C&NS Lab Assignment 2

Onkar Santosh Gavali (2019BTECS00037)

Batch B2

Index

Play fair cipher

Matrix-based block cipher used in WWI

In a 5x5 matrix, write the letters of the word “Playfair” (for example) without dups, and fill in with other letters of the alphabet, except I, J used interchangeably.

Here we use a 5x5 matrix to map all characters and used it for encryption.

K1	K2	….	.....	....
….	Kn	C1	C2	….
….	….	….	….	….
….	….	….	….	Cm

Her k is the Key. and C is the character.

The operation to perform on plain text:

- Plain text divided into groups of 2.
- If both characters in one group are the same, then 2 different 2 groups formed with 1st character followed with ‘X’ or ‘Z’ and 2nd ground is formed using the normal method.
- Eg. AAB=> AZ AB
- If the last group only has 1 letter then the group is formed using the last character and the ‘X’.

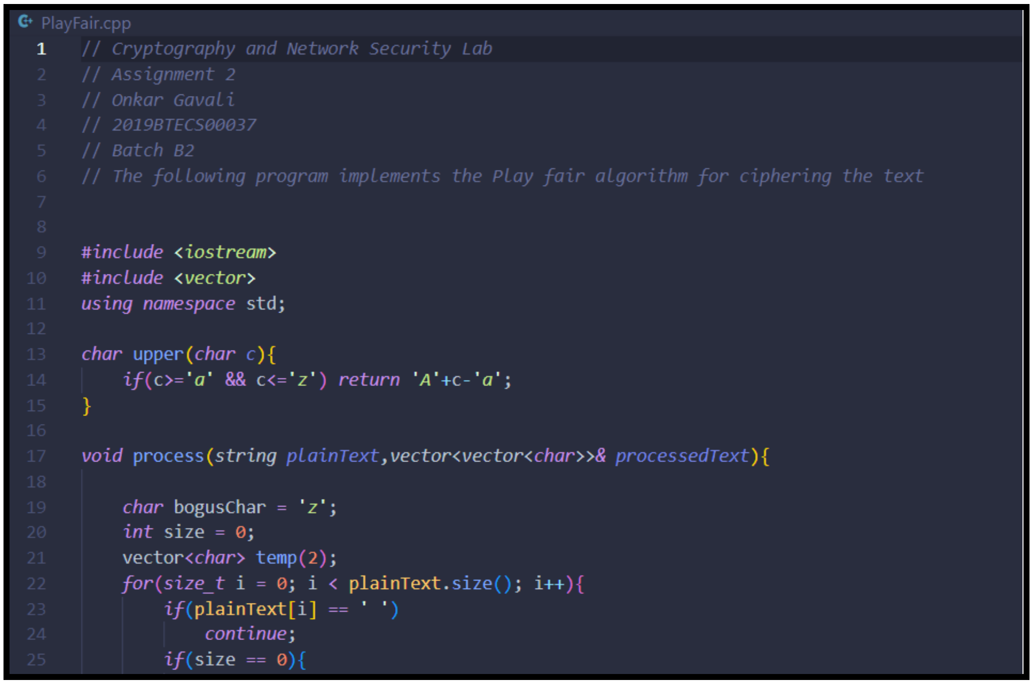
Key Formation:

- Place all nonrepeated characters of a key in the matrix (except for j use I instead).
- After that place all remaining alphabets in the matrix to complete the 5x5 matrix.(as shown in table)
- Place i and j in the same box.

Encryption:

