Experiment no.: 05

Title: Implement rail fence technique & Simple columnar techniques.

Course Outcome: Use cryptography algorithms and protocols to achieve Computer Security

Theory:

1. Rail Fence Cipher

1.1 Introduction

The Rail Fence Cipher is a transposition cipher that encrypts text by rearranging the order of its characters. The message is written in a zigzag pattern across multiple rails (rows), and then read row by row to obtain the ciphertext.

It does not substitute characters, but rather changes their positions, which makes it simple yet effective for basic text scrambling.

1.2 Working Principle

1. Choose the number of rails (the key).
2. Write the plaintext in a zigzag fashion diagonally down and up the rails.
3. After writing the entire message, read the letters row by row to get the encrypted message (ciphertext).
4. To decrypt, the letters are arranged again in the same zigzag pattern and read diagonally to recover the original text.

1.3 Example

Let's take:

Plaintext: ADINAENCRYPTING

Key (Number of Rails): 3

Step 1: Write in Zigzag Form

R1	A				A				R				I		
R2		D		N		E		C		Y		T		N	
R3			I				N				P				G

Step 2: Read Row by Row

Reading each row sequentially gives us the ciphertext:

Ciphertext: AARIDNECYTNINPG

Step 3: Decryption Process

- Place the ciphertext letters back into the zigzag positions following the same rail pattern.
- Then read diagonally as per the writing order to obtain the original text: ADINAENCRYPTING

1.4 Advantages

- Simple and easy to understand.
- Demonstrates basic concept of transposition ciphers.
- Easy to implement in software and hardware.

1.5 Limitations

- Not secure for real communication, as it can be easily decrypted through pattern analysis.

- Frequency analysis or brute-force attack can reveal the message if the key is small.

2. Simple Columnar Cipher

2.1 Introduction

The Simple Columnar Cipher is another form of transposition cipher.
Instead of writing text diagonally like the Rail Fence, this method writes the message horizontally in rows and reads it vertically column by column based on a key.
The key determines the number of columns and their reading order.
It rearranges the characters of the plaintext, resulting in a permuted version of the original message.

2.2 Working Principle

1. Choose a key (a word or a number) which determines the column order.
2. Write the plaintext row-wise in a grid with columns equal to the length of the key.
3. Read the text column-wise according to the order of the key to get the ciphertext.
4. To decrypt, the text is filled back column-wise and then read row-wise.

2.3 Example

Let's take the same message:
Plaintext: ADINAENCRYPTING
Key: 3 1 4 2 (which represents column order)

Step 1: Write Text in Grid (based on key length = 4)

Column 1	Column 2	Column 3	Column 4
A	D	I	N
A	E	N	C
R	Y	P	T
I	N	G	

Step 2: Read According to Key Order (3 → 1 → 4 → 2)

- Column 3 → I N P G
- Column 1 → A A R I
- Column 4 → N C T _
- Column 2 → D E Y N

Now combine all the letters in this order:
Ciphertext: INPGAARINCTDEYN

Step 3: Decryption Process

- The ciphertext is written column by column following the key order (3 → 1 → 4 → 2).
- Once the grid is filled, read the letters row by row to recover the original plaintext.

→ Decrypted Text: ADINAENCRYPTING

2.4 Advantages

- Slightly more secure than the Rail Fence Cipher.
- Can handle longer messages easily.
- Flexible — key length can vary based on security needs.

2.5 Limitations

- Still vulnerable to frequency and pattern analysis.
- If the key is known, decryption is straightforward.
- Padding is sometimes needed to complete the rectangular grid.

