Name : Ritesh Pawar
PRN : 2020BTECS00068

# *Assignment No 2*

# *Cryptography and Network Security Lab (5CS453)*

**Name: Ritesh Pawar**

**PRN: 2020BTECS00068**

## Title:

**Encryption and Decryption using Transposition Cipher Technique.**

## Aim:

**To Study and Implement Encryption and Decryption using Rail Fence Transposition Cipher Technique and Columnar Transposition Cipher Technique**

## Theory:

1. **Rail Fence Transposition Cipher Technique**

The Rail Fence Transposition Cipher, also known as the Zigzag Cipher, is a simple columnar transposition cipher technique.

It involves arranging the plaintext characters in a zigzag pattern across multiple rows, known as "rails," and then reading them off row by row to create the encrypted message.

While this cipher is easy to understand and implement, it lacks strong security and is mainly used for educational purposes or simple puzzles.

**Encryption:**

- Choose the number of rails (rows) for the zigzag pattern.
- Write the message diagonally across the rails, moving up and down.
- Read the characters row by row to form the encrypted message.

**Decryption:**

- Create the zigzag pattern with the chosen number of rails.
- Leave blank spaces in the pattern for characters to be placed.
- Fill in the blanks with the encrypted characters, row by row.
- Read the characters diagonally to retrieve the original message.

**Advantages**:

- Easy to understand and implement.
- Provides basic encryption and breaks up character repetition.

Name : Ritesh Pawar
PRN : 2020BTECS00068

**Disadvantages:**

- Not secure against modern cryptanalysis.
- Security depends on the number of rails, making it less practical for strong encryption.

**Limitations :**

**Low Security**: The Rail Fence Cipher is not very secure and can be easily cracked using basic cryptanalysis techniques. It provides minimal protection against modern encryption-breaking methods.

**Predictable Pattern**: The encryption pattern is easily recognizable, making it susceptible to pattern analysis. An attacker can often discern the fact that a Rail Fence Cipher has been used, and this knowledge can simplify decryption.

**Limited Key Space**: The key space of the Rail Fence Cipher is limited to the number of rails or rows used for encryption. Once an attacker identifies the number of rails, they can easily decrypt the message.

**Difficulty with Long Messages**: For long messages, the Rail Fence Cipher can become unwieldy, as the number of rails must be specified. If you don't know the correct number of rails, decryption becomes challenging.

**Data Loss and Padding**: To create a perfect zigzag pattern, you may need to add padding characters to the plaintext. These padding characters must be removed during decryption, which can complicate the process.

**CODE**:

```
#include <iostream>

#include <string>

#include <vector>



using namespace std;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
// Function to encrypt a message using the
Rail Fence Cipher
string encryptRailFence(const string&
message, int rails) {
vector<string> fence(rails, "");
int currentRail = 0;
bool goingDown = false;


for (char c : message) {
fence[currentRail] += c;


if (currentRail == 0 || currentRail == rails
- 1) {
```

```
goingDown = !goingDown;
}


currentRail += goingDown ? 1 : -1;
}
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
string encryptedMessage = "";

for (const string& rail : fence) {

encryptedMessage += rail;

}


return encryptedMessage;

}


// Function to decrypt a message encrypted
with the Rail Fence Cipher

string decryptRailFence(const string&
message, int rails) {

vector<string> fence(rails, "");

vector<int> railSizes(rails, 0);
```

```
int currentRail = 0;

bool goingDown = false;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
for (int i = 0; i < message.length(); i++) {

railSizes[currentRail]++;


if (currentRail == 0 || currentRail == rails
- 1) {

goingDown = !goingDown;

}


currentRail += goingDown ? 1 : -1;

}


int messageIndex = 0;

for (int rail = 0; rail < rails; rail++) {

fence[rail] = message.substr(messageIndex,
railSizes[rail]);

messageIndex += railSizes[rail];

}
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
string decryptedMessage = "";
currentRail = 0;
goingDown = false;

for (int i = 0; i < message.length(); i++) {
decryptedMessage += fence[currentRail][0];
fence[currentRail].erase(0, 1);

if (currentRail == 0 || currentRail == rails
- 1) {
goingDown = !goingDown;
}

currentRail += goingDown ? 1 : -1;
}

return decryptedMessage;
}
```

