SUB: Information Security

Bhuvi Ghosh
60009210191

AY 2023-24 (Semester-V)

Experiment No: 3

Batch: D22

Aim: Design and implement Encryption and Decryption Algorithm for Caesar Cipher / Shift Cipher. Also Perform Brute Force Attack on Ciphers.

Theory:

1. Caesar Cipher / Shift Cipher:

Introduction: Caesar Cipher, also known as Shift Cipher, is a simple substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet. The encryption and decryption process involves shifting the letters by a fixed number, often denoted as the key (K).

Encryption Algorithm:

Let (P) be the plaintext and (K) be the key.
Encrypt each letter (p_i) in (P) using the formula (E(p_i) = (p_i + K) \mod 26), where 26 represents the number of letters in the English alphabet.
Decryption Algorithm:

Let (C) be the ciphertext.
Decrypt each letter (c_i) in (C) using the formula (D(c_i) = (c_i - K + 26) \mod 26).
Example:

Plaintext: ATTACK, Key (K): 3
Encryption: A -> D, T -> W, T -> W, A -> D, C -> F, K -> N
Ciphertext: DWWDFN
2. Brute Force Attack on Ciphers:

Introduction: A brute force attack is an attempt to systematically try all possible keys until the correct one is found. In the context of Caesar Cipher, this involves decrypting the ciphertext with all possible keys.

Brute Force Algorithm:

SUB: Information Security

For each possible key (K) from 1 to 25:
Decrypt the ciphertext using the Caesar Cipher decryption algorithm with key (K).
Check if the resulting plaintext makes sense (e.g., by using a dictionary or language model).
If the result is meaningful, consider it a potential plaintext.
Example:

Ciphertext: DWWDFN
Brute Force:
Attempt with Key (K = 1): CVVCEM
Attempt with Key (K = 2): BUUBDL
...
Attempt with Key (K = 22): BSSBQN
Attempt with Key (K = 23): ARRAPM (potential meaningful plaintext)
Note: Brute force attacks become impractical for larger key spaces, but they are effective against simpler ciphers like the Caesar Cipher.

Implementation: Below is a Python implementation of the Caesar Cipher encryption, decryption, and brute force attack.

```
def caesar_encrypt(plaintext, key):
    ciphertext = ''
    for char in plaintext:
        if char.isalpha():
            offset = ord('A') if char.isupper() else ord('a')
            encrypted_char = chr((ord(char) - offset + key) % 26 + offset)
            ciphertext += encrypted_char
        else:
            ciphertext += char
    return ciphertext

def caesar_decrypt(ciphertext, key):
    return caesar_encrypt(ciphertext, -key)

def brute_force_attack(ciphertext):
    potential_plaintexts = []
    for key in range(1, 26):
        decrypted_text = caesar_decrypt(ciphertext, key)
        # Check if the decrypted text is meaningful
        if any(word in decrypted_text.lower() for word in ["the", "and", "is"]):
            potential_plaintexts.append((key, decrypted_text))
    return potential_plaintexts
```

SUB: Information Security

```
# Example
plaintext = "ATTACK"
key = 3
ciphertext = caesar_encrypt(plaintext, key)
print(f"Plaintext: {plaintext}, Key: {key}")
print(f"Ciphertext: {ciphertext}")

# Brute Force Attack Example
ciphertext_to_attack = "DWWDFN"
potential_plaintexts = brute_force_attack(ciphertext_to_attack)
print("\nBrute Force Attack Results:")
for key, potential_plaintext in potential_plaintexts:
    print(f"Key: {key}, Potential Plaintext: {potential_plaintext}")
```

This example demonstrates the encryption of the plaintext "ATTACK" with a key of 3 and performs a brute force attack on the ciphertext "DWWDFN." The implementation includes functions for encryption, decryption, and brute force attack.

