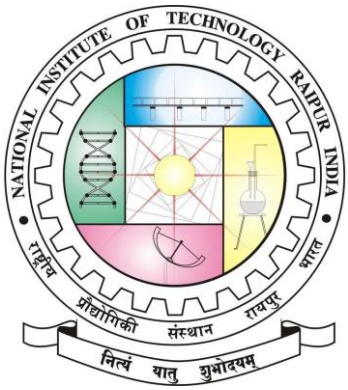


National Institute of Technology Raipur

Cryptography and Network Security
Term Project

Implement Playfair Cipher and Demonstrate the attack on it

By

Kunal Sachdeva

Roll No. 19115045

(6th Semester Computer Science & Engineering 2020-21)

Abstract

The Playfair Cipher is a message encryption or encoding technique. It's the same as a standard cypher. The main difference is that instead of a single letter, it encrypts a digraph (a pair of two characters). The implementation of Playfair Cipher as well as the cryptoanalysis of Playfair Cipher are covered in this work.

Introduction

Charles Whetstone proposed the Playfair cypher in 1889. However, because one of his companions, Lord Lyon Playfair, advocated its use, it was named after him. It is the most widely used symmetric encryption technique, and it belongs to the substitution cypher family. It's a method of encoding that encrypts many letters at once.

The Playfair cypher was the first digraph substitution cypher to be used in practice. Unlike standard cyphers, Playfair cypher encrypts a pair of alphabets (digraphs) rather than a single alphabet. It was used by British forces in the Second Boer War and World War I for tactical purposes, and by Australian forces in World War II for the same purpose. This is due to the fact that Playfair is simple to use and does not require any extra equipment.

It starts by generating a 5x5 matrix key-table. The matrix comprises alphabets that serve as the plaintext's encryption key. It's important to remember that no alphabet should be duplicated. Another thing to keep in mind is that there are 26 alphabets and only 25 blocks to fit a letter into. As a result of the surplus letter, a letter (typically J) will be omitted from the matrix. Despite this, J appears in the plaintext, which is then substituted by I. It implies to regard I and J as if they were the same letter.

The message is encrypted digraph by digraph with the Playfair cypher. As a result, the Playfair cypher might be considered a digraph substitution cypher.

Playfair Cipher Encryption Rules

1. Split the plaintext into digraphs first (pair of two letters). Append the letter Z to the end of the plaintext if it has an odd number of letters. It creates an even plaintext. The plaintext MANGO, for example, has five letters. As a result, there is no way to build a digraph. We'll append a letter Z to the end of the plaintext, resulting in MANGOZ.

2. Next, divide the plaintext into digraphs (pair of two letters). If a letter appears twice (side by side), mark the second occurrence with an X. The plaintext becomes CO MX MU NI CA TE if the digraph is CO MX MU

NI CA TE. Similarly, the plaintext JAZZ will have a digraph of JA ZX ZX, while plaintext GREET will have a digraph of GR EX ET.

3. To figure out the cypher (encryption) text, start by making a 5x5 key-matrix or key-table and filling it with alphabet letters, as shown below:
   - Fill the first row with the letters of the provided keyword (from left to right) (MONARCHY). If there are any duplicate letters in the keyword, avoid them. This indicates that a letter will only be considered once. Fill in the remaining letters in alphabetical sequence after that. For the keyword MONARCHY, let's make a 5x5 key-matrix.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

4. One or more of the following three circumstances may exist:

   a. If a digraph (a pair of letters) appears in the same row.
   Replace each letter of the digraph with the letters directly to its right in this case. Consider the first letter of the same row as the right letter if there is no letter to the right.

   b. If a digraph (a pair of letters) appears in the same column.
   Replace each letter of the digraph with the letters immediately below it in this case. Wrap around to the top of the same column if there is no letter below.

   c. If a digraph (a pair of letters) appears in a separate row and column
   In this scenario, choose a 3x3 matrix from a 5x5 matrix so that the 3x3 matrix contains a pair of letters. Because they are in the matrix, they occupy two opposite corners of a square. The other corner will be a cypher for the specified digraph, i.e., take the letters from the rectangle's horizontal opposite corner.

