**Name : Akash Babu Misal**

**PRN: 21520005**

**Class:** Final Year (Computer Science and Engineering)

**Course Name: Cryptography and Network Security**  **Lab**

**Assignment No – 3**

Aim - Implementation of Playfair Cypher

Theory –

**Playfair Cipher Overview:**

The Playfair Cipher is a polyalphabetic substitution cipher that provides a more secure way of encrypting text compared to simpler substitution ciphers like the Caesar Cipher. It operates on pairs of letters (bigrams) from the plaintext and uses a 5x5 key matrix (also known as the Playfair Square) to perform the encryption and decryption.

**Part-A: Encryption**

In this part of the experiment, we will implement the Playfair Cipher encryption process. The encryption process involves the following steps:

**Key Setup:** The user provides a keyword, which is used to generate the Playfair Square. The keyword is used to create the 5x5 matrix, excluding duplicate letters and filling in the remaining cells with the remaining letters of the alphabet.

**Input:** The user provides a plaintext message to be encrypted.

**Text Preparation:** The plaintext message is processed to remove spaces, convert to uppercase, and break it into bigrams (pairs of two letters). If there is an odd number of characters, a dummy character (usually 'X') is added at the end.

**Encryption:** For each bigram in the plaintext, we perform the following steps:

Locate the positions of the two letters in the Playfair Square.

Apply the Playfair Cipher rules to determine the encrypted bigram.

Replace the original bigram with the encrypted bigram.

**Output**: The encrypted message is produced, consisting of the transformed bigrams.

**Part-B: Decryption**

In this part of the experiment, we will implement the Playfair Cipher decryption process. The decryption process is the reverse of encryption and involves the following steps:

**Input:** The user provides the ciphertext and the keyword used for encryption.

**Key Setup:** Using the provided keyword, recreate the same Playfair Square used for encryption.

**Decryption:** For each bigram in the ciphertext, we perform the following steps:

Locate the positions of the two letters in the Playfair Square.

Apply the Playfair Cipher rules to determine the decrypted bigram.

Replace the original bigram in the ciphertext with the decrypted bigram.

**Output:** The decrypted message is produced, which should match the original plaintext message.

**Code**:

#include <bits/stdc++.h>

using namespace std;

class PlayfairCipher

{

public:

    static pair<vector<vector<char>>, unordered\_map<char, pair<int, int>>>

    getKeyMatrixAndPositions(const string &key)

    {

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

        int i = 0, j = 0;

        unordered\_set<char> set;

        unordered\_map<char, pair<int, int>> position;

        for (char c : key)

        {

            if (c == 'j')

                c = 'i';

            if (set.find(c) != set.end())

                continue;

            set.insert(c);

            keyMatrix[i][j] = c;

            position[c] = {i, j};

            j++;

            if (j == 5)

            {

                j = 0;

                i++;

            }

        }

        for (char c = 'a'; c <= 'z'; c++)

        {

            if (c == 'j')

                continue;

            if (set.find(c) != set.end())

                continue;

            set.insert(c);

            keyMatrix[i][j] = c;

            position[c] = {i, j};

            j++;

            if (j == 5)

            {

                j = 0;

                i++;

            }

        }

        position[j] = position[i];

        return {keyMatrix, position};

    }

    static vector<string> getDiagrams(const string &text)

    {

        int n = text.size();

        int i = 0;

        vector<string> diagrams;

        while (i + 1 < n)

        {

            if (text[i] != text[i + 1])

            {

                string d;

                d += tolower(text[i]);

                d += tolower(text[i + 1]);

                diagrams.push\_back(d);

                i += 2;

            }

            else

            {

                string d;

                d += tolower(text[i]);

                d += 'x';

                diagrams.push\_back(d);

                i++;

            }

        }

        if (i == n - 1)

        {

            string d;

            d += tolower(text[i]);

            d += 'x';

            diagrams.push\_back(d);

        }

        return diagrams;

    }

    static string encrypt(const string &plaintext, const string &key)

    {

        auto p = getKeyMatrixAndPositions(key);

        auto keyMatrix = p.first;

        auto position = p.second;

        vector<string> diagrams = getDiagrams(plaintext);

        stringstream ciphertext;

        for (string &diagram : diagrams)

        {

            auto p1 = position[diagram[0]];

            auto p2 = position[diagram[1]];

            auto i0 = p1.first, j0 = p1.second;

            auto i1 = p2.first, j1 = p2.second;

            if (i0 == i1)

            {

                diagram[0] = keyMatrix[i0][(j0 + 1) % 5];

                diagram[1] = keyMatrix[i0][(j1 + 1) % 5];

