Walchand College of Engineering, Sangli (Government Aided Autonomous Institute)	
AY 2025-26	
Course Information	
Programme	B.Tech. (Computer Science and Engineering)
Class, Semester	Final Year B. Tech., Sem VII
Course Code	6CS451 C
Course Name	Cryptography and Network Security Lab

Experiment No. 02

Name : Soham U. Patil

PRN NO : 22510019

Batch : B6

Title – Perform encryption and decryption using following transposition techniques a. Rail fence b. row and Column Transformation

Objectives:

Design and implement a cryptographic system that performs encryption and decryption using the following transposition techniques

1. Rail Fence Cipher
2. Row and Column (Matrix) Transposition Cipher

The system should take plaintext input and return the corresponding ciphertext for each cipher method and be able to reverse the process to recover the plaintext from the ciphertext.

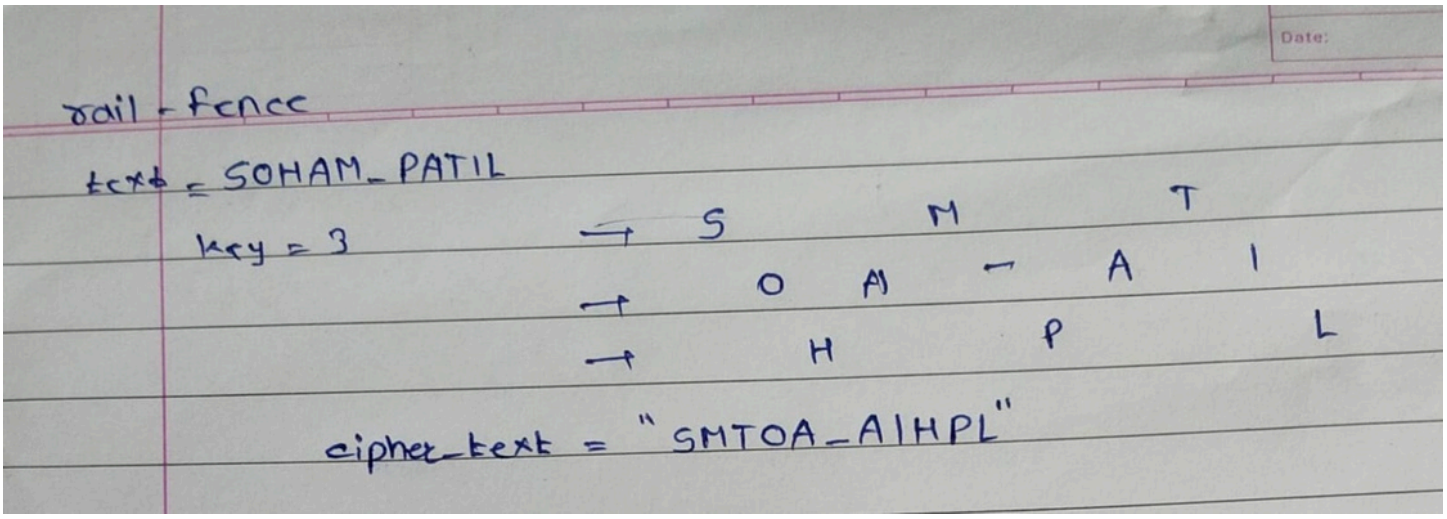
Theory:

Transposition Techniques

In transposition ciphers, the positions of the characters in the plaintext are shifted according to a defined permutation, without changing the actual characters. Unlike substitution ciphers, transposition ciphers rearrange the order of characters to obscure the message.

1. Rail Fence Cipher

- **Definition**: A form of transposition cipher where the plaintext is written diagonally over a set number of “rails” (rows) and then read row by row.
- **Encryption**:
  1. Choose the number of rails.
  2. Write the plaintext in a zigzag pattern down and up between rails.
  3. Read off row-wise to get the ciphertext.
- **Decryption**:
  1. Determine the zigzag pattern based on number of rails and length of ciphertext.
  2. Place characters in their positions and read along the zigzag to get plaintext.
- **Example**:Plaintext: HELLO WORLD with 3 rails → Ciphertext: HOLEL WRDLO.


