Cryptography and Network Security Lab

PRN: 2019BTECS00090

Name: Udaykumar Gadikar

Batch: B8

Assignment No. 3

Title:

Playfair Cipher

Aim:

To Perform Playfair Cipher

Theory:

The Playfair cipher or Playfair square or Wheatstone–Playfair cipher is a manual symmetric encryption technique and was the first literal digram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone, but bears the name of Lord Playfair for promoting its use.

The technique encrypts pairs of letters (bigrams or digrams), instead of single letters as in the simple substitution cipher and rather more complex Vigenère cipher systems then in use. The Playfair is thus significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it. The frequency analysis of bigrams is possible, but considerably more difficult. With 600[1] possible bigrams rather than the 26 possible monograms (single symbols, usually letters in this context), a considerably larger cipher text is required in order to be useful.

Procedure:

Note: We treat I and J as the same letter

1) First, remove spaces & split the plaintext into digraphs (pair of two letters). If any letter appears twice (side by side), push X at the place of the second occurrence.
2) After that, if the plaintext has the odd number of letters, append the letter X at the end of the plaintext. It makes the plaintext of even.
3) To determine the cipher (encryption) text, first, build a 5*5 key-matrix or key-table and filled it with the letters of alphabets, as directed below:

   Fill the first row (left to right) with the letters of the given keyword. If the keyword has duplicate letters (if any) avoid them. It means a letter will be considered only once. After that, fill the remaining letters in alphabetical order. Let's create a 5*5 key-matrix for the keyword

   There may be the following three conditions:

   a) If a pair of letters (digraph) appears in the same row

   In this case, replace each letter of the digraph with the letters immediately to their right. If there is no letter to the right, consider the first letter of the same row as the right letter.

   b) If a pair of letters (digraph) appears in the same column

   In this case, replace each letter of the digraph with the letters immediately below them. If there is no letter below, wrap around to the top of the same column.

   c) If a pair of letters (digraph) appears in a different row and different column

   In this case, select a 3*3 matrix from a 5*5 matrix such that pair of letters appears in the 3*3 matrix. Since they occupy two opposite corners of a square within the matrix, the other corner will be a cipher for the given digraph.

The decryption procedure is the same as encryption but the steps are applied in reverse order. For decryption cipher is symmetric (move left along rows and up along columns). The receiver of the plain text has the same key and can create the same key-table that is used to decrypt the message.

Code:

```
#include <bits/stdc++.h>
using namespace std;

class PlayFair
{
    vector<vector<char>> table;

public:
    PlayFair(string key)
    {
        this->table = tableCreation(key);
        cout << endl;
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                cout << table[i][j] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }

    vector<vector<char>> tableCreation(string key)
    {
        vector<bool> arr(26, true);

        vector<vector<char>> myTable(5, vector<char>(5, ' '));
        int x = 0, y = 0;

        for (int i = 0; i < key.length(); i++)
        {
            if (arr[key[i] - 'A'] && key[i] != 'I')
            {
                myTable[x][y] = key[i];
                y++;
                if (y == 5)
                {
                    x++;
                    y = 0;
                }
                arr[key[i] - 'A'] = false;
            }
        }
```

```
    for (int i = 0; i < 26; i++)
    {
        if (arr[i] && i != 8)
        {
            myTable[x][y] = 'A' + i;
            y++;
            if (y == 5)
            {
                x++;
                y = 0;
            }
        }
    }

    return myTable;
}

string findEncryption(char first, char second)
{
    pair<int, int> pos1;
    pair<int, int> pos2;

    string text = "";
    for (int i = 0; i < table.size(); i++)
    {
        for (int j = 0; j < table[0].size(); j++)
        {
            if (table[i][j] == first)
            {
                pos1 = {i, j};
            }
            if (table[i][j] == second)
            {
                pos2 = {i, j};
            }
        }
    }

    if (pos1.first == pos2.first)
    {
        text += table[pos1.first][(pos1.second + 1) % 5];
        text += table[pos1.first][(pos2.second + 1) % 5];
        return text;
    }
    if (pos1.second == pos2.second)
    {
        text += table[(pos1.first + 1) % 5][pos1.second];
```

```
            text += table[(pos2.first + 1) % 5][pos2.second];
            return text;
        }
        int col = pos1.second;
        while (col != pos2.second)
        {
            col = (col + 1) % 5;
        }
        text += table[pos1.first][col];
        col = pos2.second;
        while (col != pos1.second)
        {
            col = (col + 1) % 5;
        }
        text += table[pos2.first][col];

        return text;
    }

    string findDecryption(char first, char second)
    {
        pair<int, int> pos1;
        pair<int, int> pos2;

        string text = "";
        for (int i = 0; i < table.size(); i++)
        {
            for (int j = 0; j < table[0].size(); j++)
            {
                if (table[i][j] == first)
                {
                    pos1 = {i, j};
                }
                if (table[i][j] == second)
                {
                    pos2 = {i, j};
                }
            }
        }

        if (pos1.first == pos2.first)
        {
            text += table[pos1.first][(5 + (pos1.second - 1)) % 5];
            text += table[pos1.first][(5 + (pos2.second - 1)) % 5];
            return text;
        }
        if (pos1.second == pos2.second)
        {
```