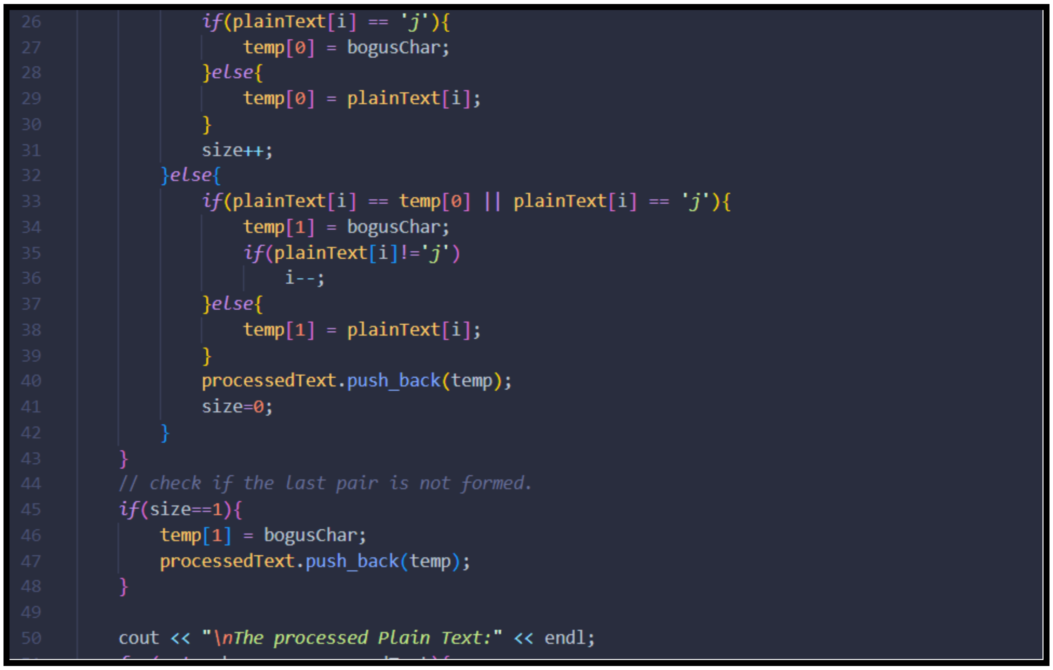
- Play fair is a polyalphabetic algorithm.
- It takes 2 characters at each time to encrypt plain text.
- If both characters of the group are in the same row then it is replaced by the immediate next character to it. Eg. if the group is “SF” and both are in the same row then and giver row is “ASGTF” the “SF” became “GA”
- If both characters in the group are in the same column then the immediate next character in the same column of the matrix is used to substitute the letters.
- If both are in different rows and columns the A(i,j)B(m,n) then A is replaced by letter at (i,n), and B is replaced by the letter at (m,j).