rail-fence

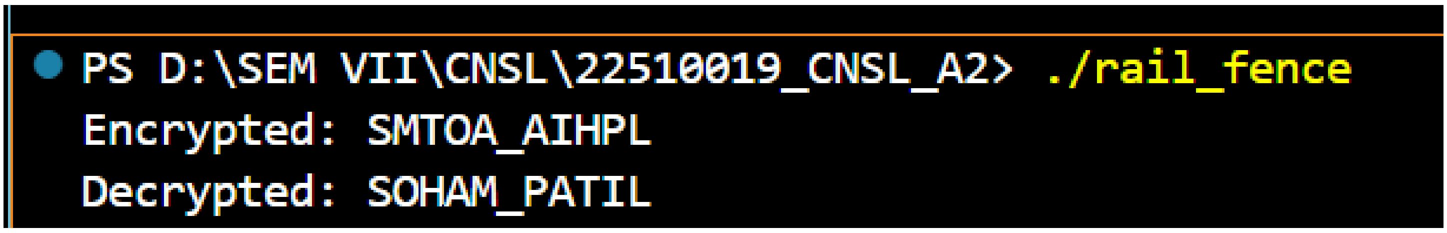
text = SOHAM_PATIL

key = 3

cipher_text = "SMTOA_AIHPL"

text) {

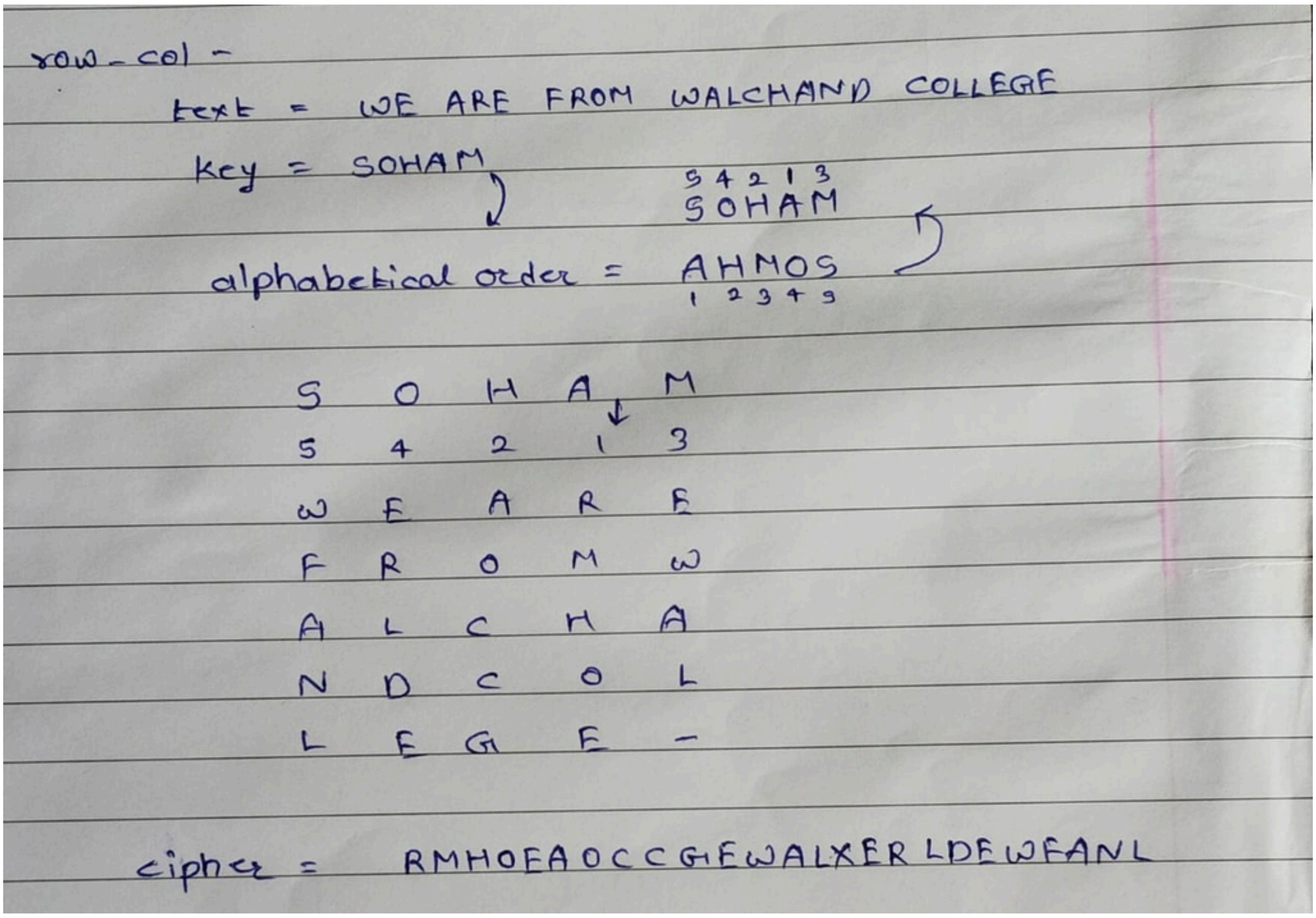
}


```
PS D:\SEM VII\CNSL\22510019_CNSL_A2> ./rail_fence
Encrypted: SMTOA_AIHPL
Decrypted: SOHAM_PATIL
```

```
#include <bits/stdc++.h>

using namespace std;

string railFenceEncrypt(string text, int key) {
    vector<vector<char>> rail(key, vector<char>(text.length(), '\n'));
    bool dir_down = false;
    int row = 0, col = 0;

    for (char ch : text) {
        if (row == 0 || row == key - 1)
            dir_down = !dir_down;

        rail[row][col++] = ch;

        row += dir_down ? 1 : -1;
    }

    string result;
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < (int)text.length(); j++) {
            if (rail[i][j] != '\n')
                result.push_back(rail[i][j]);
        }
    }
    return result;
}

string railFenceDecrypt(string cipher, int key) {
    vector<vector<char>> rail(key, vector<char>(cipher.length(), '\n'));

    bool dir_down = false;
    int row = 0, col = 0;
    for (int i = 0; i < (int)cipher.length(); i++) {
        if (row == 0) dir_down = true;
        if (row == key - 1) dir_down = false;

        rail[row][col++] = '*';
        row += dir_down ? 1 : -1;
    }

    int index = 0;
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < (int)cipher.length(); j++) {
            if (rail[i][j] == '*' && index < (int)cipher.length())
                rail[i][j] = cipher[index++];
        }
    }

    string result;
    row = 0; col = 0;
    for (int i = 0; i < (int)cipher.length(); i++) {
        if (row == 0) dir_down = true;
        if (row == key - 1) dir_down = false;

        if (rail[row][col] != '*')
            result.push_back(rail[row][col++]);

        row += dir_down ? 1 : -1;
    }

    return result;
}

int main() {
    string text = "HELLO_WORLD";
    int key = 3;

    string encrypted = railFenceEncrypt(text, key);
    cout << "Encrypted: " << encrypted << endl;

    string decrypted = railFenceDecrypt(encrypted, key);
    cout << "Decrypted: " << decrypted << endl;

    return 0;
}
```