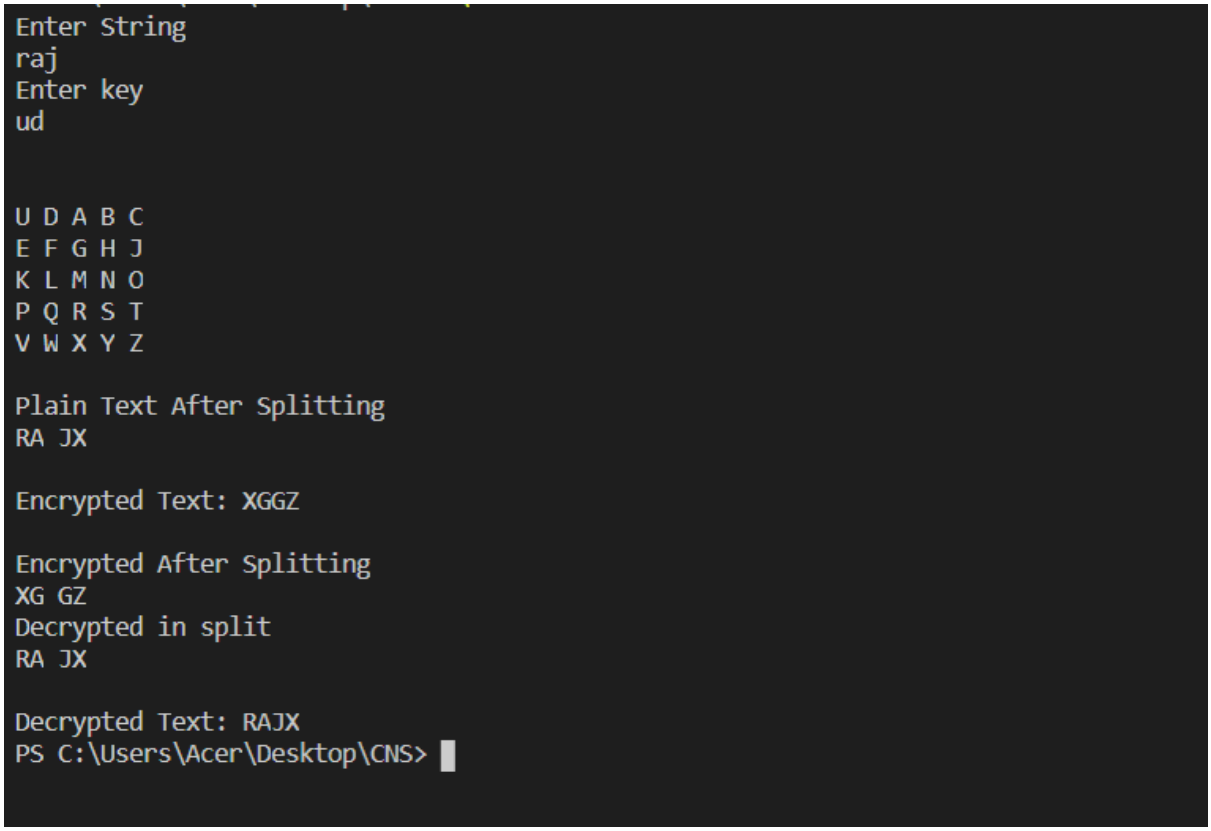
```
            text += table[(5 + pos1.first - 1) % 5][pos1.second];
            text += table[(5 + pos2.first - 1) % 5][pos2.second];
            return text;
        }
        int col = pos1.second;
        while (col != pos2.second)
        {
            col = (col + 1) % 5;
        }
        text += table[pos1.first][col];
        col = pos2.second;
        while (col != pos1.second)
        {
            col = (col + 1) % 5;
        }
        text += table[pos2.first][col];

        return text;
    }

    string encrypt(string plainText)
    {

        string str = "";

        for (int i = 0; i < plainText.length(); i++)
        {
            if (plainText[i] != ' ')
            {
                str += plainText[i];
            }
        }
        plainText = str;
        int n = plainText.length();

        //splitting into two chars
        int i = 0;
        vector<pair<char, char>> groups;
        while (i < n)
        {
            if (i != n - 1 && plainText[i] != plainText[i + 1])
            {
                groups.push_back({plainText[i], plainText[i + 1]});
                i += 2;
            }
            else
            {
                groups.push_back({plainText[i], 'X'});
```

```
                i += 1;
            }
        }
        cout << "Plain Text After Splitting" << endl;

        string encryptedString = "";

        for (i = 0; i < groups.size(); i++)
        {
            cout << groups[i].first << groups[i].second << " ";

            encryptedString += findEncryption(groups[i].first, groups[i].secon
d);
        }
        cout << endl
             << endl;

        return encryptedString;
    }

    string decrypt(string encrypted)
    {
        int n = encrypted.length();

        //splitting into two chars
