Lec. 5

7- Playfair Cipher

The most common definition for playfair cipher is it operates on pairs of letters. The cipher text is broken into blocks, each block contains two letters. The key is 5x5 square consisting of every letter expect j.

The **Playfair cipher** was the first practical digraph substitution cipher. The scheme was invented in **1854** by **Charles Wheatstone** but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II. This was because Playfair is reasonably fast to use and requires no special equipment

Encryption Technique

The algorithm consists of 2 steps:

1. **Generate the key Square (5x5):**

   - The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.
   - The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. **Algorithm to encrypt the plain text:** The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

Example

PlainText: "instruments"

After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'

Explanation: Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter. Here **'z'** is the bogus letter.

Plain Text: "hello"

After Split: 'he' 'lx' 'lo'

Explanation: Here **'x'** is the bogus letter.

Plain Text: "helloe"

After Split: 'he' 'lx' 'lo' 'ez'

Explanation: If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter. Here **'x'** and **'z'** are the bogus letters.

Rules for Encryption

If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).

For example:

Diagraph: "me"

Encrypted Text: cl

Encryption: m -> c e -> l

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

If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

For example:

Diagraph: "st"

Encrypted Text: tl

Encryption: s -> t t -> l

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

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.

For example:

Diagraph: "nt"

Encrypted Text: rq

Encryption: n -> r t -> q

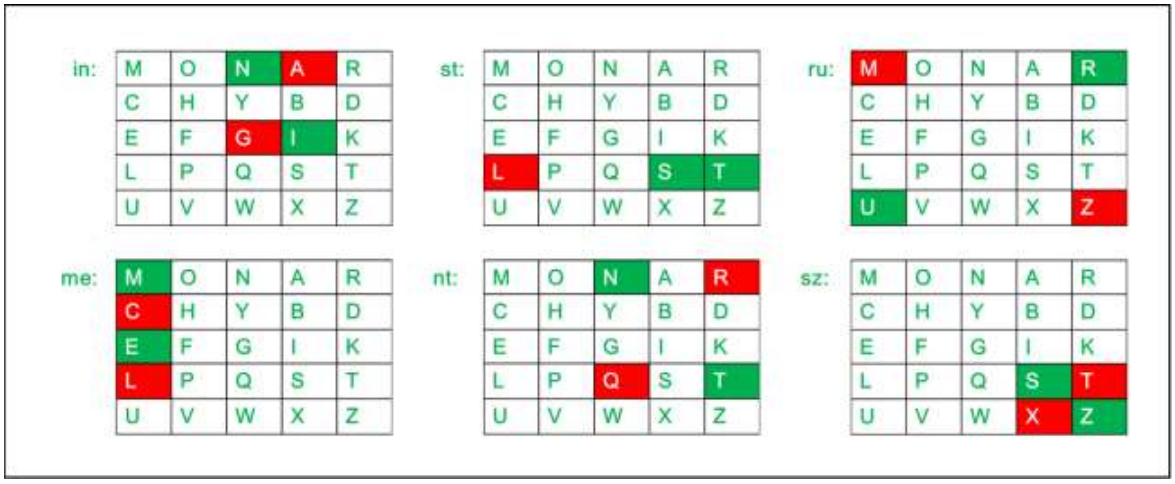
M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

For example:

Palin ext : “instrumentsz”