            }

            else if (j0 == j1)

            {

                diagram[0] = keyMatrix[(i0 + 1) % 5][j0];

                diagram[1] = keyMatrix[(i1 + 1) % 5][j0];

            }

            else

            {

                diagram[0] = keyMatrix[i0][j1];

                diagram[1] = keyMatrix[i1][j0];

            }

            ciphertext << diagram;

        }

        string answer = ciphertext.str();

        transform(answer.begin(), answer.end(), answer.begin(), ::toupper);

        return answer;

    }

    static string decrypt(const string &ciphertext, const string &key)

    {

        auto p = getKeyMatrixAndPositions(key);

        auto keyMatrix = p.first;

        auto position = p.second;

        vector<string> diagrams = getDiagrams(ciphertext);

        stringstream plaintext;

        for (string &diagram : diagrams)

        {

            auto p1 = position[diagram[0]];

            auto p2 = position[diagram[1]];

            auto i0 = p1.first, j0 = p1.second;

            auto i1 = p2.first, j1 = p2.second;

            if (i0 == i1)

            {

                diagram[0] = keyMatrix[i0][(j0 - 1 + 5) % 5];

                diagram[1] = keyMatrix[i0][(j1 - 1 + 5) % 5];

            }

            else if (j0 == j1)

            {

                diagram[0] = keyMatrix[(i0 - 1 + 5) % 5][j0];

                diagram[1] = keyMatrix[(i1 - 1 + 5) % 5][j0];

            }

            else

            {

                diagram[0] = keyMatrix[i0][j1];

                diagram[1] = keyMatrix[i1][j0];

            }

            plaintext << diagram;

        }

        return plaintext.str();

    }

};

int main()

{

    cout << "PlayFair Cipher:\n"

         << "Enter your choice:\n"

         << "1. Encrypt\n"

         << "2. Decrypt\n";

    int choice;

    cin >> choice;

    switch (choice)

    {

    case 1:

    {

        cout << "Enter plaintext: ";

        string plaintext;

        cin.get();

        getline(cin, plaintext);

        plaintext.erase(remove\_if(plaintext.begin(), plaintext.end(), ::isspace),

                        plaintext.end());

        cout << "Enter key : ";

        string key;

        cin >> key;

        string ciphertext = PlayfairCipher::encrypt(plaintext, key);

        cout << "Plaintext:  " << plaintext << "\n"

             << "Ciphertext: " << ciphertext << "\n";

    }

    break;

    case 2:

    {

        cout << "Enter ciphertext: ";

        string ciphertext;

        cin >> ciphertext;

        cout << "Enter key : ";

        string key;

        cin >> key;

        string plaintext = PlayfairCipher::decrypt(ciphertext, key);

        cout << "Ciphertext: " << ciphertext << "\n"

             << "Plaintext:  " << plaintext << "\n";

    }

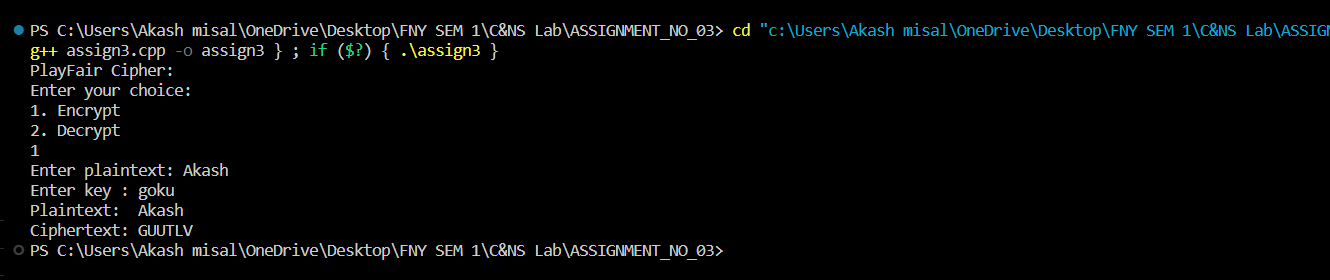
    break;

    }

    return 0;

}

**Output –**

****

**A screen shot of a computer

Description automatically generated**

**Conclusion**:

The Playfair Cipher is a more complex encryption technique that offers improved security compared to simpler ciphers. This experiment allows us to understand both the encryption and decryption processes of the Playfair Cipher and highlights the importance of the keyword in generating the key matrix. Proper implementation and key management are crucial for maintaining the confidentiality of the message.