Code

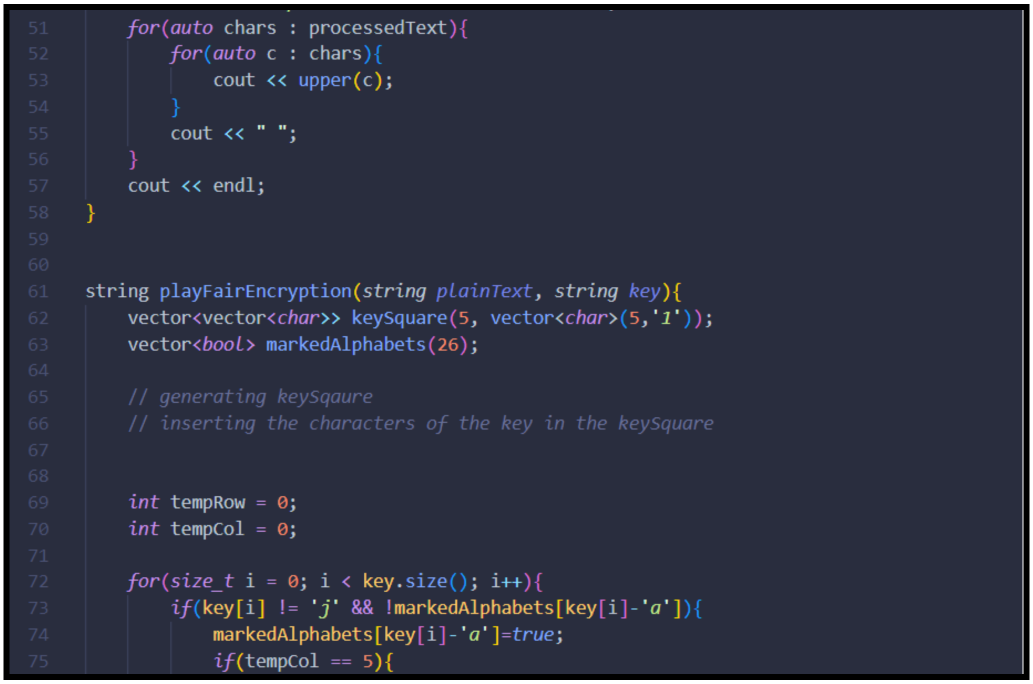


PlayFair.cpp

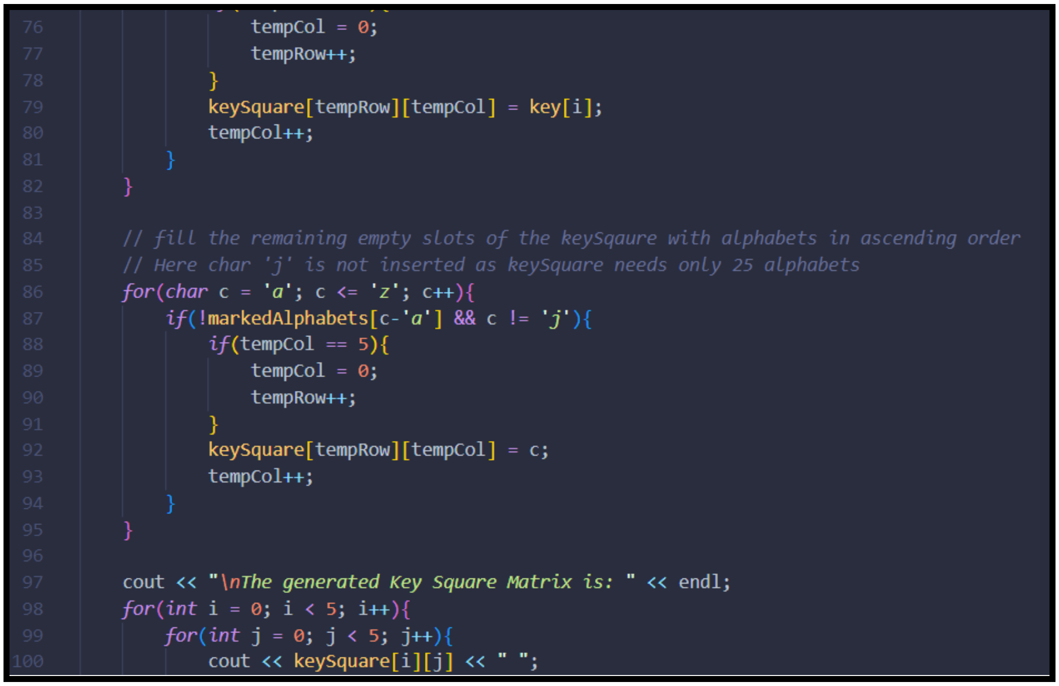
```cpp
// Cryptography and Network Security Lab
// Assignment 2
// Onkar Gavali
// 2019BTECS00037
// Batch B2
// The following program implements the Play fair algorithm for ciphering the text


#include <iostream>
#include <vector>
using namespace std;

char upper(char c){
    if(c>='a' && c<='z') return 'A'+c-'a';
}

void process(string plainText,vector<vector<char>>& processedText){

    char bogusChar = 'z';
    int size = 0;
    vector<char> temp(2);
    for(size_t i = 0; i < plainText.size(); i++){
        if(plainText[i] == ' ')
            continue;
        if(size == 0){
```



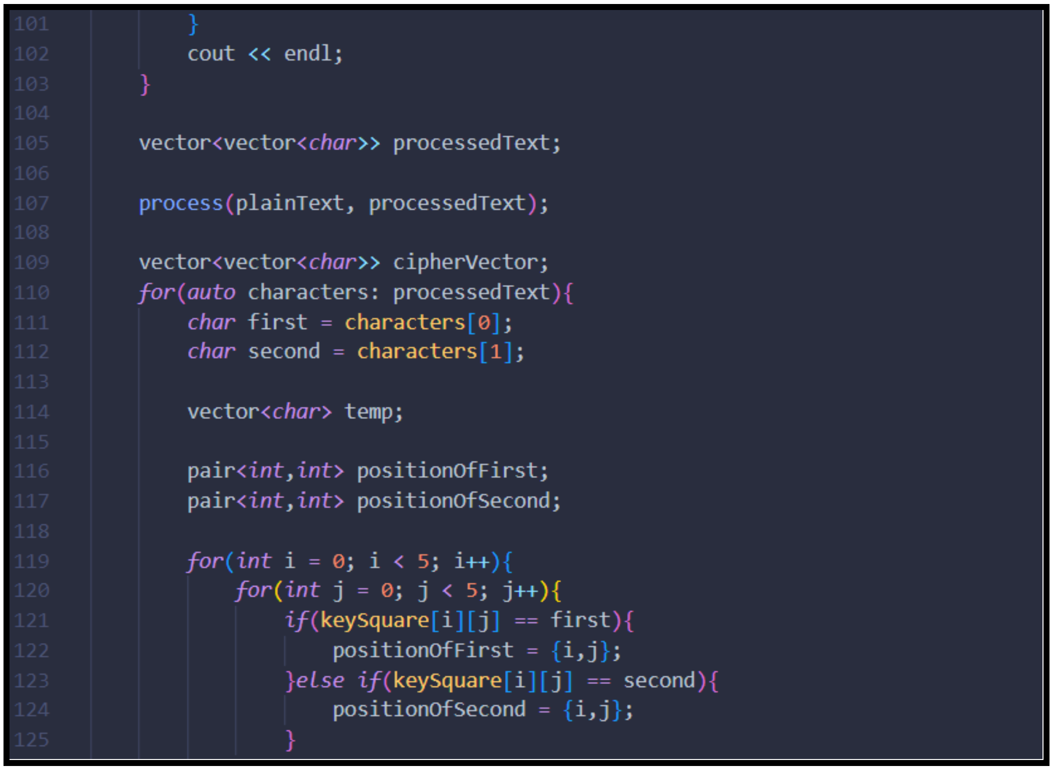
```cpp
            if(plainText[i] == 'j'){
                temp[0] = bogusChar;
            }else{
                temp[0] = plainText[i];
            }
            size++;
        }else{
            if(plainText[i] == temp[0] || plainText[i] == 'j'){
                temp[1] = bogusChar;
                if(plainText[i]!='j')
                    i--;
            }else{
                temp[1] = plainText[i];
            }
            processedText.push_back(temp);
            size=0;
        }
    }
    // check if the last pair is not formed.
    if(size==1){
        temp[1] = bogusChar;
        processedText.push_back(temp);
    }

    cout << "\nThe processed Plain Text:" << endl;
```