2. Row and Column (Matrix) Transposition Cipher

- **Definition**: The plaintext is written in rows of a matrix of fixed size, then rearranged column-wise according to a key.
- **Encryption**:
  1. Write plaintext row-by-row in a matrix.
  2. Rearrange the columns based on numeric key order.
  3. Read column-by-column in the new order to get ciphertext.
- **Decryption**:
  1. Arrange ciphertext in columns according to the key order.
  2. Read row-by-row to get plaintext.
- **Example**:
- Plaintext: HELLOWORLD
- Key: 3142 → Ciphertext: (columns rearranged based on key sequence).


row - col -

text = WE ARE FROM WALCHAND COLLEGE

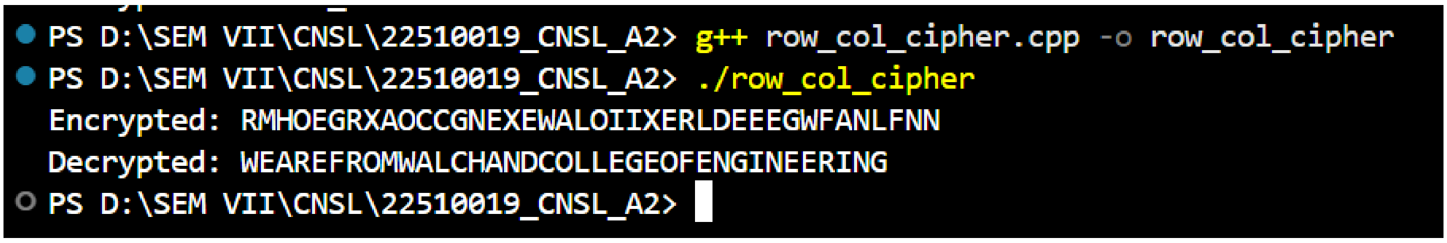
key = SOHAM

5 4 2 1 3
SOHAM

alphabetical order = AHMOS
1 2 3 4 5

S	O	H	A	M
5	4	2	1	3
W	E	A	R	E
F	R	O	M	W
A	L	C	H	A
N	D	C	O	L
L	E	G	E	-

cipher = RMHOEAOCCGEWALXERLDEWFANL



```
PS D:\SEM VII\CNSL\22510019_CNSL_A2> g++ row_col_cipher.cpp -o row_col_cipher
PS D:\SEM VII\CNSL\22510019_CNSL_A2> ./row_col_cipher
Encrypted: RMHOEGRXAOCCGNEXEWALOIIXERLDEEEGWFANLFNN
Decrypted: WEAREFROMWALCHANDCOLLEGEOFENGINEERING
PS D:\SEM VII\CNSL\22510019_CNSL_A2>
```

```cpp
#include <bits/stdc++.h>

using namespace std;


vector<int> createOrder(const string &key) {

    string sortedKey = key;

    sort(sortedKey.begin(), sortedKey.end());

    vector<int> order;


    for (char k : key) {
        order.push_back(find(sortedKey.begin(), sortedKey.end(), k) - sortedKey.begin() + 1);
    }
    return order;
}

string rowColumnEncrypt(string plaintext, const string &key) {
    int key_len = key.length();
    vector<int> order = createOrder(key);

    plaintext.erase(remove(plaintext.begin(), plaintext.end(), ' '), plaintext.end());

    int rows = plaintext.length() / key_len + (plaintext.length() % key_len != 0);

    vector<vector<char>> matrix(rows, vector<char>(key_len, 'X'));

    int index = 0;
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < key_len; c++) {
            if (index < plaintext.length()) {
                matrix[r][c] = plaintext[index++];
            }
        }
    }

    string ciphertext;
    for (int num = 1; num <= key_len; num++) {
        int col = find(order.begin(), order.end(), num) - order.begin();
        for (int r = 0; r < rows; r++) {
            ciphertext.push_back(matrix[r][col]);
        }
    }

    return ciphertext;
}
```

```
string rowColumnDecrypt(string ciphertext, const string &key) {
    int key_len = key.length();
    vector<int> order = createOrder(key);

    int rows = ciphertext.length() / key_len;
    vector<vector<char>> matrix(rows, vector<char>(key_len, '\0'));

    int index = 0;
    for (int num = 1; num <= key_len; num++) {
        int col = find(order.begin(), order.end(), num) - order.begin();
        for (int r = 0; r < rows; r++) {
            matrix[r][col] = ciphertext[index++];
        }
    }

    string plaintext;
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < key_len; c++) {
            plaintext.push_back(matrix[r][c]);
        }
    }

    while (!plaintext.empty() && plaintext.back() == 'X')
        plaintext.pop_back();

    return plaintext;
}

int main() {
    string key = "SOHAM";
    string plaintext = "WE ARE FROM WALCHAND COLLEGE OF ENGINEERING";

    string encrypted = rowColumnEncrypt(plaintext, key);
    cout << "Encrypted: " << encrypted << endl;

    string decrypted = rowColumnDecrypt(encrypted, key);
    cout << "Decrypted: " << decrypted << endl;

    return 0;
}
```

Advantages:

- Preserves frequency distribution but makes letter position unpredictable.
- Can be combined with substitution for higher security.

Procedure:

1. **Start the program**.
2. **Input**:
   - Plaintext string (convert to uppercase, remove spaces/punctuation).
   - For Rail Fence: number of rails.
   - For Row/Column: key indicating column permutation.
3. **Rail Fence Encryption**:
   - Create a 2D array for rails.
   - Fill it in zigzag pattern.
   - Read row-by-row to get ciphertext.
4. **Row and Column Encryption**:
   - Create matrix and fill row-by-row.
   - Rearrange columns based on key.
   - Read column-by-column to get ciphertext.
5. **Display ciphertext** for both methods.
6. **Rail Fence Decryption**:
   - Reconstruct zigzag pattern positions.
   - Fill ciphertext characters accordingly.
   - Read row-by-row along zigzag path.
7. **Row and Column Decryption**:
   - Reverse column rearrangement using key.
   - Read row-by-row to recover plaintext.
8. **Display decrypted text** and verify it matches the original plaintext.
9. **End program**.

Steps

 Start program execution.

□ Read plaintext input from user.

□ Ask for Rail Fence key (number of rails) and Row/Column key (numeric order).

□ Perform Rail Fence encryption → store ciphertext.

□ Perform Rail Fence decryption → check match with plaintext.

□ Perform Row and Column encryption → store ciphertext.

□ Perform Row and Column decryption → check match with plaintext.

□ Display all results.

□ End program execution.

Conclusions

- Both Rail Fence and Row/Column transposition ciphers successfully encrypted and decrypted messages, preserving exact characters but changing their positions.
- The decrypted texts matched the original plaintext, confirming the correctness of the algorithms.
- Rail Fence is simple and works by a zigzag writing pattern, while Row/Column transposition uses a matrix and key-based column permutation.
- Transposition alone does not disguise letter frequency, so it is often combined with substitution for stronger encryption.
- This experiment reinforced the concept of permutation-based cryptography and its implementation in programming.