Implementation of Playfair Cypher Encryption

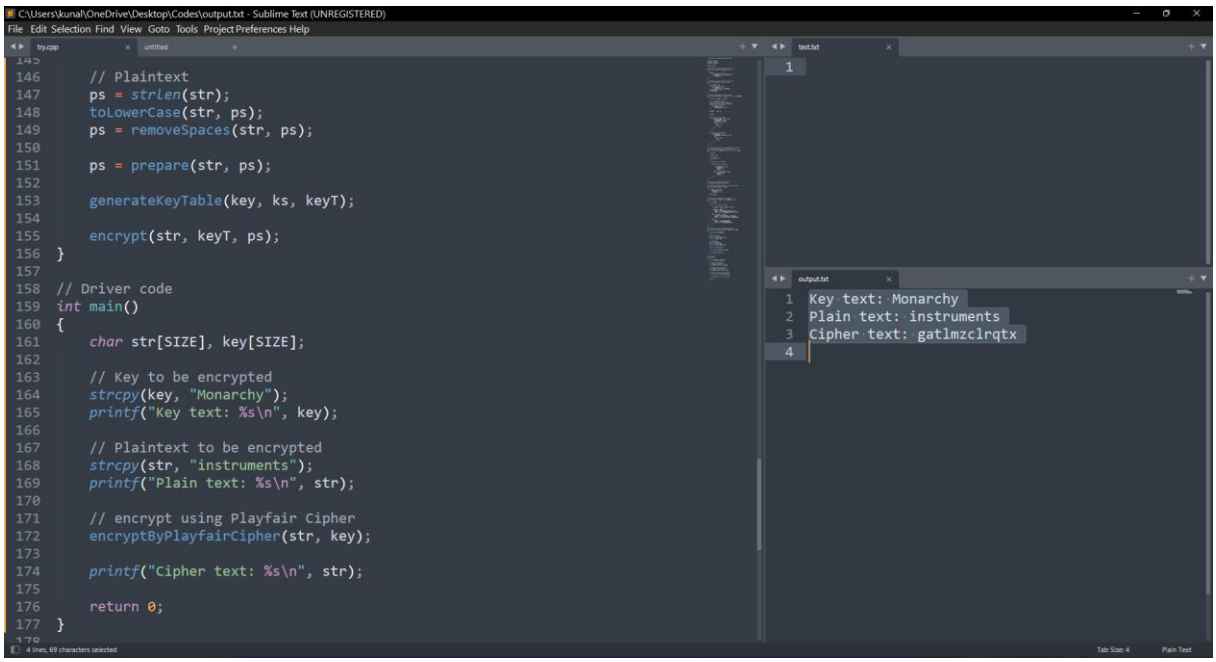
```
// C program to implement Playfair Cipher Encryption

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 30

// Function to convert the string to lowercase
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Function to remove all spaces in a string
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// Function to generate the 5x5 key square
void generateKeyTable(char key[], int ks, char
keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap
    // to store count of the alphabet
    dicty = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
```

```
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

// Function to search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int
arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
```

```
            b = 'i';

        for (i = 0; i < 5; i++) {

            for (j = 0; j < 5; j++) {

                if (keyT[i][j] == a) {
                    arr[0] = i;
                    arr[1] = j;
                }
                else if (keyT[i][j] == b) {
                    arr[2] = i;
                    arr[3] = j;
                }
            }
        }
}

// Function to find the modulus with 5
int mod5(int a) { return (a % 5); }

// Function to make the plain text length to be even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// Function for performing the encryption
void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
```

```
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}
// Function to encrypt using Playfair Cipher
void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // Plaintext
    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}
// Driver code
int main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    printf("Key text: %s\n", key);
```

```
    // Plaintext to be encrypted
    strcpy(str, "instruments");
    printf("Plain text: %s\n", str);

    // encrypt using Playfair Cipher
    encryptByPlayfairCipher(str, key);

    printf("Cipher text: %s\n", str);

    return 0;
}
```



Playfair Cipher Decryption Rules

The decryption technique follows the same procedures as encryption, but in reverse order. The cypher is symmetric for decryption (move left along rows and up along columns). The plain text recipient has the same key as the sender and can use the same key-table to decrypt the message.