        int i = 0;
        vector<pair<char, char>> groups;
        while (i < n)
        {
            groups.push_back({encrypted[i], encrypted[i + 1]});
            i += 2;
        }
        cout << "Encrypted After Splitting" << endl;

        string decryptedString = "";

        for (i = 0; i < groups.size(); i++)
        {
            cout << groups[i].first << groups[i].second << " ";
        }
        cout << endl;
        cout << "Decrypted in split" << endl;

        for (i = 0; i < groups.size(); i++)
        {
            string text = findDecryption(groups[i].first, groups[i].second);
            cout << text << " ";
            decryptedString += text;
```

```
        }
        cout << endl
             << endl;

        return decryptedString;
    }
};
int main()
{
    string input;
    cout << "Enter String" << endl;
    getline(cin, input);
    cout << "Enter key" << endl;
    string key;
    cin >> key;

    for (int i = 0; i < input.length(); i++)
    {
        input[i] = toupper(input[i]);

        if (input[i] == 'I')
        {
            input[i] = 'J';
        }
    }
    cout << endl;
    for (int i = 0; i < key.length(); i++)
    {
        key[i] = toupper(key[i]);
    }
    PlayFair cipher(key);

    string encryptedText = cipher.encrypt(input);
    cout << "Encrypted Text: " << encryptedText << endl
         << endl;
    ;
    string decryptedText = cipher.decrypt(encryptedText);
    cout << "Decrypted Text: " << decryptedText << endl;

    return 0;
}
```

Output:

Console:



Conclusion:

In comparison to mono alphabetic cipher, the Playfair cipher (poly alphabetic cipher) is more secure, because in mono alphabetic cipher the attacker has to search for 26 letters, while in poly alphabetic cipher attacker has to search for 26^2 (676) digraphs.