Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

Lab 12: Write a python program for Transposition Technique using Rail fence Technique and columnar Technique.

Date of the Session:_____/___/_____ **Session Time:____to_____**

Learning outcome:

- Understand the principles and operations of both the Rail Fence and Columnar ciphers. Be familiar with their historical context and modern applications.
- Learn how to encrypt and decrypt messages using both the Rail Fence and Columnar ciphers. Understand the algorithms and processes involved
- Understand how the Rail Fence cipher works, where letters are written diagonally in a "zigzag" pattern..
- Understand where they are suitable and how they've been historically employed.

Pre-Lab Task:

1. Learn how to set up the number of rails and perform encryption and decryption operations. Practice these methods on sample messages.

- Choose a **number of rails** (e.g., 3).
- Write the message in a **zigzag** pattern.
- Read row-by-row to encrypt, reverse to decrypt.
- Try with different rails on short messages.

Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

2. Explore potential cryptanalysis methods for breaking the Rail Fence cipher and learn how to recover plaintext from an encrypted message.

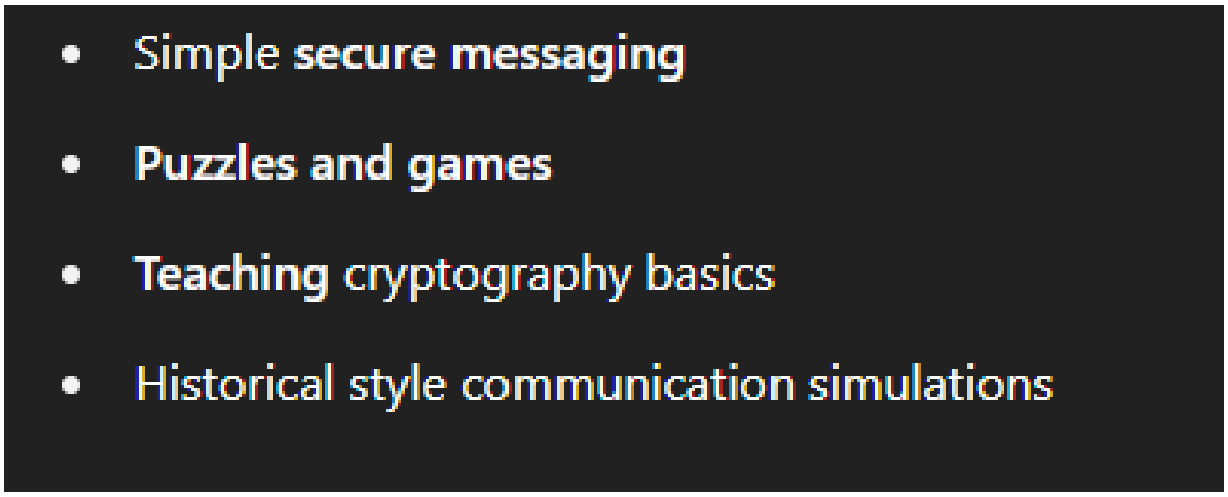
- Use **brute force**: try all rail numbers.
- Look for **readable patterns** in outputs.
- Useful for short ciphertexts or weak keys.

3. Practice selecting appropriate keys or keywords for different messages and varying the key lengths for encryption.

- For **Rail Fence**: test different rail values.
- For **Columnar**: try keywords of various lengths.
- See how the key changes the ciphertext.

Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

4. Identify specific use cases and scenarios where the Columnar cipher may be useful, such as secure messaging



Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

In Lab Task: Write a python program for Transposition Technique using Rail fence Technique and columnar Technique

Rail fence Technique

Implementing the Rail Fence transposition technique is relatively straightforward. It involves arranging the characters of a plaintext message in a zigzag pattern along a set number of "rails" or rows, then reading the characters in a specific order to create the ciphertext.

Here are the steps for implementing the Rail Fence technique:

Step 1: Choose the Number of Rails (Rows)

- Decide on the number of rails or rows you want to use for the Rail Fence. This determines the depth of the zigzag pattern.

Step 2: Prepare the Plaintext

- Take your plaintext message, remove spaces, and special characters if necessary, and ensure it is in a suitable format.

Step 3: Create the Zigzag Pattern

- Start at the top-left corner of your rail (row 1).
- Write the first character of your plaintext in this position.
- Continue writing characters diagonally, moving down one rail after each character, until you reach the bottom rail.
- When you reach the bottom rail, start moving diagonally up toward the top rail, following the zigzag pattern. Repeat this process until you've used all characters from your plaintext.

Step 4: Reading the Ciphertext

- Read the characters along each rail from left to right, starting with the top rail and moving downward.
- Concatenate the characters from each rail to form the ciphertext.

