

Don Bosco Institute of Technology, Mumbai 400070
Department of Information Technology

Experiment No. : 3

Date: 02/09/22

Title : Playfair Cipher Implementation

Problem Definition : Implement Playfair cipher and illustrate encoding and decoding process on user entered sentence.

Pre-requisite : Any programming knowledge – C, C++, Java, Python and concepts of symmetric cryptography.

Theory :

Playfair cipher is also called Playfair square. It is a cryptographic technique that is used to encrypt the data. The Playfair cipher process is as follows:

- Creation and population of the matrix.
 1. Enter the keyword in the matrix in a row-wise manner, i.e. from left to right and top to bottom.
 2. Skip the duplicate words in the keyword.
 3. Fill the remaining spaces with the rest of the alphabets (A – Z) that were not a part of the keyword.
- Encryption process.
 1. Break the alphabets into groups (each group must contain two values). The encryption processes will be performed on these groups.
 2. If both alphabets in the group are the same, add x after the first alphabet.
 3. If both the alphabet in the group are present in the same row of the matrix, replace them with the alphabets to their immediate right, respectively. If the original group is on the right side of the row, then wrapping around to the row's left side happens.
 4. If both the alphabet in the group are present in the same column, replace them with the alphabets immediate with below, respectively. If the original group is on the bottom side of the row, then wrapping around to the row's top side happens.
 5. If both the alphabet in the group are not in the same row or column, replace them with the alphabets in the same row immediately but at the other pair of corners of the rectangle, which the original group defines.

Procedure/ Algorithm :

```
// 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;

    // 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;
            }
        }
    }
}

```

```

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);

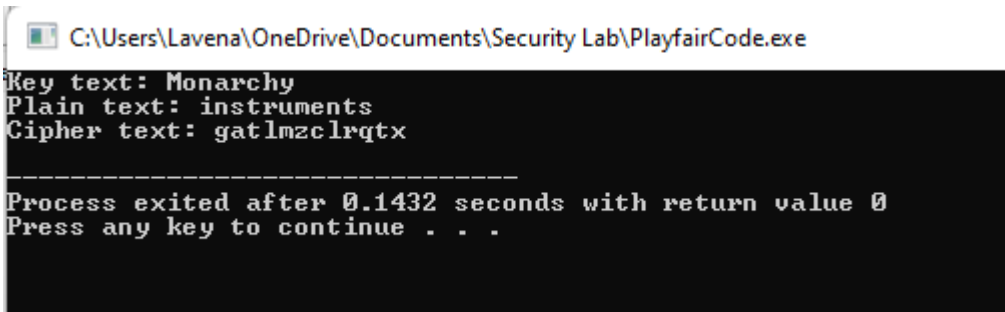
//encrypt using Playfair Cipher
encryptByPlayfairCipher(str, key);

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

return 0;
}

```

Results :



```

C:\Users\Lavena\OneDrive\Documents\Security Lab\PlayfairCode.exe
Key text: Monarchy
Plain text: instruments
Cipher text: gatlmzclrqtx

-----
Process exited after 0.1432 seconds with return value 0
Press any key to continue . . .

```

References :

- 1) <https://www.educba.com/types-of-cipher/>
- 2) <https://www.geeksforgeeks.org/playfair-cipher-with-examples/>

Lab practice (optional) :

L1. Implement Vigenere cipher.

Problem Definition : Implement Playfair cipher and illustrate encoding and decoding process on user entered sentence

Questions (Short, Long, MCQs) (optional) :

S1: Monoalphabetic substitution v/s Polyalphabetic substitution.

L1: Encrypt given sentence using Playfair cipher.

L2: Encrypt given sentence using Vigenere cipher.