Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4

Practical No – 4

Title: To implement transposition cipher.

a. Railfence cipher
b. Columnar cipher

Theory:

a. Railfence Cipher

→

The rail fence cipher (sometimes called zigzag cipher) is a transposition cipher that jumbles up the order of the letters of a message using a basic algorithm. The rail fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn.

Example:

Let's consider the plaintext "This is a secret message".

Plaintext T H I S I S A S E C R E T M E S S A G E

To encode this message we will first write over two lines (the "rails of the fence") as follows:

Rail Fence Encoding																			
	T		I		I		A		E		R		T		E		S		G	
		H		S		S		S		C		E		M		S		A		E

Note that all white spaces have been removed from the plain text.

The **ciphertext** is then read off by writing the top row first, followed by the bottom row:

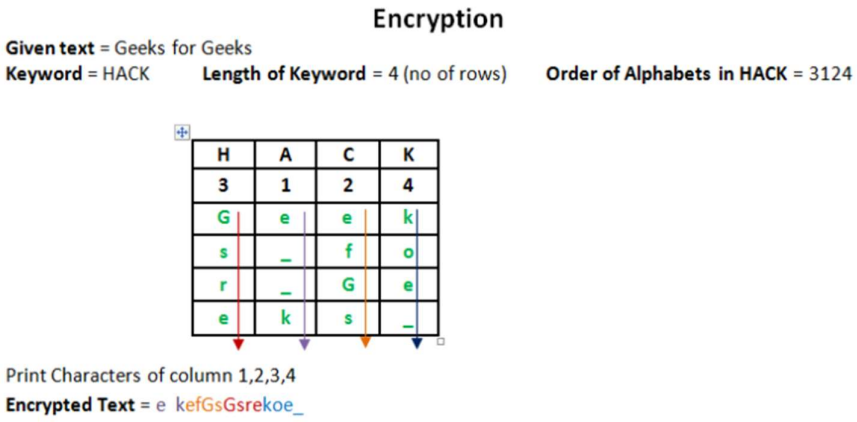
Ciphertext T I I A E R T E S G H S S S C E M S A E

b. Columnar Cipher

→

The columnar transposition cipher is a fairly simple, easy to implement cipher. It is a transposition cipher that follows a simple rule for mixing up the characters in the plaintext to form the ciphertext.

Although weak on its own, it can be combined with other ciphers, such as a substitution cipher, the combination of which can be more difficult to break than either cipher on its own. The ADFGVX cipher uses a columnar transposition to greatly improve its security.



Code Snapshots:

a. Railfence Cipher

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int t,n,m,i,j,k,sum=0;
    string s;
    cout<<"Enter the message: ";
    cin>>s;
    cout<<"Enter key: ";
    cin>>n;

     cout<<"Enter your choice."<<endl;
    cout<<"1. Encryption\n";
    cout<<"2. Decryption\n";
    cout<<"Your choice: ";
    int choice;
    cin>>choice;

    if(choice==1)
    {
    vector<vector<char>> a(n,vector<char>(s.size(),' '));
    j=0;
    int flag=0;
    for(i=0;i<s.size();i++){
        a[j][i] = s[i];
         if(j==n-1){
```

```
            flag=1;
        }
        else if(j==0)
            flag=0;
        if(flag==0){
            j++;
        }
        else j--;
    }
    for(i=0;i<n;i++){
        for(j=0;j<s.size();j++){
            if(a[i][j]!=' ')
                cout<<a[i][j];
        }
    }
    cout<<'\n';
    }
    if(choice==2)
    {
        vector<vector<char>> a(n,vector<char>(s.size(),' '));
    j=0;
    int flag=0;
    for(i=0;i<s.size();i++){
        a[j][i] = '0';
         if(j==n-1){
            flag=1;
        }
        else if(j==0)
            flag=0;
        if(flag==0){
            j++;
        }
        else j--;
    }
    int temp =0;
    for(i=0;i<n;i++){
        for(j=0;j<s.size();j++){
                if(a[i][j]=='0')
                    a[i][j]= s[temp++];
        }
    }
    flag=0;
    j=0;
    for(i=0;i<s.size();i++){
        cout<<a[j][i];
         if(j==n-1){
            flag=1;
        }
```

```
        else if(j==0)
            flag=0;
        if(flag==0){
            j++;
        }
        else j--;
    }
    cout<<'\n';
    }
    return 0;
}
```

b. Columnar Cipher

```
// CPP program for illustrating
// Columnar Transposition Cipher
#include<bits/stdc++.h>
using namespace std;

// Key for Columnar Transposition
string const key = "STAR";
map<int,int> keyMap;

void setPermutationOrder()
{
    // Add the permutation order into map
    for(int i=0; i < key.length(); i++)
    {
        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;
```

```
    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];
                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++)
        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++]];
        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 = "THISISASECRETMESSAGE";

    setPermutationOrder();

    // Calling encryption function
    string cipher = encryptMessage(msg);
    cout << "Encrypted Message: " << cipher << endl;
```

```
    // Calling Decryption function
    cout << "Decrypted Message: " << decryptMessage(cipher) << endl;

    return 0;
}
```

Output Snapshots:

a. Railfence CIpher

Encryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ RailFenceED.cpp -o RailFenceED } ; if ($
?) { .\RailFenceED }
Enter the message: THISISASECRETMESSAGE
Enter key: 3
Enter your choice.
1. Encryption
2. Decryption
Your choice: 1
TIETSHSSSCEMSAEIAREG
```

Decryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ RailFenceED.cpp -o RailFenceED } ; if ($
?) { .\RailFenceED }
Enter the message: TIETSHSSSCEMSAEIAREG
Enter key: 3
Enter your choice.
1. Encryption
2. Decryption
Your choice: 2
THISISASECRETMESSAGE
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

b. Columnar Cipher

Encryption And Decryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ ColumnarED.cpp -o ColumnarED } ; if ($?)
 { .\ColumnarED }
Encrypted Message: IAREGSSESETIETSHSCMA
Decrypted Message: THISISASECRETMESSAGE
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Conclusion:

a. Railfence Cipher

- 1. The Rail Fence algorithm is a simple cryptography algorithm. However, it is not secure.
- The key is how many rows is implemented. It can be guessed by making a brute-force attack.

b. Columnar Cipher

- The Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher.
- Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.