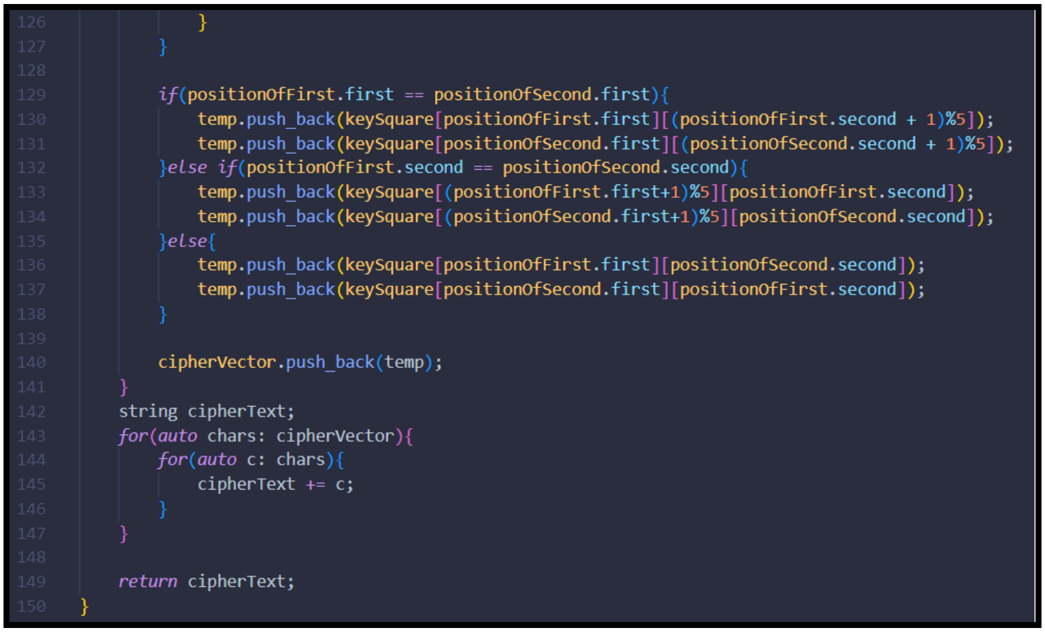
```cpp
    for(auto chars : processedText){
        for(auto c : chars){
            cout << upper(c);
        }
        cout << " ";
    }
    cout << endl;
}


string playFairEncryption(string plainText, string key){
    vector<vector<char>> keySquare(5, vector<char>(5,'1'));
    vector<bool> markedAlphabets(26);

    // generating keySqaure
    // inserting the characters of the key in the keySquare


    int tempRow = 0;
    int tempCol = 0;

    for(size_t i = 0; i < key.size(); i++){
        if(key[i] != 'j' && !markedAlphabets[key[i]-'a']){
            markedAlphabets[key[i]-'a']=true;
            if(tempCol == 5){
```