Step 5: Encryption Example

- Let's say you have a plaintext message: "HELLO WORLD" and you want to use 3 rails.

Create the zigzag pattern as follows:

H……..…..…O L

E……..L……..…W……..R.....D

L .....O

Read the characters from left to right along each rail: "HORDELLWOL."

The ciphertext is "HORDELLWOL."

Step 6: Decryption (Optional)

- To decrypt a Rail Fence-encrypted message, you need to know the number of rails used.
- Create an empty zigzag pattern with the same number of rails.
- Fill in the ciphertext characters in the pattern following the same zigzag pattern.
- Read the characters in the pattern from left to right to reveal the original plaintext.

Step 7: Key Management

- For decryption, it's crucial to know the number of rails used. This information serves as the decryption key.

By following these steps, you can successfully implement the Rail Fence transposition technique to encrypt and decrypt messages. Keep in mind that the security of the Rail Fence cipher is relatively low, and it is primarily used for educational or illustrative purposes.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	THANOS

Writing space for the Program: (For Student's use only)

Write a python program for Transposition Technique using Rail fence Technique

```
def rail_fence_encrypt(plaintext, rails):
    plaintext = ''.join(filter(str.isalnum, plaintext.upper()))
    rail = ['' for _ in range(rails)]
    direction_down = False
    row = 0

    for char in plaintext:
        rail[row] += char
        if row == 0 or row == rails - 1:
            direction_down = not direction_down
        row += 1 if direction_down else -1

    return ''.join(rail)


def rail_fence_decrypt(ciphertext, rails):
    n = len(ciphertext)
    matrix = [['\n' for _ in range(n)] for _ in range(rails)]
    direction_down = None
    row, col = 0, 0

    for i in range(n):
        if row == 0:
            direction_down = True
        if row == rails - 1:
            direction_down = False
        matrix[row][col] = '*'
        col += 1
        row += 1 if direction_down else -1

    index = 0
    for i in range(rails):
        for j in range(n):
            if matrix[i][j] == '*' and index < n:
                matrix[i][j] = ciphertext[index]
```

```
    index += 1

  result = []
  row, col = 0, 0
  for i in range(n):
    if row == 0:
      direction_down = True
    if row == rails - 1:
      direction_down = False
    if matrix[row][col] != '\n':
      result.append(matrix[row][col])
      col += 1
    row += 1 if direction_down else -1

  return ''.join(result)


plaintext = "HELLO WORLD"
rails = 3

cipher = rail_fence_encrypt(plaintext, rails)
print(f"Encrypted: {cipher}")

decrypted = rail_fence_decrypt(cipher, rails)
print(f"Decrypted: {decrypted}")
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	THANOS

Output

Encrypted: HOLELWRDLO

Decrypted: HELLOWORLD

Columnar Transposition Technique

Implementing the Columnar Transposition Technique involves a series of steps to encrypt and decrypt messages. Below are the steps to implement the Columnar Transposition Technique:

Encryption using Columnar Transposition:

1. **Select a Keyword:** Choose a keyword or key phrase that will determine the order of columns for transposition. For example, let's use the keyword "CRYPTO" for this demonstration.
2. **Write the Message:** Write down your plaintext message in rows beneath the keyword, starting from left to right. The keyword dictates the order of the columns.
   Example:
   Keyword: CRYPTO
   Plaintext: THIS IS A SECRET MESSAGE
   Arranged in columns:
   CRYPTO
   THISIS
   ASECRET
   MESSAGE
3. **Order the Columns**: Rearrange the columns alphabetically based on the letters in your keyword. In this case, the keyword "CRYPTO" would be ordered as "COPRTY."

Example:
CRYPTO
THISIS
ASECRET
MESSAGE
Rearranged:
COPRTY
TSIHIS
AETSREC
GMEASSE

4. **Read the Ciphertext**: Read the message row by row, from left to right. This is your ciphertext.
   Example:
   Ciphertext: "CTT AIG EME HSAS RST SEO S"

Decryption using Columnar Transposition:

1. Select a Keyword: Choose the same keyword used for encryption, in this case, "CRYPTO."
2. Determine the Number of Columns: Count the number of columns, which is equal to the length of your keyword.
3. Calculate the Number of Rows: To find the number of rows, divide the length of the ciphertext by the number of columns. If there's a remainder, add one row.
   Example:
   - Ciphertext length: 23
   - Number of columns: 6 (based on the keyword)
   - Rows = 23 / 6 = 3 rows with a remainder of 5, so we add one more row for a total of 4 rows.
4. **Recreate the Grid**: Create a grid with the same number of columns as the keyword and the calculated number of rows. Fill in the grid with the ciphertext in a row-by-row manner, left to right.
   Example:
   Keyword: CRYPTO
   Ciphertext: CTT AIG EME HSAS RST SEO S
   Reconstructed grid:
   CTTAIG
   EMEHSA
   SRSTSE
   OS

5. Sort the Columns: Sort the columns based on the keyword (in alphabetical order). In this case, it would be sorted as "COPRTY."
   Example:
   Sorted grid:

COPRTY
TSIHIS
AETSREC
GMEASSE

6. Read the Plaintext: Read the message row by row, from left to right. This is your plaintext.
   Example:
   Plaintext: "THIS IS A SECRET MESSAGE"

By following these steps, you can encrypt and decrypt messages using the Columnar Transposition Technique.

Writing space for the Program: (For Student's use only)

Write a python program for Transposition Technique using columnar Technique

