**Experiment-6**

**Aim:** Study and Implement a program for 5\*5 Playfair Cipher.

**Introduction:**

The Playfair cipher or Playfair square or Wheatstone–Playfair cipher is a manual symmetric encryption technique and was the first literal digraph substitution cipher.

The technique encrypts pairs of letters (digraph) 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.

**Method of Encryption and Decryption:**

**1) Generate playfair matrix (5X5):** The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I. The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

**2) Generate Digraph:** The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a X is added to the last letter. Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter.

**3)** **Encryption:**

Take two letters at a time from digraph.

Case 1: Both are in same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

Case 2: Both are in same column: Take the letter below each one (going back to the top if at the bottom).

Case 3: Different row and column: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

**4) Decryption:**

Take two letters at a time from digraph.

In case 1: instead of going right, go left.

In case 2: instead of going top to bottom, go bottom to top.

Case 3 remains the same.

**Program (Source Code):**

#include <bits/stdc++.h>

using namespace std;

string encrypt(string plainText, string key){

    string encryptedText;

    //make a matrix of 5\*5 and fill it with '-'

    char matrix[5][5];

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            matrix[i][j] = '-';

        }

    }

    //make a hash array for all alphabets

    int hashArray[26] = {0};

    //mark j in hash array, as we will not use it. (we use i instead of j)

    int temp = 'j' - 'a';

    hashArray[temp] = 1;

    //fill the matrix with key

    //when filling a character, mark it in the hash array so that we don't insert it again afterwards

    int keyPointer = 0;

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            //to stop filling once whole key is inserted, we put condition

            while (keyPointer < key.length()){

                //check if key is repeated or not

                if (hashArray[int(key[keyPointer]) - int('a')] == 0){

                    matrix[i][j] = key[keyPointer];

                    hashArray[int(key[keyPointer]) - int('a')] = 1;

                    keyPointer++;

                    break;  //means break current iteration and do next one

                }

                else{

                    keyPointer++;

                }

            }

        }

    }

    //filling the rest of the matrix alphabet wise, with no character repeat

    string alphabet = "abcdefghijklmnopqrstuvwxyz";

    int alphabetPointer = 0;

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            //to reach the unfilled values

            if (matrix[i][j] == '-'){

                while (alphabetPointer < alphabet.length()){

                    if (hashArray[int(alphabet[alphabetPointer]) - int('a')] == 0){

                        matrix[i][j] = alphabet[alphabetPointer];

                        hashArray[int(alphabet[alphabetPointer]) - int('a')] = 1;

                        alphabetPointer++;

                        break;

                    }

                    else{

                        alphabetPointer++;

                    }

                }

            }

        }

    }

    // //printing the matrix

    // cout<<"\n";

    // for (int i=0;i<5;i++){

    //     for (int j=0;j<5;j++){

    //         cout<<matrix[i][j]<<" ";

    //     }

    //     cout<<"\n";

    // }

    // cout<<"\n";

    //replacing all 'j' in plainText by 'i'

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

        if (plainText[i] == 'j'){

            plainText[i] = 'i';

        }

    }

    //making the plainText (adding the bogus letter 'x' wherever two letters are same in pair)

    string actualText;

    int noOfxAdded = 0;     //used later

    int plainTextPointer = 0;

    while ((plainTextPointer+1) < plainText.length()){

        if (plainText[plainTextPointer] != plainText[plainTextPointer+1]){

            actualText += plainText[plainTextPointer];

            actualText += plainText[plainTextPointer+1];

            plainTextPointer = plainTextPointer+2;

        }

        else{

            actualText += plainText[plainTextPointer];

            actualText += 'x';

            noOfxAdded++;

            plainTextPointer++;

        }

    }

    //checking if last character needs to be added or not (cuz it is not added in the above steps sometimes, due to range issuess)

    if ((plainText.length() + noOfxAdded) != actualText.length()){

        actualText += plainText[plainText.length()-1];