Encrypted text: gatlmzclrqtx

gatlmzclrqtx



C++

```
#include <bits/stdc++.h>
using namespace std;
// Function to convert the string to lowercase
void toLowerCase(string &plain) {
    int n = plain.size();
    for (int i = 0; i < n; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}
// Function to remove all spaces in a string
void removeSpaces(string &plain) {
    int n = plain.size();
    string temp;
    for (int i = 0; i < n; i++) {
        if (plain[i] != ' ') {
            temp += plain[i];
        }
    }
   plain = temp;
}
```

```
// Function to generate the 5x5 key square
void generateKeyTable(string &key,
        vector<vector<char>> &keyT) {
    int n = key.size();
    // 5x5 key table
    keyT.resize(5, vector<char>(5, 0));
    // a 26 character hashmap
    // to store count of the alphabet
    vector<int> hash(26, 0);
    int i, j, k, flag = 0;
    for (i = 0; i < n; i++) {
        if (key[i] != 'j')
            hash[key[i] - 97] = 2;
    }
    hash['j' - 97] = 1;
    i = 0;
    j = 0;
    for (k = 0; k < n; k++) {
        if (hash[key[k] - 97] == 2) {
            hash[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
```

```
    for (k = 0; k < 26; k++) {
        if (hash[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(vector<vector<char>> &keyT,
        char a, char b, vector<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 make the plain text length to be even
int prepare(string &str) {
    if (str.size() % 2 != 0) {
        str += 'z';
    }
    int n = str.size();
    return n;
}
// Function for performing the encryption
void encrypt(string &str, vector<vector<char>> &keyT) {
    int n = str.size();
    vector<int> arr(4);
    for (int i = 0; i < n; i += 2) {
        search(keyT, str[i], str[i + 1], arr);
        if (arr[0] == arr[2]) {
            str[i] = keyT[arr[0]][(arr[1] + 1) % 5];
            str[i + 1] = keyT[arr[0]][(arr[3] + 1) % 5];
        }
        else if (arr[1] == arr[3]) {
            str[i] = keyT[(arr[0] + 1) % 5][arr[1]];
            str[i + 1] = keyT[(arr[2] + 1) % 5][arr[1]];
        }
        else {
            str[i] = keyT[arr[0]][arr[3]];
```

```
            str[i + 1] = keyT[arr[2]][arr[1]];
        }
    }
}

// Function to encrypt using Playfair Cipher
void encryptByPlayfairCipher(string &str, string &key) {
    vector<vector<char>> keyT;
    removeSpaces(key);
    toLowerCase(key);
    toLowerCase(str);
    removeSpaces(str);
    prepare(str);
    generateKeyTable(key, keyT);
    encrypt(str, keyT);
}

int main() {
    string key = "Monarchy";
    string str = "instruments";
    cout << "Key text: " << key << endl;
    cout << "Plain text: " << str << endl;
    encryptByPlayfairCipher(str, key);
    cout << "Cipher text: " << str << endl;
    return 0;
```

in python

```
# Function to convert the string to lowercase
def toLowerCase(plain):
    n = len(plain)
    result = ""
    for i in range(n):
        if 64 < ord(plain[i]) < 91:
            result += chr(ord(plain[i]) + 32)
        else:
            result += plain[i]
    return result
# Function to remove all spaces in a string
def removeSpaces(plain):
    n = len(plain)
    temp = ""
    for i in range(n):
        if plain[i] != ' ':
            temp += plain[i]
    return temp
# Function to generate the 5x5 key square
def generateKeyTable(key, keyT):
    n = len(key)
    keyT.clear()
    for i in range(5):
        keyT.append([0]*5)
    hashMap = [0]*26
    for i in range(n):
        if key[i] != 'j':
```

```
            hashMap[ord(key[i]) - 97] = 2


    hashMap[ord('j') - 97] = 1
    i = 0
    j = 0
    for k in range(n):
        if hashMap[ord(key[k]) - 97] == 2:
            hashMap[ord(key[k]) - 97] -= 1
            keyT[i][j] = key[k]
            j += 1
            if j == 5:
                i += 1
                j = 0
    for k in range(26):
        if hashMap[k] == 0:
            keyT[i][j] = chr(k + 97)
            j += 1
            if j == 5:
                i += 1
                j = 0
# Function to search for the characters of a digraph
# in the key square and return their position
def search(keyT, a, b, arr):
```

```
    if a == 'j':
        a = 'i'
    if b == 'j':
        b = 'i'
    for i in range(5):
        for j in range(5):
            if keyT[i][j] == a:
                arr[0] = i
                arr[1] = j
            elif keyT[i][j] == b:
                arr[2] = i
                arr[3] = j
# Function to make the plain text length to be even
def prepare(string):
    if len(string) % 2 != 0:
        string += 'z'
    return string
# Function for performing the encryption
def encrypt(string, keyT):
    n = len(string)
    arr = [0]*4
    result = list(string)
    for i in range(0, n, 2):
```

```
        search(keyT, result[i], result[i+1], arr)
        if arr[0] == arr[2]:
            result[i] = keyT[arr[0]][(arr[1] + 1) % 5]
            result[i+1] = keyT[arr[0]][(arr[3] + 1) % 5]
        elif arr[1] == arr[3]:
            result[i] = keyT[(arr[0] + 1) % 5][arr[1]]
            result[i+1] = keyT[(arr[2] + 1) % 5][arr[1]]
        else:
            result[i] = keyT[arr[0]][arr[3]]
            result[i+1] = keyT[arr[2]][arr[1]]
    return ''.join(result)
# Function to encrypt using Playfair Cipher
def encryptByPlayfairCipher(string, key):
    keyT = []
    key = toLowerCase(removeSpaces(key))
    string = toLowerCase(removeSpaces(string))
    string = prepare(string)
    generateKeyTable(key, keyT)
    return encrypt(string, keyT)
key = "Monarchy"
string = "instruments"
print("Key text:", key)
print("Plain text:", string)
```

```
string = encryptByPlayfairCipher(string, key)

print("Cipher text:", string)
```

Output

```
Key text: Monarchy
Plain text: instruments
Cipher text: gatlmzclrqtx
```

Decryption Technique

Decrypting the Playfair cipher is as simple as doing the same process in reverse. The receiver has the same key and can create the same key table, and then decrypt any messages made using that key.