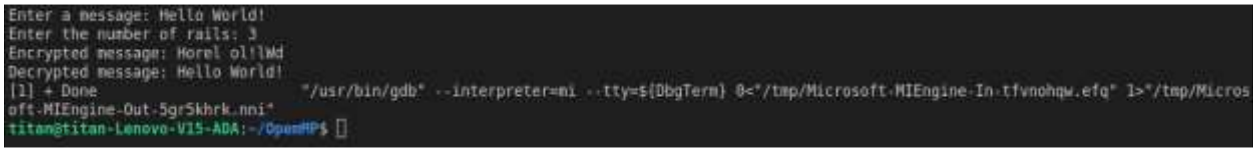
Name : Ritesh Pawar
PRN : 2020BTECS00068

```
int main() {

string message;

int rails;


cout << "Enter a message: ";

getline(cin, message);


cout << "Enter the number of rails: ";

cin >> rails;


string encryptedMessage =
encryptRailFence(message, rails);

string decryptedMessage =
decryptRailFence(encryptedMessage, rails);


cout << "Encrypted message: " <<
encryptedMessage << endl;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
cout << "Decrypted message: " <<
decryptedMessage << endl;


return 0;

}
```

**OUTPUT:**



```
Enter a message: Hello World!
Enter the number of rails: 3
Encrypted message: Horel ol!lWd
Decrypted message: Hello World!
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-tfvnohqw.efq" 1>"/tmp/Micros
oft-MIEngine-Out-5gr5khrk.nni"
titan@titan-Lenovo-V15-ADA:~/OpenMP$ 
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

## 2. Columnar Transposition Cipher Technique

The Columnar Transposition Cipher is a more advanced transposition cipher technique that involves reordering the characters of a message based on a chosen keyword or key phrase.

It provides a higher level of security compared to simpler ciphers like the Rail Fence Cipher. Here's how the Columnar Transposition Cipher works:

**Encryption:**

- Choose a keyword or key phrase. The unique characters of the keyword determine the order of columns in the transposition grid.
- Write the message row by row into a grid, using the keyword to determine the order of columns.
- Read the characters column by column to obtain the encrypted message.

**Decryption:**

- Use the keyword to determine the order of columns in the transposition grid.
- Write the encrypted message into the grid column by column.
- Read the characters row by row to retrieve the original plaintext.

**Advantages:**

- Offers stronger security compared to simpler ciphers.
- Security depends on the length and uniqueness of the keyword.

**Disadvantages**:

- Can be vulnerable to attacks if the keyword is short or easily guessed.
- May require additional padding characters for messages that don't fit evenly into the grid.

**Limitations :**

**Frequency Analysis**: The Columnar Transposition Cipher retains the frequency distribution of letters in the plaintext. Attackers can still perform frequency analysis to guess the key length or potentially discover patterns within the ciphertext.

**Known-Plaintext Attack**: If an attacker has access to some portions of the plaintext and the corresponding ciphertext, they can use this information to make educated guesses about the key or deduce parts of the message, potentially compromising security.

**Key Length Requirement**: Choosing an appropriate key length can be challenging. A too-short key may not provide sufficient security, while a too-long key can make the process cumbersome, especially if you have to remember or share it securely.

Name : Ritesh Pawar
PRN : 2020BTECS00068

**Limited Complexity:** The Columnar Transposition Cipher, like other transposition ciphers, provides relatively low complexity in terms of encryption, making it vulnerable to automated attacks and modern cryptographic techniques.

**No Substitution of Characters**: Unlike substitution ciphers (e.g., Caesar Cipher), the Columnar Transposition Cipher does not alter the characters themselves, making it susceptible to known-plaintext attacks and character frequency analysis.

**CODE**:

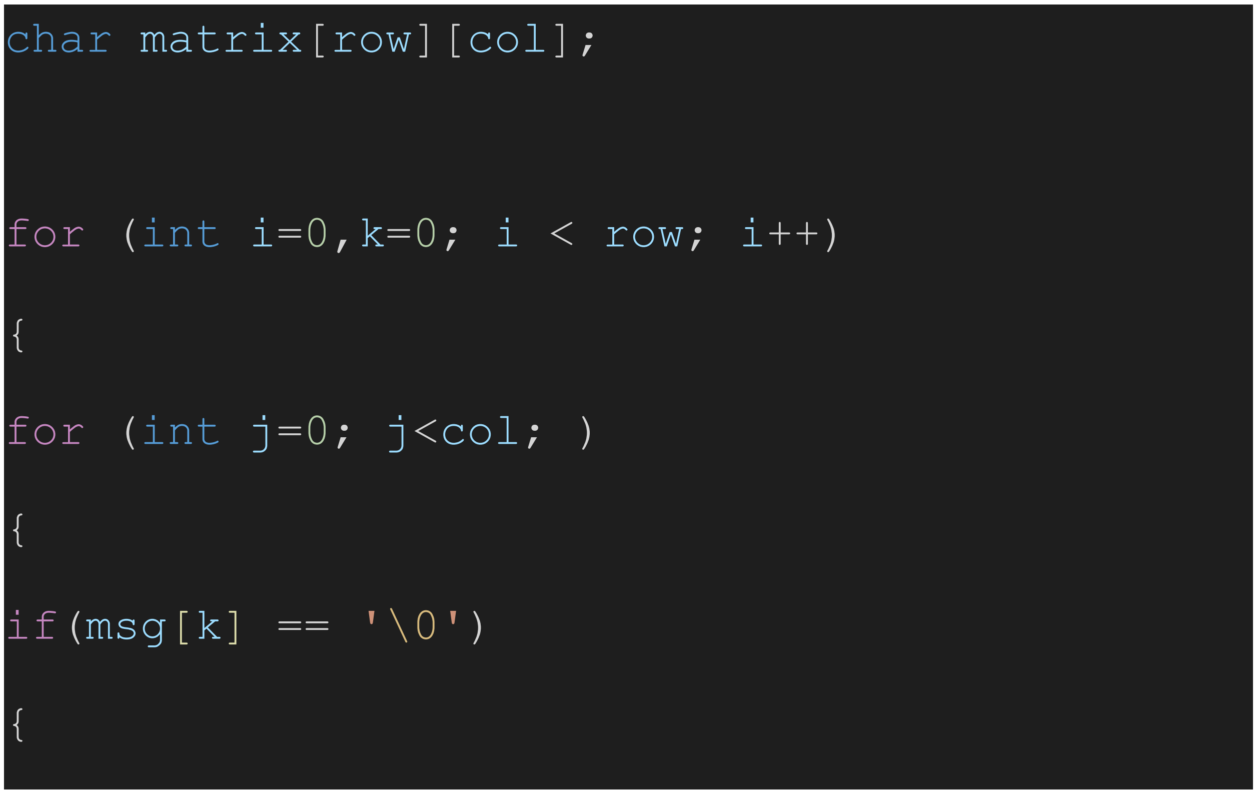
```
// CPP program for illustrating

