PRACTICAL-2

AIM:

The Playfair cipher was predominantly used by British forces during the Second Boer War (1899-1902) and World War I (1914-1918). Soldier from field wants to send message to base. Implement the cipher to encrypt and decrypt the message.

THEORY:

- 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.

- The Playfair Cipher Algorithm consists basically consists of 3 steps:
    1. Convert Plain Text into Diagraphs
    2. Generate the cipher Matrix
    3. Encrypt the Diagraph with the help of cipher matrix

- Before encrypting the text, you need to divide the Playfair cipher plaintext into digraphs – pairs of two letters. In the case of plaintext with an odd number of letters, add the letter 'X' to the last letter. If there are any double letters in the plain text, replace the second occurrence of the letter with 'X'

- Eg: jogger -> jo gx ge rx

NOTE: we can use other alphabet like 'Z'

Rules for Playfair Cipher Encryption:

- **Case I** – Both the letters in the digraph are in the same row – Consider the letters right of each alphabet. Thus, if one of the digraph letters is the rightmost alphabet in the grid, consider the leftmost alphabet in the same row.
- **Case II** – Both the letters in the digraph are in the same column – Consider the letters below each alphabet. Thus, if one of the digraph letters is the grid's bottommost letter, consider the topmost alphabet in the same column.
- **Case III** – Neither Case I or II is true – Form a rectangle with the two letters in the digraph and consider the rectangle's horizontal opposite corners.

Rules for Playfair Cipher Decryption

- **Case I** – Both the letters in the digraph are in the same row – Consider the letters left of each alphabet. Thus, if one of the digraph letters is the leftmost letter in the grid, consider the rightmost alphabet in the same row.
- **Case II** – Both the letters in the digraph are in the same column – Consider the letters above each alphabet. Thus, if one of the digraph letters is the topmost letter in the grid, consider the bottommost alphabet in the same column.
- **Case III** – Neither Case I or II is true – Form a rectangle with the two letters in the digraph and consider the rectangle's horizontal opposite corners.

ADVANTAGES:

- The algorithm is relatively difficult to crack compare to Caeser Cipher and additive cipher.
- Crpytanalyze is almost not possible.

DISADVANTAGES:

- It only supports alphabets and not numbers.
- It is only limited to English Language
- There is high probability that we won't be getting the real plain text, i.e. The extra added characters like 'X' and replacing 'J' with 'I' will be visible once cipher text is converted to plain text.

PROGRAM CODE:

LANGUAGE OF CODE: PYTHON

```
#NAME: PARTH N PATEL
#ID: 19DCS098
#-----------------------------------------------#
#LAGUAGE OF CODE: PYTHON
#-----------------------------------------------#

# FUNCTION TO CREATE AND INITIALIZE 5x5 MATRIX
def initialize_Matrix():
    return [[0 for x in range(0,5)] for y in range(0,5)]

# FUNCTION TO REMOVE J FROM THE KEY AND REPLACE IT WITH I
def process_Key(key):

    #CREATING LIST TO STORE THE NEW KEY
    processed_Key=list()
```

```
    #MAIN LOGIC OF FUNCTION
    key=key.replace("J","I")
    for letter in key:
        if letter not in processed_Key:
            processed_Key.append(letter)
        else:
            continue
    return processed_Key

#FUNCTION TO GENERATE THE CIPER MATRIX
def generate_Cipher_Matrix(key):
    key=process_Key(key)

    #CALLING THE FUNCTION FOR CREATING THE NEW 5x5 MATRIX
    cipher_Matrix=initialize_Matrix()

    #CREATING A LIST WHICH SERVES AS TEMPORARY MATRIX
    temp_Matrix=list()
    for letter in key:
        temp_Matrix.append(letter)
```

```
    # APPENDING THE REMAINIG THE LETTERS IN THE LIST
    for letter in range(65,91):
        if letter== 74:
            continue
        if chr(letter) not in key:
            temp_Matrix.append(chr(letter))

    # ARRANGING THE LIST IN THE FORM OF THE MATRIX
    index=0
    for i in range(0,5):
        for j in range(0,5):
            cipher_Matrix[i][j]=temp_Matrix[index]
            index+=1
    return cipher_Matrix

# FUNCTION TO LOCATE THE POSITION OF THE CHARACTER IN THE MATRIX
def index_Locator(letter)
    #LIST TO STORE THE LOCATION
    location=list()
    if letter=='J':
        letter='I'
    for i,j in enumerate(cipher_Matrix):
        for k,l in enumerate(j):
            if letter==l:
```

```
            location.append(i)
            location.append(k)
            return location


#FUNCTION FOR THE ENCRYPTING THE PLAIN TEXT TO CIPHER TEXT WITH THE HELP OF THE KEY
#PARAMETERS: PLAIN TEXT AND KEY
def encryption(plain_Text,key)
    cipher_Text="" #STRING TO STORE THE CIPHER TEXT
    i=0
    #MAIN LOGIC
    for letter in range(0,len(plain_Text)+1,2):
        if letter<len(plain_Text)-1:
            if plain_Text[letter]==plain_Text[letter+1]:
                plain_Text=plain_Text[:letter+1]+'X'+plain_Text[letter+1:]
    if len(plain_Text)%2!=0:
        plain_Text+='X'
    print("PROCESSED PLAIN TEXT FOR THE ENCRYPTION : "+plain_Text)
    print()
    while(i<len(plain_Text)):
        location_1=list()
        location_2=list()
        location_1=index_Locator(plain_Text[i])
        location_2=index_Locator(plain_Text[i+1])
```

```
    #RULES FOR THE ENCRYPTION:


    # 1. If both the letters are in the same column: Take the letter below each one
    # 2. If both the letters are in the same row: Take the letter to the right of each one
    # 3. 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.


    if location_1[1]==location_2[1]:
        cipher_Text+=cipher_Matrix[(location_1[0]+1)%5][location_1[1]]
        cipher_Text+=cipher_Matrix[(location_2[0]+1)%5][location_2[1]]
    elif location_1[0]==location_2[0]:
        cipher_Text+=cipher_Matrix[location_1[0]][(location_2[1]+1)%5]
        cipher_Text+=cipher_Matrix[location_2[0]][(location_2[1]+1)%5]
    else:
        cipher_Text+=cipher_Matrix[location_1[0]][location_2[1]]
        cipher_Text+=cipher_Matrix[location_2[0]][location_1[1]
    i=i+2
  return cipher_Text
```

```
#FUNCTION FOR THE DECRYPTION OF THE CIPER TEXT WITH THE HELP
# OF THE SAME KEY
#INPUT: CIPHER TEXT AND THE KEY
def decryption(cipher_Text,key):
    decrypted_Text="" #STRING TO STORE THE DECRYPTED TEXT
    i=0
    #MAIN LOGIC
    while i<len(cipher_Text):
        loc=list()
        loc=index_Locator(cipher_Text[i])
        loc1=list()
        loc1=index_Locator(cipher_Text[i+1])
        if loc[1]==loc1[1]:
            decrypted_Text+=cipher_Matrix[(loc[0]-1)%5][loc[1]]
            decrypted_Text+=cipher_Matrix[(loc1[0]-1)%5][loc1[1]]
        elif loc[0]==loc1[0]:
            decrypted_Text+=cipher_Matrix[loc[0]][(loc[1]-1)%5]
            decrypted_Text+=cipher_Matrix[loc1[0]][(loc1[1]-1)%5]
        else:
            decrypted_Text+=cipher_Matrix[loc[0]][loc1[1]]
            decrypted_Text+=cipher_Matrix[loc1[0]][loc[1]]
        i=i+2
    return decrypted_Text
```

```
plain_Text=input("ENTER THE PLAIN TEXT: ")
key=input("ENTER THE KEY: ")

cipher_Matrix=generate_Cipher_Matrix(key)

cipher_Text=encryption(plain_Text,key)
print("GENERATED CIPHER TEXT: "+cipher_Text)
print()

generated_Plain_Text=decryption(cipher_Text,key)
print("GENERATED PLAIN TEXT : "+generated_Plain_Text)

print()
print("PARTH PATEL\n19DCS098")
```

OUTPUT:

```
C:\Users\Parth Patel>python -u "c:\Users\Parth Patel\
ENTER THE PLAIN TEXT: HIROSHIMA
ENTER THE KEY: PEARLHARBOUR
PROCESSED PLAIN TEXT FOR THE ENCRYPTION : HIROSHIMAX

GENERATED CIPHER TEXT: UDAUMUDSOA

GENERATED PLAIN TEXT : HIROSHIMAX

PARTH PATEL
19DCS098
```

CONCLUSION:

- By performing the above practical, I learned how to implement the PLAYFAIR CIPHER Algorithm, along with its history.

- I also learned about its pros and cons and how it overcomes the limitations of the Caeser Cipher and the Additive Cipher.