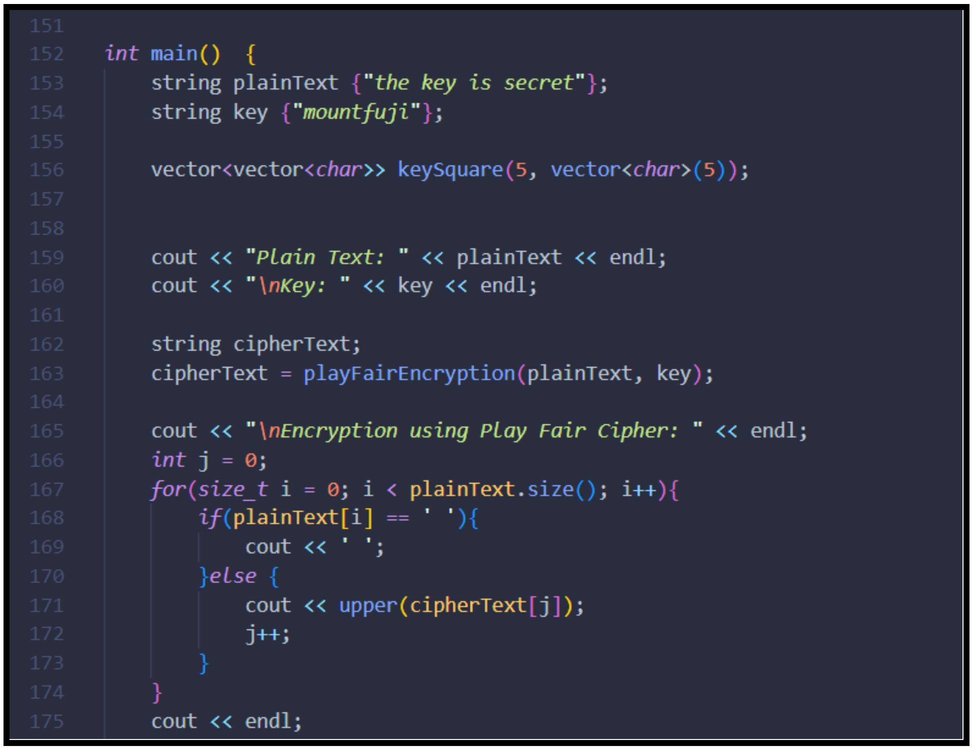
```cpp
                tempCol = 0;
                tempRow++;
            }
            keySquare[tempRow][tempCol] = key[i];
            tempCol++;
        }
    }

    // fill the remaining empty slots of the keySqaure with alphabets in ascending order
    // Here char 'j' is not inserted as keySquare needs only 25 alphabets
    for(char c = 'a'; c <= 'z'; c++){
        if(!markedAlphabets[c-'a'] && c != 'j'){
            if(tempCol == 5){
                tempCol = 0;
                tempRow++;
            }
            keySquare[tempRow][tempCol] = c;
            tempCol++;
        }
    }

    cout << "\nThe generated Key Square Matrix is: " << endl;
    for(int i = 0; i < 5; i++){
        for(int j = 0; j < 5; j++){
            cout << keySquare[i][j] << " ";
```



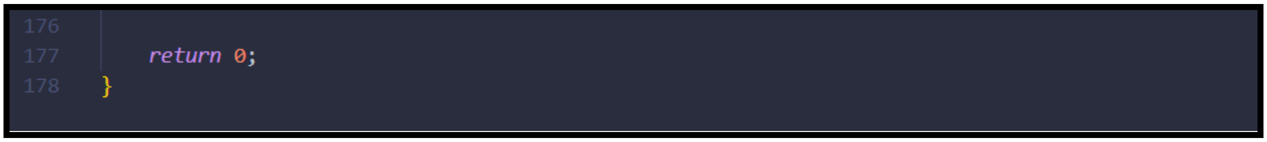
```
        }
        cout << endl;
    }

    vector<vector<char>> processedText;

    process(plainText, processedText);

    vector<vector<char>> cipherVector;
    for(auto characters: processedText){
        char first = characters[0];
        char second = characters[1];

        vector<char> temp;

        pair<int,int> positionOfFirst;
        pair<int,int> positionOfSecond;

        for(int i = 0; i < 5; i++){
            for(int j = 0; j < 5; j++){
                if(keySquare[i][j] == first){
                    positionOfFirst = {i,j};
                }else if(keySquare[i][j] == second){
                    positionOfSecond = {i,j};
                }
```



```
            }
        }

        if(positionOfFirst.first == positionOfSecond.first){
            temp.push_back(keySquare[positionOfFirst.first][(positionOfFirst.second + 1)%5]);
            temp.push_back(keySquare[positionOfSecond.first][(positionOfSecond.second + 1)%5]);
        }else if(positionOfFirst.second == positionOfSecond.second){
            temp.push_back(keySquare[(positionOfFirst.first+1)%5][positionOfFirst.second]);
            temp.push_back(keySquare[(positionOfSecond.first+1)%5][positionOfSecond.second]);
        }else{
            temp.push_back(keySquare[positionOfFirst.first][positionOfSecond.second]);
            temp.push_back(keySquare[positionOfSecond.first][positionOfFirst.second]);
        }

        cipherVector.push_back(temp);
    }
    string cipherText;
    for(auto chars: cipherVector){
        for(auto c: chars){
            cipherText += c;
        }
    }

    return cipherText;
}
```