// Columnar Transposition Cipher

#include<bits/stdc++.h>

using namespace std;


// Key for Columnar Transposition

string const key = "HACK";

map<int,int> keyMap;


void setPermutationOrder()

{

// Add the permutation order into map

for(int i=0; i < key.length(); i++)
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

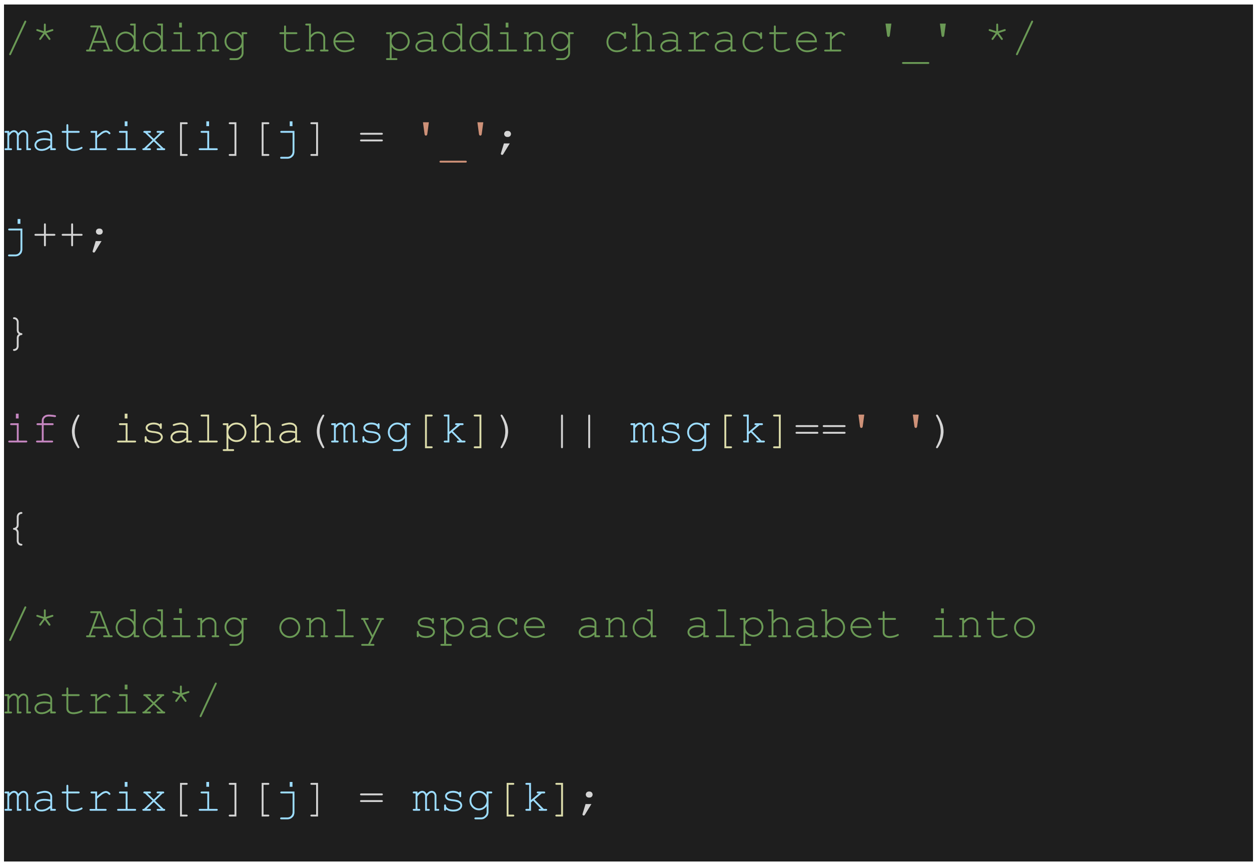
```
{

keyMap[key[i]] = i;

}

}



// Encryption
```

```
string encryptMessage(string msg)

{

int row,col,j;

string cipher = "";

/* calculate column of the matrix*/

col = key.length();

/* calculate Maximum row of the matrix*/

row = msg.length()/col;

if (msg.length() % col)

row += 1;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068