The Playfair Cipher Decryption Algorithm:

There are two steps in the algorithm:

1. At the receiver's end, generate the key Square(5x5):
    - The key square is a 5x5-character grid of alphabets that serves as the encryption key. One letter of the alphabet (typically J) is missing from the table because each of the 25 alphabets must be unique (as the table can hold only 25 alphabets). If J appears in the plaintext, it is substituted by I.

- The beginning alphabets in the key square are the key's unique alphabets in order of appearance, followed by the remaining letters of the alphabet in order of appearance.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

2. Algorithm to decrypt the ciphertext: The ciphertext is split into pairs of two letters (digraphs).
   Rules for Decryption:
   - If both the letters are in the same column: Take the letter above each one (going back to the bottom if at the top).
   - If both the letters are in the same row: Take the letter to the left of each one (going back to the rightmost if at the leftmost position).
   - If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Implementation of Playfair Cypher Decryption

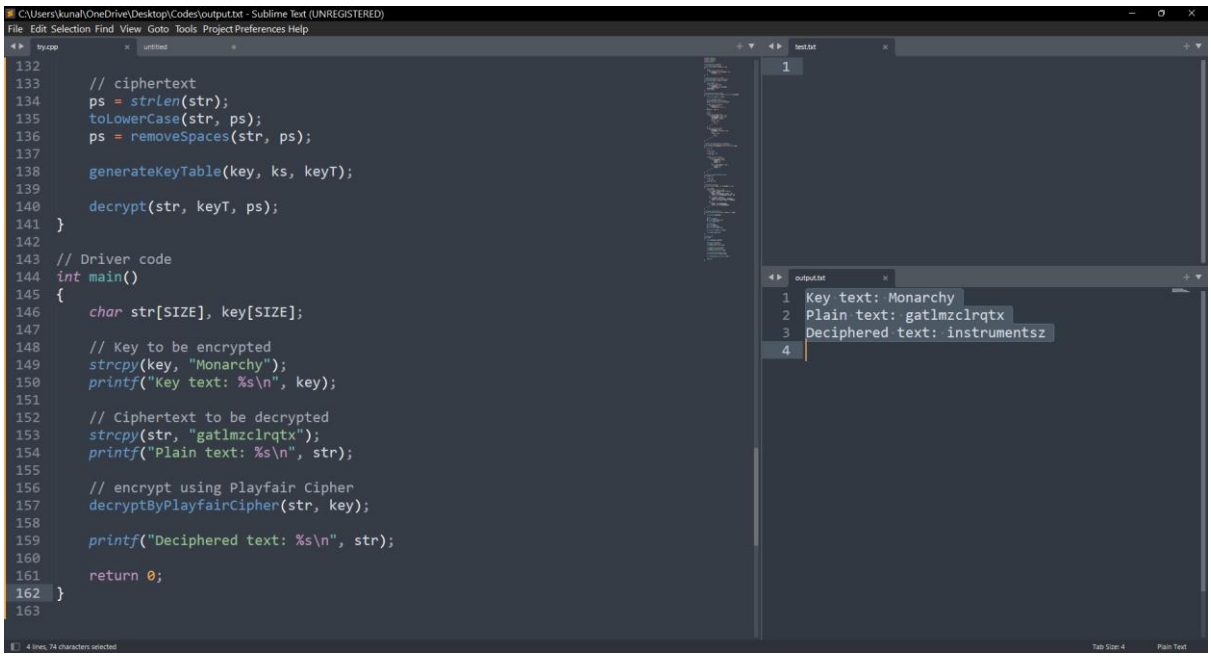
```
// C program to implement Playfair Cipher Decryption

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 30

// Convert all the characters
// of a string to lowercase
void toLowerCase(char plain[], int ps)
{
    int i;
```

```
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Remove all spaces in a string
// can be extended to remove punctuation
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// generates the 5x5 key square
void generateKeyTable(char key[], int ks, char
keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap
    // to store count of the alphabet
    dicty = (int*)calloc(26, sizeof(int));

    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }
    dicty['j' - 97] = 1;

    i = 0;
    j = 0;
    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
```

```
                i++;
                j = 0;
            }
        }
    }
    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

// Search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int
arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}
```

```
// Function to find the modulus with 5
int mod5(int a)
{
    if (a < 0)
        a += 5;
    return (a % 5);
}
// Function to decrypt
void decrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];
    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] - 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] - 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// Function to call decrypt
void decryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // ciphertext
    ps = strlen(str);
    toLowerCase(str, ps);
```

```
    ps = removeSpaces(str, ps);

    generateKeyTable(key, ks, keyT);

    decrypt(str, keyT, ps);
}
// Driver code
int main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    printf("Key text: %s\n", key);

    // Ciphertext to be decrypted
    strcpy(str, "gatlmzclrqtx");
    printf("Plain text: %s\n", str);

    // encrypt using Playfair Cipher
    decryptByPlayfairCipher(str, key);

    printf("Deciphered text: %s\n", str);

    return 0;
}
```