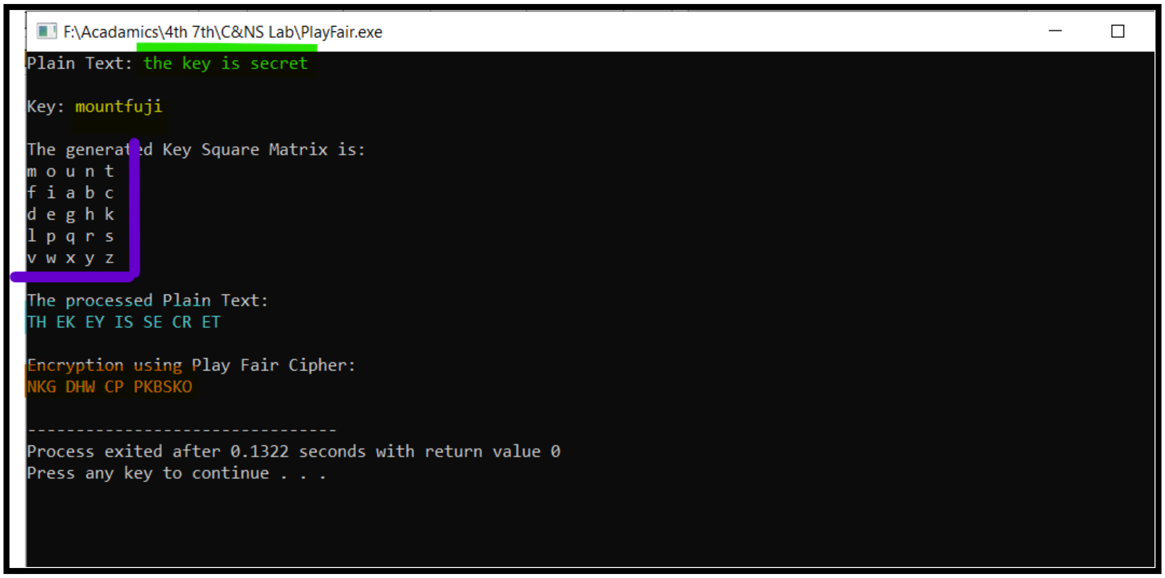


```cpp

int main()  {
    string plainText {"the key is secret"};
    string key {"mountfuji"};

    vector<vector<char>> keySquare(5, vector<char>(5));


    cout << "Plain Text: " << plainText << endl;
    cout << "\nKey: " << key << endl;

    string cipherText;
    cipherText = playFairEncryption(plainText, key);

    cout << "\nEncryption using Play Fair Cipher: " << endl;
    int j = 0;
    for(size_t i = 0; i < plainText.size(); i++){
        if(plainText[i] == ' '){
            cout << ' ';
        }else {
            cout << upper(cipherText[j]);
            j++;
        }
    }
    cout << endl;
```



```cpp

    return 0;
}
```

Output



```
Plain Text: the key is secret

Key: mountfuji

The generated Key Square Matrix is:
m o u n t
f i a b c
d e g h k
l p q r s
v w x y z

The processed Plain Text:
TH EK EY IS SE CR ET

Encryption using Play Fair Cipher:
NKG DHW CP PKBSKO

--------------------------------
Process exited after 0.1322 seconds with return value 0
Press any key to continue . . .
```

Conclusion

- PlayFair better solution to the caesar cipher
- More complex
- One of the good well know examples of Polyalphabetic encryption algorithms.
- But easily gets cracked by today’s computer World