Cryptography Techniques – I

Introduction

1. **Caesar Cipher**
   The Caesar Cipher is one of the **oldest and simplest encryption techniques**. It involves **shifting each letter** of the plaintext by a **fixed number of positions** in the alphabet. For instance, a shift of three would turn 'A' into 'D,' 'B' into 'E,' and so on. The encryption and decryption processes are straightforward, but the strength of the cipher depends on the key, which is the shift value.

2. **Brute Force Attack**
   A Brute Force Attack is a straightforward yet **time-consuming** method used to crack encrypted messages. It involves systematically **trying all possible keys** until the correct one is found. While modern ciphers are designed to withstand brute force attacks through large key spaces, some historical ciphers like the Caesar Cipher can be vulnerable due to their small key space.

3. **Playfair Cipher**
   The Playfair Cipher, invented by Charles Wheatstone in 1854, is a **polyalphabetic substitution cipher** that encrypts digraphs (pairs of letters) from the plaintext. It uses a **5x5 matrix** filled with a **keyword** (excluding repeated letters) to determine the letter substitutions. The encryption process involves finding the positions of the two letters in the matrix and applying specific rules to generate the ciphertext.

Programs

1. Caesar Cipher

```
text = input("Enter some text: ")
key = int(input("Enter key: "))

def encrypt(text, key):
    encrypted_text = ""
    for i in text:
        encrypted_text += chr((ord(i) + key) % 255)
    return encrypted_text

def decrypt(text, key):
    decrypted_text = ""
    for i in text:
        decrypted_text += chr((ord(i) - key) % 255)
    return decrypted_text

print("Encrypted Text: ", encrypt(text, key))
print("Decrypted Text: ", decrypt(encrypt(text, key), key))
```

2. Brute Force Attack

 # Method 1

```
password = input("Enter password to guess (alphabets of length 4): ")

alphabets = [
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
    'X', 'Y', "Z",
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
    'x', 'y', "z",
]
for a in alphabets:
    for b in alphabets:
        for c in alphabets:
            for d in alphabets:
                guess = a + b + c + d
                if password == guess:
                    print(guess)
                    break
```

 # Methos 2 (Optimized)

```
password = input("Enter the password: ")
guess = ""
count = 0
for i in range(len(password)):
    character = password[i]
    for a in range(256):
        if character == chr(a):
            guess += character
            break
print(guess)
```

 # Program to find the key for a cipher text

```
ciphertext = input("Enter some ciphertext: ")
text = input("Enter the original text to verify: ")

for key in range(-25, 26):
    decrypted_text = ""
    for i in ciphertext:
        decrypted_text += chr(ord(i) - key)
    print(decrypted_text)
    if decrypted_text == text:
        print("Key:", key)
        break
```

3. Playfair Cipher

```
plain_text = "Attack at palace"
keyword = "Falling Gravity"

# plain_text = input("Enter text to encrypt: ")
# keyword = input("Enter the keyword: ")


def generate_playfair_matrix(key):
    key = key.replace(" ", "").upper()
    key = key.replace("J", "I")
    key_set = set(key)
    alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"
    remaining_chars = [char for char in alphabet if char not in key_set]

    playfair_matrix = []
    for char in key:
        if char not in playfair_matrix:
            playfair_matrix.append(char)
    for char in remaining_chars:
        playfair_matrix.append(char)

    matrix = [[" for i in range(5)] for j in range(5)]

    i = 0
    for j in range(0, len(playfair_matrix), 5):
        matrix[i] = playfair_matrix[j:j + 5]
        i += 1

    return matrix


def prepare_input(text):
    text = text.replace(" ", "").upper()
    text = text.replace("J", "I")
    prepared_text = ""
    i = 0
    while i < len(text):
        if i == len(text) - 1:
            prepared_text += text[i] + "X"
            i += 1
        elif text[i] == text[i + 1]:
            prepared_text += text[i] + "X"
            i += 1
```

```
            else:
                prepared_text += text[i] + text[i + 1]
                i += 2
        return prepared_text


    def find_coordinates(matrix, char):
        row = 0
        col = 0

        for i in range(5):
            for j in range(5):
                if matrix[i][j] == char:
                    return i, j


    def playfair_encrypt(plaintext, key):
        matrix = generate_playfair_matrix(key)
        text = prepare_input(plaintext)
        ciphertext = ""

        for i in range(0, len(text), 2):
            char1 = text[i]
            char2 = text[i + 1]

            row1, col1 = find_coordinates(matrix, text[i])
            row2, col2 = find_coordinates(matrix, text[i + 1])

            if row1 == row2:
                ciphertext += matrix[row1][(col1 + 1) % 5] + matrix[row2][(col2 + 1) % 5]
            elif col1 == col2:
                ciphertext += matrix[(row1 + 1) % 5][col1] + matrix[(row2 + 1) % 5][col2]
            else:
                ciphertext += matrix[row1][col2] + matrix[row2][col1]

        return ciphertext


    def playfair_decrypt(ciphertext, key):
        matrix = generate_playfair_matrix(key)
        text = prepare_input(ciphertext)
        plain_text = ""

        for i in range(0, len(text), 2):
            char1 = text[i]
```

```
        char2 = text[i + 1]

        row1, col1 = find_coordinates(matrix, text[i])
        row2, col2 = find_coordinates(matrix, text[i + 1])

        if row1 == row2:
            plain_text += matrix[row1][(col1 - 1) % 5] + matrix[row2][(col2 - 1) % 5]
        elif col1 == col2:
            plain_text += matrix[(row1 - 1) % 5][col1] + matrix[(row2 - 1) % 5][col2]
        else:
            plain_text += matrix[row1][col2] + matrix[row2][col1]

    return plain_text


print("Cipher Text:", playfair_encrypt(plain_text, keyword))
print("Plain Text:", playfair_decrypt(playfair_encrypt(plain_text, keyword), keyword))
```