Advantages and Limitations of Playfair Cipher

It is much more difficult to break because the frequency analysis approach used to break basic substitution cyphers is tough, but it may still be applied on (25*25) = 625 digraphs rather than the difficult 25 monographs. To crack the encryption, frequency analysis requires more cypher text.

Advantages of Playfair Cipher

1. Various ciphertexts If we look at the Algorithm closely, we can see that we get different ciphertext at each stage, which causes greater difficulty for the cryptanalyst.
2. It is unaffected by brute force attacks.
3. It is not possible to cryptanalyze (decode a cypher without knowing the key).
4. The constraint of the simple Playfair square cypher is overcome.
5. The swap is simple to carry out.

Limitations of Playfair Cipher

The fact that a digraph in the ciphertext (AB) and its reverse (BA) would have equivalent plaintexts like UR and RU is an interesting flaw (and also ciphertext UR and RU will correspond to plaintext AB and BA, i.e., the substitution is self-inverse). If the plaintext's language is known, this can be easily exploited via frequency analysis. Another drawback is that the Playfair cypher is a symmetric cypher, which means that the same key is used for encryption and decryption.

The Playfair cypher has the following limitations:

1. There are only 25 alphabets that can be used.
2. Numeric characters are not supported.
3. Only upper- or lower-case letters are supported.
4. Special characters (such as blank space, newlines, punctuations, and so on) are not permitted.
5. Other than English, it does not support any other languages.
6. It's also not possible to encrypt media files.

Cryptanalysis of the Playfair Cipher

Long message

The Playfair cypher, like Substitution cyphers, can be broken via a frequency attack. However, because the Playfair cypher uses digrams rather than single letters, such an attack is far more difficult (as used in most substitution ciphers). Attacking a Playfair cypher necessitates identifying over 600 mappings between

pairs of digrams, rather than determining a mapping of twenty-six plaintext letters to ciphertext letters. Because each digram must occur a significant number of times in the ciphertext for a credible estimate of its relative probability of usage to be determined (for example, if each digram appears only a few times in the ciphertext, the least probable diagram may appear the most merely by coincidence).

Short message

It is necessary to rebuild the encryption matrix in order to break the Playfair cypher. The total number of matrices is about 25 factorial = 25! = 1 x 2 x 3... 25. This presupposes that a complete ordering of the 25 characters is used as a key rather than a key word. On a regular computer, this is impossible.

As previously stated, a typical frequency attack against Playfair cyphers with a short ciphertext is challenging. The shotgun hill climbing strategy, on the other hand, can boost this performance. This attack is carried out in the following manner:

1. Make a random permutation of the 25 letters to fill the matrix.
2. Using the current decryption matrix, decrypt
3. The decryption is graded based on how well the decrypted ciphertext matches the predicted frequencies.
4. If the score exceeds a certain level, have a human check the plaintext for accuracy. Continue if it is incorrect.
5. Use one of the following operations to transform the matrix: reflection, row rotation, column rotation, or switching two letters.
6. Return to step 2

Even if the ciphertext is brief, a computer using this method has a reasonable chance of decrypting a Playfair cypher. The procedure's effectiveness can be boosted even more by repeating it numerous times, which may result in varied results due to the process's randomized phases.