    }

    //adding a bogus 'z' if length is odd

    if (actualText.length() % 2 != 0){

        actualText += 'z';

    }

    //plainText is updated and ready to be encrypted

    //form the encrypted text by finding pairs in matrix and replacing them

    for (int i=0;i<actualText.length();i=i+2){

        char a = actualText[i];

        char b = actualText[i+1];

        //position of actual plain text characters

        int row\_a=0;

        int col\_a=0;

        int row\_b=0;

        int col\_b=0;

        //encrypted characters

        char ch1;

        char ch2;

        for (int j=0;j<5;j++){

            for (int k=0;k<5;k++){

                if (matrix[j][k] == a){

                    row\_a = j;

                    col\_a = k;

                }

                else if (matrix[j][k] == b){

                    row\_b = j;

                    col\_b = k;

                }

            }

        }

        //if both characters are in same row

        if (row\_a == row\_b){

            col\_a = (col\_a + 1)%5;

            col\_b = (col\_b + 1)%5;

        }

        //if both characters are in same column

        else if (col\_a == col\_b){

            row\_a = (row\_a + 1)%5;

            row\_b = (row\_b + 1)%5;

        }

        //if both are in different rows and columns

        else{

            //just interchange the col values

            int tempCol = col\_a;

            col\_a = col\_b;

            col\_b = tempCol;

        }

        ch1 = matrix[row\_a][col\_a];

        ch2 = matrix[row\_b][col\_b];

        encryptedText += ch1;

        encryptedText += ch2;

    }

    return encryptedText;

}

//similar but just opposite of encryption method.

//form the key matrix which will be the same.

//difference is only in forming the decryptedText.

string decrypt(string encryptedText, string key){

    string decryptedText;

    //make a matrix of 5\*5 and fill it with '-'

    char matrix[5][5];

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            matrix[i][j] = '-';

        }

    }

    //make a hash array for all alphabets

    int hashArray[26] = {0};

    //mark j in hash array, as we will not use it. (we use i instead of j)

    int temp = 'j' - 'a';

    hashArray[temp] = 1;

    //fill the matrix with key

    //when filling a character, mark it in the hash array so that we don't insert it again afterwards

    int keyPointer = 0;

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            //to stop filling once whole key is inserted, we put condition

            while (keyPointer < key.length()){

                //check if key is repeated or not

                if (hashArray[int(key[keyPointer]) - int('a')] == 0){

                    matrix[i][j] = key[keyPointer];

                    hashArray[int(key[keyPointer]) - int('a')] = 1;

                    keyPointer++;

                    break;  //means break current iteration and do next one

                }

                else{

                    keyPointer++;

                }

            }

        }

    }

    //filling the rest of the matrix alphabet wise, with no character repeat

    string alphabet = "abcdefghijklmnopqrstuvwxyz";

    int alphabetPointer = 0;

    for (int i=0;i<5;i++){

        for (int j=0;j<5;j++){

            //to reach the unfilled values

            if (matrix[i][j] == '-'){

                while (alphabetPointer < alphabet.length()){

                    if (hashArray[int(alphabet[alphabetPointer]) - int('a')] == 0){

                        matrix[i][j] = alphabet[alphabetPointer];

                        hashArray[int(alphabet[alphabetPointer]) - int('a')] = 1;

                        alphabetPointer++;

                        break;

                    }

                    else{

                        alphabetPointer++;

                    }

                }

            }

        }

    }

    //length of encryptedText (cipher text) is always even

    //form the decrypted text by finding pairs in matrix and replacing them

    for (int i=0;i<encryptedText.length();i=i+2){

        char a = encryptedText[i];

        char b = encryptedText[i+1];

        //position of encryptedText characters

        int row\_a=0;

        int col\_a=0;

        int row\_b=0;

        int col\_b=0;

        //decrypted characters

        char ch1;

        char ch2;

        for (int j=0;j<5;j++){

            for (int k=0;k<5;k++){

                if (matrix[j][k] == a){

                    row\_a = j;

                    col\_a = k;