The Algorithm consists of 2 steps:

1. **Generate the key Square(5x5) at the receiver's end:**

   - The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.
   - The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. **Algorithm to decrypt the ciphertext:** The ciphertext is split into pairs of two letters (digraphs).

Note: The **ciphertext** always have **even** number of characters.

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).

For example:

Diagraph: "cl"

Decrypted Text: me

Decryption: c -> m l -> e

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

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).

For example:

Diagraph: "tl"

Decrypted Text: st

Decryption: t -> s l -> t

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

f 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.

For example:

Diagraph: "rq"

Decrypted Text: nt

Decryption: r -> n q -> t

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

or example:

Plain Text: *"gatlmzclrqtx"*

Decrypted Text: *instrumentsz*

Decryption:

(red)-> (green)

ga -> in

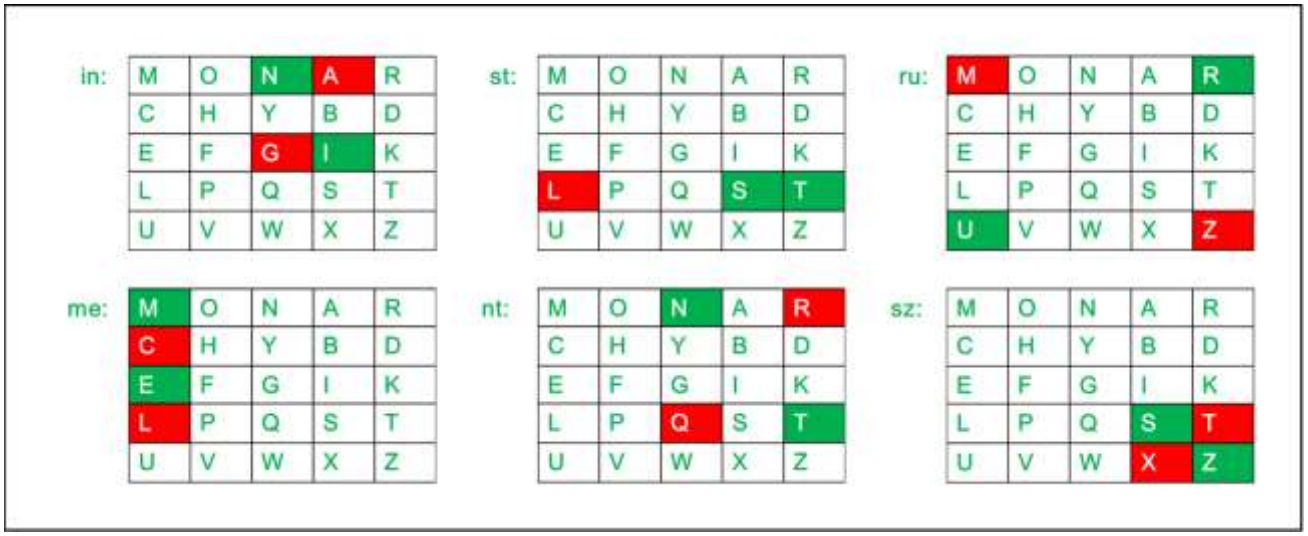
tl -> st

mz -> ru

cl -> me

rq -> nt

tx -> sz



Advantages:

- It is significantly harder to break since the frequency analysis technique used to break simple substitution ciphers is difficult but still can be used on (25*25) = 625 digraphs rather than 25 monographs which is difficult.
- Frequency analysis thus requires more cipher text to crack the encryption.

Disadvantages:

- An interesting weakness is the fact that a digraph in the ciphertext (AB) and it's reverse (BA) will have corresponding plaintexts like UR and RU (and also ciphertext UR and RU will correspond to plaintext AB and BA, i.e. the substitution is self-inverse). That can easily be exploited with the aid of frequency analysis, if the language of the plaintext is known.
    - Another disadvantage is that playfair cipher is a symmetric cipher thus same key is used for both encryption and decryption.