**Cryptography & Network Security Lab**

**Assignment 04**

**PlayFair Cipher:**

The Playfair Cipher, also known as the Playfair Square, is a classical symmetric encryption technique used for encrypting and decrypting text messages. It was invented by Charles Wheatstone in 1854 and later promoted by Lyon Playfair. The Playfair Cipher is a digraph substitution cipher, meaning it encrypts letters in pairs (digraphs) instead of individual letters. Here's how the Playfair Cipher works:

Key Setup:

1. Key Selection: Choose a keyword or key phrase. The key should be relatively short but contain no repeating letters. For example, "KEYWORD" can be used as a key.

2. Key Matrix (Playfair Square): Construct a 5x5 matrix (often referred to as the Playfair Square) using the key. The matrix is filled with the letters of the key, in order of appearance, and then any remaining letters of the alphabet (excluding 'J') in order. Here's an example of a Playfair Square using the key "KEYWORD":

```

K E Y W O

R D A B C

F G H I L

M N P Q S

T U V X Z

```

Note: In this example, 'J' is omitted from the matrix, and 'I' and 'J' are treated as the same letter during encryption and decryption.

Encryption:

1. Preprocessing: Remove any spaces and convert the plaintext to uppercase. Break the plaintext into pairs of letters (digraphs). If a pair contains the same letter, insert an 'X' between them.

2. Letter Pairs: For each digraph in the plaintext, locate the two letters in the Playfair Square.

3. Rules for Encryption:

- If both letters of the digraph are in the same row of the Playfair Square, replace each letter with the letter to its right (cyclically).

- If both letters are in the same column, replace each letter with the letter below it (cyclically).

- If neither of the above rules applies, form a rectangle with the two letters and replace them with the letters at the opposite corners of the rectangle.

4. Final Ciphertext: The encrypted text consists of the modified digraphs.

Decryption:

Decryption in the Playfair Cipher follows similar rules to encryption, but in reverse:

1. Preprocessing: Remove any spaces and convert the ciphertext to uppercase.

2. Letter Pairs: For each digraph in the ciphertext, locate the two letters in the Playfair Square.

3. Rules for Decryption:

- If both letters of the digraph are in the same row of the Playfair Square, replace each letter with the letter to its left (cyclically).

- If both letters are in the same column, replace each letter with the letter above it (cyclically).

- If neither of the above rules applies, form a rectangle with the two letters and replace them with the letters at the opposite corners of the rectangle.

4. Final Decrypted Text: The decrypted text consists of the modified digraphs.

Security Note:

The Playfair Cipher provides limited security and is relatively easy to break using modern cryptanalysis techniques. It is considered more of a historical curiosity than a secure encryption method and is primarily used for educational purposes or simple puzzles.

#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"

