Name : Ritesh Pawar
PRN : 2020BTECS00068

Cryptography and Network Security Lab

Name:Ritesh Pawar

PRN: 2020BTECS00068

Batch: B5

PLAY FAIR CIPHER ALGORITHM

Aim:

To encrypt plain text using PlayFair cipher and decrypt the cipher text to plain text.

Theory:

Playfair cipher is a manual symmetric encryption technique and was first diagram substitution cipher. In playfair cipher group of letters is encrypted instead of a single letter so it is little bit complex than caesar cipher. So it is hard to break playfair cipher algorithm as in simple caesar cipher one can easily predict k value and decrypt the text easily. So this playfair cipher algorithm is more secure than caesar cipher.

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)
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
{
```

```
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};
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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};
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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) {
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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) {
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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) {
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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;

}
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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) {
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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];
}
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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: {
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
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;
```

Name : Ritesh Pawar
PRN : 2020BTECS00068

```
}
```

Output:

Limitations:

Key Length and Reusability: The Playfair Cipher requires a relatively long and complex key, which can be challenging to manage and remember securely. Reusing the same key for multiple messages can compromise security.

Frequency Analysis: Although the Playfair Cipher can disguise single-letter frequencies, it does not effectively disguise digraph (two-letter) frequencies. Skilled cryptanalysts can use frequency analysis to break the cipher, especially when the message is long.

Known-Plaintext Attacks: If an attacker has access to both the ciphertext and corresponding plaintext for a portion of the message, they can potentially deduce parts of the key or plaintext, compromising the entire message's security.

Determined Attackers: With enough ciphertext, skilled attackers can employ various techniques like hill climbing, genetic algorithms, and simulated annealing to break the Playfair Cipher through brute force or optimization.

Pattern Recognition: The Playfair Cipher retains certain patterns in the plaintext, such as repeated letters in the same pair, which can be exploited by cryptanalysts to guess parts of the key or plaintext.

Name : Ritesh Pawar
PRN : 2020BTECS00068

Complexity for Manual Encryption: When used manually, the Playfair Cipher can be error-prone and complex, leading to mistakes in key setup or encryption/decryption.

Block Size Limitations: The Playfair Cipher encrypts plaintext in pairs of letters, which may result in uneven block sizes for messages with an odd number of characters. Padding may be needed, introducing complexity.

Not Suitable for Modern Secure Communication: Due to its weaknesses and vulnerability to modern cryptographic attacks, the Playfair Cipher is not suitable for securing sensitive or confidential information in today's digital age. Modern encryption algorithms like AES (Advanced Encryption Standard) are more secure and recommended for secure communication.