                }

                else if (matrix[j][k] == b){

                    row\_b = j;

                    col\_b = k;

                }

            }

        }

        //if both characters are in same row

        if (row\_a == row\_b){

            col\_a = (col\_a - 1 +5)%5;

            col\_b = (col\_b - 1 +5)%5;

        }

        //if both characters are in same column

        else if (col\_a == col\_b){

            row\_a = (row\_a - 1 +5)%5;

            row\_b = (row\_b - 1 +5)%5;

        }

        //if both are in different rows and columns

        else{

            //just interchange the col values

            int tempCol = col\_a;

            col\_a = col\_b;

            col\_b = tempCol;

        }

        ch1 = matrix[row\_a][col\_a];

        ch2 = matrix[row\_b][col\_b];

        decryptedText += ch1;

        decryptedText += ch2;

    }

    return decryptedText;

}

int main(){

    string plainText0 = "mynameisravi";

    string keyword0 = "pdeu";

    string plainText;

    string keyword;

    //truncate spaces from key and plainText, and convert to lowercase

    for (int i=0;i<plainText0.length();i++){

        if (plainText0[i] != ' '){

            plainText += tolower(plainText0[i]);

        }

    }

    for (int i=0;i<keyword0.length();i++){

        if (keyword0[i] != ' '){

            keyword += tolower(keyword0[i]);

        }

    }

    //replacing 'j' in keyword with 'i'

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

        if (keyword[i] == 'j'){

            keyword[i] = 'i';

        }

    }

    cout<<"Plain Text: "<<plainText;

    cout<<"\nKeyword: "<<keyword;

    string encryptedText = encrypt(plainText, keyword);

    cout<<"\nEncrypted Text: "<<encryptedText;

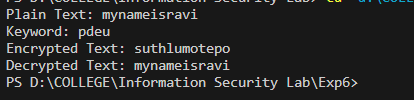
    string decryptedText = decrypt(encryptedText, keyword);

    cout<<"\nDecrypted Text: "<<decryptedText;

    return 0;

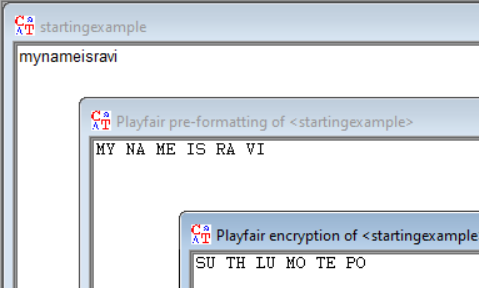
}

**Output (Program):**

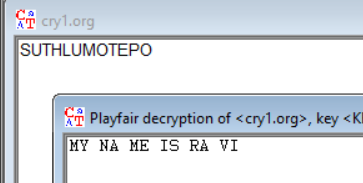
****

**Output (Cryptool):**

**Encryption:**

****

**Decryption:**



**Cryptanalysis:**

**Digraph Frequency Analysis:**

To perform digraph frequency analysis, begin by constructing a list of the most prevalent pairs of letters (digraphs) found in English text. Next, tally the occurrences of these digraphs in the encrypted message. If the distribution of digraph frequencies in the encrypted message differs from that in standard English text, this divergence can be utilized to deduce the encryption key. For instance, if the digraph "TH" appears more often in the encrypted message than it typically does in English text, it indicates that the key likely incorporates the "TH" digraph.

**Brute Force Attack:**

A brute force attack on the Playfair cipher involves systematically trying all possible 5x5 grids as decryption keys, using the 25 letters of the alphabet (minus duplicates) to form these grids. For each generated grid, the attacker decrypts the ciphertext and checks if the resulting text makes sense. This process continues until a valid decryption is found or all possibilities are exhausted. While it is time-consuming due to the large number of potential keys, it can eventually reveal the correct decryption key.

**Applications:**

• Military Communication

• Historical Encryption

• Classic Cryptography Education

**References:**

1. GeeksforGeeks
2. www.nku.edu
3. en.wikipedia.org