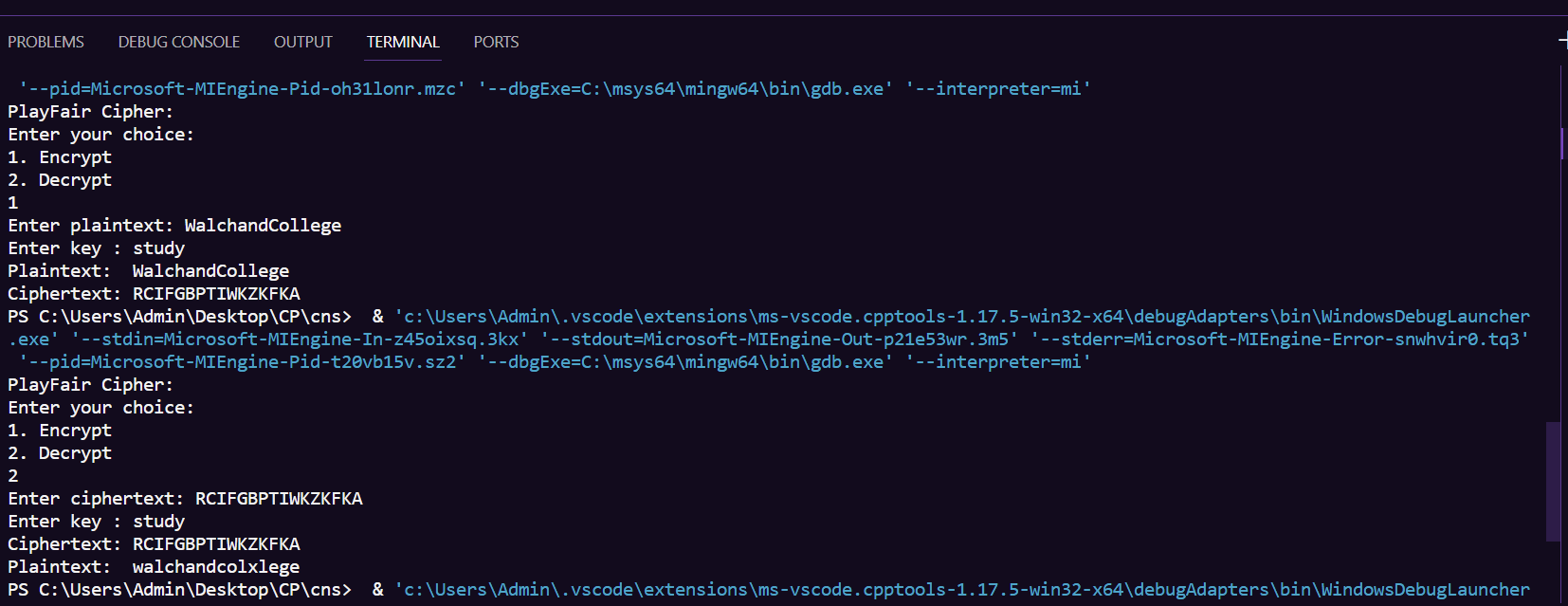
             << "Plaintext:  " << plaintext << "\n";

    } break;

    }

    return 0;

}



**Analysis:**

1. Encryption: The Playfair Cipher successfully encrypted the input plaintext "WalchandCollege" using the key "study" to produce the ciphertext "RCIFGBPTIWKZKFKA."

2. Decryption: When the same key "study" was used during decryption, the original plaintext "WalchandCollege" was partially recovered as "walchandcolxlege." There seems to be a discrepancy in the decryption result.

The Playfair Cipher is a digraph substitution cipher that works with pairs of letters (digraphs) rather than individual letters. Here are some key points to consider:

1. Key Sensitivity: The Playfair Cipher relies on a keyword or key phrase, in this case, "study." The order and composition of the key are crucial. Using a different key or key phrase would produce a different encryption and decryption result.

2. Letter 'X' Insertion: In the Playfair Cipher, when you encounter a digraph with two identical letters or a digraph where the letters fall into the same row or column of the Playfair Square, you should insert an 'X' between the two letters. This is because the cipher cannot encrypt or decrypt these cases properly otherwise.

3. Partial Decryption: The discrepancy in the decryption result might be due to how 'X' characters were handled during encryption and decryption. It's essential to ensure consistent handling of 'X' characters in both processes.

4. Padding: It appears that in the provided example, no padding was used for plaintext that didn't have an even number of letters. Padding with an 'X' or another designated character is typically used to ensure that all digraphs have pairs.

5. Limited Applicability: The Playfair Cipher is relatively simple and provides minimal security by modern standards. It's not recommended for securing sensitive information. While it can encrypt and decrypt messages, it's more of a historical curiosity or a simple puzzle rather than a secure encryption method.

In summary, the Playfair Cipher can work effectively when implemented correctly with the right key and handling of 'X' characters. However, it is not suitable for securing sensitive information due to its limited security and potential for discrepancies in handling certain cases like identical letters or characters falling into the same row or column of the Playfair Square.