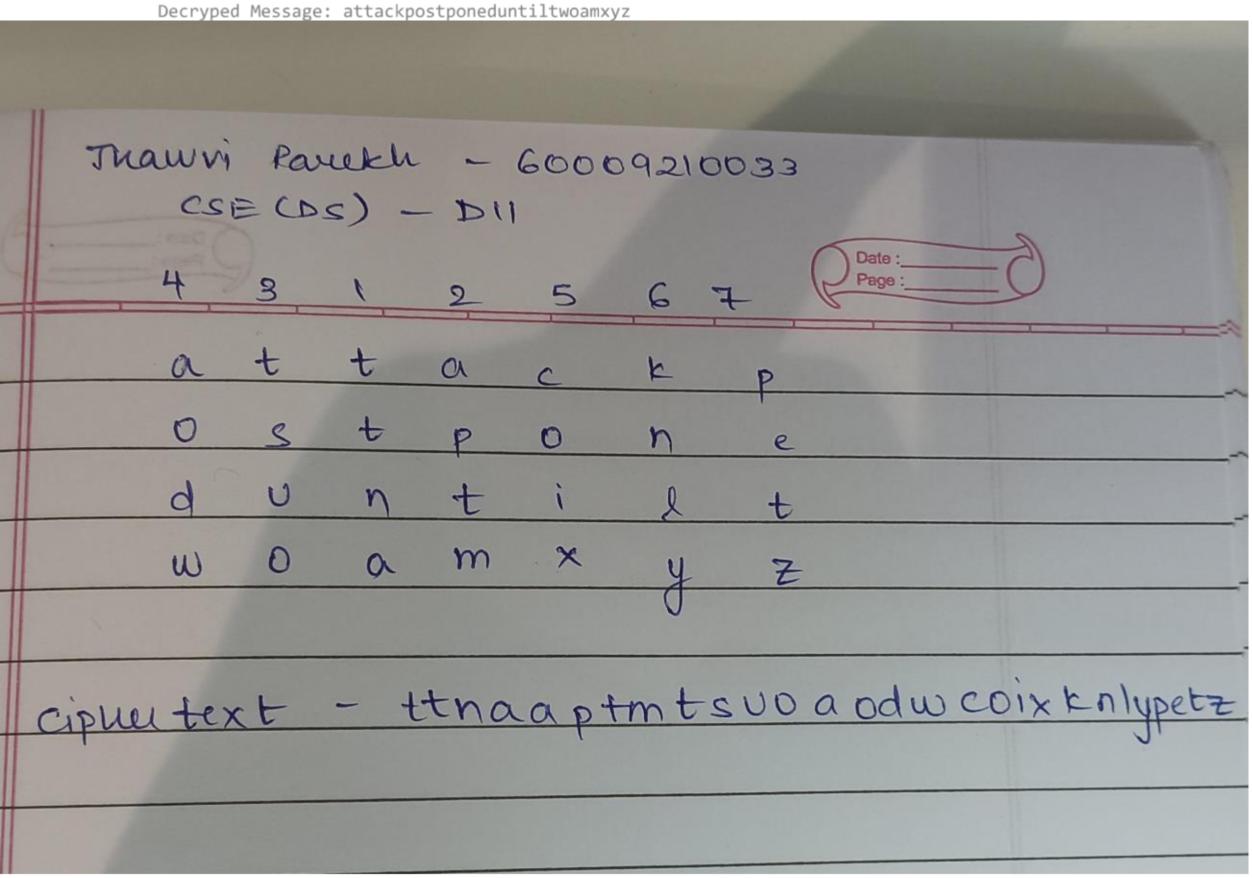
```
import math

key = "4312567"

# Encryption
def encryptMessage(msg):
    cipher = ""
    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    col = len(key)
    row = int(math.ceil(msg_len / col))
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)
    matrix = [msg_lst[i: i + col] for i in range(0, len(msg_lst), col)]

    # Print the grid
    print("Grid:")
    for row in matrix:
        print(' | '.join(row))
    print()  # Add a newline for readability

    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


msg = "attackpostponeduntiltwoamxyz"

cipher = encryptMessage(msg)
print("Encrypted Message: {}".format(cipher))
```

```
Grid:
a | t | t | a | c | k | p
o | s | t | p | o | n | e
d | u | n | t | i | l | t
w | o | a | m | x | y | z

Encrypted Message: ttnaaptmtsuoaodwcoixknlypetz
```

```
def decryptMessage(cipher):
  msg = ""
  k_indx = 0
  msg_indx = 0
  msg_len = float(len(cipher))
  msg_lst = list(cipher)
  col = len(key)
  row = int(math.ceil(msg_len / col))

  key_lst = sorted(list(key))
  dec_cipher = []
  for _ in range(row):
    dec_cipher += [[None] * col]
  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
  try:
    msg = ''.join(sum(dec_cipher, []))
  except TypeError:
    raise TypeError("This program cannot",
            "handle repeating words.")

  null_count = msg.count('_')

  if null_count > 0:
```

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

```
Decryped Message: attackpostponeduntiltwoamxyz
```



Jhanvi Parekh - 60009210033

CSE (DS) - D11

Date :

Page :

4	3	1	2	5	6	7
a	t	t	a	c	k	p
o	s	t	p	o	n	e
d	u	n	t	i	l	t
w	o	a	m	x	y	z

cipher text - ttnaaptmtsuoaodwcoixknlypetz