Code: Rail Fence

```
#include <stdio.h>
#include <string.h>
void railFenceEncrypt(char *text, int key, char *result) {
   int len = strlen(text);
   char rail[key][len];
   memset(rail, '\n', sizeof(rail));
   int row = 0, dir_down = 0;
   for (int i = 0; i < len; i++) {
      rail[row][i] = text[i];
      if (row == 0 || row == key - 1)
         dir_down = !dir_down;
      row += dir_down ? 1 : -1;}
   int k = 0;
   for (int i = 0; i < key; i++)
      for (int j = 0; j < len; j++)
         if (rail[i][j] != '\n')
            result[k++] = rail[i][j];
   result[k] = '\0';}
void railFenceDecrypt(char *cipher, int key, char *result) {
   int len = strlen(cipher);
   char rail[key][len];
   memset(rail, '\n', sizeof(rail));
   int row = 0, dir_down = 0;
   for (int i = 0; i < len; i++) {
      rail[row][i] = '*';
      if (row == 0 || row == key - 1)
```

```
            dir_down = !dir_down;
        row += dir_down ? 1 : -1;}
    int k = 0;
    for (int i = 0; i < key; i++)
        for (int j = 0; j < len; j++)
            if (rail[i][j] == '*')
                rail[i][j] = cipher[k++];
    row = 0, dir_down = 0;
    for (int i = 0; i < len; i++) {
        result[i] = rail[row][i];
        if (row == 0 || row == key - 1)
            dir_down = !dir_down;
        row += dir_down ? 1 : -1;}
    result[len] = '\0';}
int main() {
    char text[] = "ADINAENCRYPTING";
    int key = 3;
    char encrypted[100], decrypted[100];
    railFenceEncrypt(text, key, encrypted);
    printf("Encrypted (Rail Fence): %s\n", encrypted);
    railFenceDecrypt(encrypted, key, decrypted);
    printf("Decrypted (Rail Fence): %s\n", decrypted);
    return 0;}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Adina> cd "c:\Users\Adina\OneDrive\Desktop\Computer Engineering
\SEM 5\CS\C program\" ; if ($?) { gcc RailFence.c -o RailFence } ; if ($?)
{ .\RailFence }
Encrypted (Rail Fence): AARIDNECYTNINPG
Decrypted (Rail Fence): ADINAENCRYPTING
PS C:\Users\Adina\OneDrive\Desktop\Computer Engineering\SEM 5\CS\C program>
```

Code: Simple Columnar

```
#include <stdio.h>
#include <string.h>
int main() {
    char plaintext[] = "ADINAENCRYPTING";
    int keyOrder[] = {3, 1, 4, 2};
    int keyLength = 4;
```

```
    char matrix[10][10];
    char ciphertext[50];
    int len = strlen(plaintext);
    int rows = len / keyLength;
    if (len % keyLength != 0)
        rows += 1;
    printf("Plaintext: %s\n", plaintext);
    printf("Key Order: 3 1 4 2\n");
    int index = 0;
    printf("\nStep 1: Write Text in Grid (based on key length = %d)\n",
keyLength);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < keyLength; j++) {
            if (index < len)
                matrix[i][j] = plaintext[index++];
            else
                matrix[i][j] = 'X';
            printf("%c ", matrix[i][j]);}
        printf("\n");}
    int k = 0;
    printf("\nStep 2: Read According to Key Order (3 → 1 → 4 → 2)\n");
    for (int orderIndex = 0; orderIndex < keyLength; orderIndex++) {
        int col = keyOrder[orderIndex] - 1;
        printf("Column %d ", keyOrder[orderIndex]);
        for (int i = 0; i < rows; i++) {
            ciphertext[k++] = matrix[i][col];
            printf("%c", matrix[i][col]);}
        printf("\n");}
    ciphertext[k] = '\0';
    printf("\nCiphertext: %s\n", ciphertext);
return 0;}
```

Output:

```
Plaintext: ADINAENCRYPTING
Key Order: 3 1 4 2

Step 1: Write Text in Grid (based on key length = 4)
A D I N
A E N C
R Y P T
I N G X

Step 2: Read According to Key Order (3 - 1 - 4 - 2)
Column 3 INPG
Column 1 AARI
Column 4 NCTX
Column 2 DEYN

Ciphertext: INPGAARINCTXDEYN
PS C:\Users\Adina\OneDrive\Desktop\Computer Engineering\SEM 5\CS\C program>
```

Conclusion:

I have successfully implemented rail fence technique & Simple columnar techniques.