```
char matrix[row][col];


for (int i=0,k=0; i < row; i++)

{

for (int j=0; j<col; )

{

if(msg[k] == '\0')

{
```



```
/* Adding the padding character '_' */

matrix[i][j] = '_';

j++;

}

if( isalpha(msg[k]) || msg[k]==' ')

{

/* Adding only space and alphabet into
matrix*/

matrix[i][j] = msg[k];
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
j++;

}

k++;

}

}



for (map<int,int>::iterator ii =

keyMap.begin(); ii!=keyMap.end(); ++ii)

{

j=ii->second;
```

```
// getting cipher text from matrix column

wise using permuted key

for (int i=0; i<row; i++)

{

if( isalpha(matrix[i][j]) || matrix[i][j]=='

' || matrix[i][j]=='_')

cipher += matrix[i][j];

}
```

```
}


return cipher;

}


// Decryption

string decryptMessage(string cipher)

{

/* calculate row and column for cipher Matrix
*/

int col = key.length();
```

```
int row = cipher.length()/col;

char cipherMat[row][col];


/* add character into matrix column wise */

for (int j=0,k=0; j<col; j++)
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
for (int i=0; i<row; i++)

cipherMat[i][j] = cipher[k++];



/* update the order of key for decryption */

int index = 0;

for( map<int,int>::iterator

ii=keyMap.begin(); ii!=keyMap.end(); ++ii)

ii->second = index++;



/* Arrange the matrix column wise according

to permutation order by adding into new

matrix */

char decCipher[row][col];

map<int,int>::iterator ii=keyMap.begin();

int k = 0;
```

```
for (int l=0,j; key[l]!='\0'; k++)

{

j = keyMap[key[l++]];
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
for (int i=0; i<row; i++)

{

decCipher[i][k]=cipherMat[i][j];

}

}



/* getting Message using matrix */

string msg = "";

for (int i=0; i<row; i++)

{

for(int j=0; j<col; j++)

{

if(decCipher[i][j] != '_')

msg += decCipher[i][j];

}

}
```

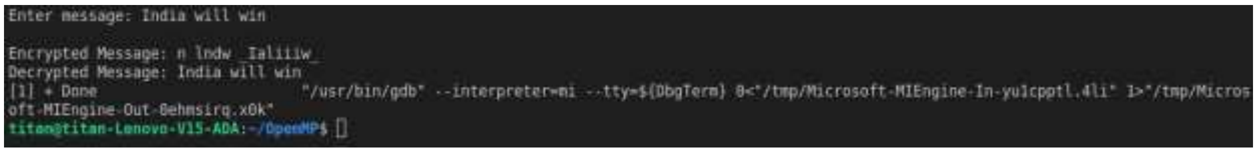
```
return msg;
```

```
}


// Driver Program
int main(void)
{
/* message */
string msg;
cout<<"Enter message: ";
getline(cin, msg);
setPermutationOrder();
// Calling encryption function
string cipher = encryptMessage(msg);
cout << "\nEncrypted Message: " << cipher <<
endl;
// Calling Decryption function
cout << "Decrypted Message: " <<
decryptMessage(cipher) << endl;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
return 0;

}
```

**OUTPUT:**



```
Enter message: India will win

Encrypted Message: n lndw _Ialiiiw_
Decrypted Message: India will win
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-yu1cpptl.4li" 1>"/tmp/Micros
oft-MIEngine-Out-8ehmsirq.x0k"
titan@titan-Lenovo-V15-ADA:~/OpenMP$ []
```