```
import math

def encrypt_columnar_transposition(plaintext, keyword):
    plaintext = plaintext.replace(" ", "").upper()
    keyword = keyword.upper()
    num_columns = len(keyword)
    num_rows = math.ceil(len(plaintext) / num_columns)
    padded_length = num_rows * num_columns
    plaintext += '_' * (padded_length - len(plaintext))
    matrix = ['' for _ in range(num_columns)]
    for index in range(len(plaintext)):
        column = index % num_columns
        matrix[column] += plaintext[index]
    keyword_order = sorted([(char, i) for i, char in enumerate(keyword)])
    ciphertext = ''
    for char, i in keyword_order:
        ciphertext += matrix[i]
    return ciphertext

def decrypt_columnar_transposition(ciphertext, keyword):
    ciphertext = ciphertext.replace(" ", "").upper()
    keyword = keyword.upper()
    num_columns = len(keyword)
    num_rows = math.ceil(len(ciphertext) / num_columns)
```

```
    total_chars = len(ciphertext)

    short_cols = (num_columns * num_rows) - total_chars
    keyword_order = sorted([(char, i) for i, char in enumerate(keyword)])
    columns = ['' for _ in range(num_columns)]
    k = 0
    for index, (char, original_index) in enumerate(keyword_order):
        col_length = num_rows if index >= short_cols else num_rows - 1
        columns[original_index] = ciphertext[k:k + col_length]
        k += col_length
    plaintext = ''
    for row in range(num_rows):
        for col in range(num_columns):
            if row < len(columns[col]):
                plaintext += columns[col][row]
    return plaintext.replace('_', '')

if __name__ == "__main__":
    keyword = "CRYPTO"
    message = "THIS IS A SECRET MESSAGE"
    encrypted = encrypt_columnar_transposition(message, keyword)
    print("Encrypted message:", encrypted)
    decrypted = decrypt_columnar_transposition(encrypted, keyword)
    print("Decrypted message:", decrypted)
```

Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

Output

Encrypted message: TATGSEA_SCS_HSMEIRS_IEE_

Decrypted message: THISISASECRETMESSAGE

VIVA-VOCE Questions (In-Lab):

1. Explain the basic concept of the Rail Fence cipher. How does it work?

It's a transposition cipher where plaintext is written in a zigzag across multiple lines (rails), then read row by row to form ciphertext.

2. What is the key factor in the Rail Fence cipher, and how does it determine the number of "rails"?

The key is the **number of rails**. It decides the zigzag pattern and how text is rearranged.

3. What is the key space in the Rail Fence cipher, and how does it impact the security of the cipher?

The key space is from 2 to (length of message - 1). It's small, making the cipher easy to break by brute force.

4. Explain the basic concept of the Columnar cipher. How does it differ from the Rail Fence cipher?

Text is written in rows under a keyword. Columns are rearranged based on the alphabetical order of the keyword, then read column-wise to form ciphertext.

Difference: Rail Fence uses zigzag; Columnar uses column swapping.

Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

5. What is the role of the keyword or key phrase in the Columnar cipher? How is it used in encryption??

The keyword decides the column order. Each letter's alphabetical position determines how columns are rearranged for encryption.

Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

Post Lab Task:

1. Were you able to decrypt the Rail Fence-encrypted message without knowing the number of rails used during encryption? If so, how did you achieve this?

Yes, by trying different rail numbers (brute force) until the message makes sense.

2. Can you provide examples of real-world scenarios where the Rail Fence cipher might be used effectively for encryption?

Used in puzzles, games, or old wartime messages—**not secure** for modern use.

3. Compare the security and operational characteristics of the Columnar cipher with the Rail Fence cipher.

- **Rail Fence:** Simple, weak, uses number of rails.
- **Columnar:** More secure, uses keyword, better for real encryption.

Experiment #	<TO BE FILLED BY STUDENT>	**Student ID**	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	**Student Name**	THANOS

4. How does the appearance of the encrypted text change when you use different keywords or key lengths in the Columnar cipher?

Different **keywords** = different column orders = different ciphertexts.

Longer keywords = more columns = more variation.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date