Example:

1) ATTACK K=3
2) ACADEMY K=25

SUB: Information Security

```python
def encrypt(text, key):
    encrypted = ""
    for char in text:
        if 'A' <= char <= 'Z':
            encrypted_char = chr((ord(char) - ord('A') + key) % 26 + ord('A'))
        elif 'a' <= char <= 'z':
            encrypted_char = chr((ord(char) - ord('a') + key) % 26 + ord('a'))
        else:
            encrypted_char = char
        encrypted += encrypted_char
    return encrypted



def decrypt(encrypted_text, key):
    decrypted = ""
    for char in encrypted_text:
        if 'A' <= char <= 'Z':
            decrypted_char = chr((ord(char) - ord('A') - key) % 26 + ord('A'))
        elif 'a' <= char <= 'z':
            decrypted_char = chr((ord(char) - ord('a') - key) % 26 + ord('a'))
        else:
            decrypted_char = char
        decrypted += decrypted_char
    return decrypted



if __name__ == "__main__":
    text = "mihir"
    key = 10



    encrypted_text = encrypt(text, key)
    print("Input text:", text)
    print("Encrypted text:", encrypted_text)



    decrypted_text = decrypt(encrypted_text, key)
    print("Decrypted text:", decrypted_text)
```

```
Input text: mihir
Encrypted text: wsrsb
Decrypted text: mihir
```

SUB: Information Security

```python
from collections import Counter

english_frequencies = {'e': 0.127, 't': 0.091, 'a': 0.082, 'o': 0.075, 'i': 0.070, 'n': 0.067,
                       's': 0.063, 'h': 0.061, 'r': 0.060, 'd': 0.043,
                       'l': 0.040, 'c': 0.028, 'u': 0.028, 'm': 0.024, 'w': 0.023, 'f': 0.022,
                       'g': 0.020, 'y': 0.019, 'p': 0.019, 'b': 0.015,
                       'v': 0.010, 'k': 0.008, 'j': 0.002, 'x': 0.001, 'q': 0.001, 'z': 0.001}

def decrypt_caesar(ciphertext, shift):
    decrypted_text = ''
    for char in ciphertext:
        if char.isalpha():
            is_upper = char.isupper()
            char = char.lower()
            decrypted_char = chr(((ord(char) - shift - 97) % 26) + 97)
            if is_upper:
                decrypted_char = decrypted_char.upper()
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text

def calculate_letter_frequency(text):
    cleaned_text = ''.join(char.lower() for char in text if char.isalpha())
    letter_count = Counter(cleaned_text)
    total_letters = len(cleaned_text)
    frequencies = {char: count / total_letters for char, count in letter_count.items()}
    return frequencies
```

SUB: Information Security

```
def decrypt_caesar_with_frequencies(ciphertext, probable_letters):

    # Calculate frequencies of letters in the ciphertext
    ciphertext_frequencies = calculate_letter_frequency(ciphertext)

    for probable_letter in probable_letters:
        best_shift = 0
        best_correlation = 0

        # Calculate the shift for the current probable letter
        shift = (ord(probable_letter) - ord('a')) % 26

        # Decrypt the ciphertext using the identified shift
        decrypted_text = decrypt_caesar(ciphertext, shift)

        # Calculate the correlation between the decrypted text and English frequencies
        correlation = sum(english_frequencies.get(char, 0) * ciphertext_frequencies.get(char, 0)
         for char in decrypted_text.lower())

        if correlation > best_correlation:
            best_correlation = correlation
            best_shift = shift

        if (text == decrypted_text):
            print(f"Cipher Text: {encrypted_text}\nMost probable letter: {probable_letter}\nKey:
            {best_shift}\nDecrypted Text: {decrypted_text}")


if __name__ == "__main__":
    probable_letters = english_frequencies.keys()
    decrypt_caesar_with_frequencies(encrypted_text, probable_letters)
```

```
Cipher Text: wsrsb
Most probable letter: k
Key: 10
Decrypted Text: mihir
```