Name: Tehseen Niazi

Reg no: FA24-BSE-057

Detailed Line-by-Line Explanation and Security Analysis

Line-by-Line Explanation

1. **def caesar_encrypt(text, shift):**
    - **Defines a function to encrypt text using Caesar Cipher with a given shift value.**
2. **encrypted_text = ""**
    - **Initializes an empty string to store the encrypted result.**
3. **for char in text:**
    - **Iterates through each character in the input text.**
4. **if char.isalpha():**
    - **Checks if the character is a letter; non-letters will remain unchanged.**
5. **ascii_offset = 65 if char.isupper() else 97**
    - **Determines the ASCII offset for uppercase ('A') or lowercase ('a') letters to maintain case.**
6. **encrypted_char = chr((ord(char) - ascii_offset + shift) % 26 + ascii_offset)**
    - **Converts the character to its encrypted form:**
        - **ord(char) → ASCII value of the character**
        - **- ascii_offset → normalize to 0–25**
        - **+ shift → apply Caesar shift**
        - **% 26 → wrap around alphabet**
        - **+ ascii_offset → convert back to ASCII value**
7. **encrypted_text += encrypted_char**

- **Adds the encrypted character to the result string.**

8. **else:**
    - **If the character is not a letter (space, punctuation), keep it unchanged.**
9. **return encrypted_text**
    - **Returns the final encrypted message.**
10. **def caesar_decrypt(ciphertext, shift):**
    - **Decrypts a message by shifting letters in the opposite direction.**
11. **return caesar_encrypt(ciphertext, -shift)**
    - **Uses the same encrypt function with negative shift to decrypt.**
12. **if __name__ == "__main__":**
    - **Main block to run example usage.**

Screenshot:

```python
# Function to encrypt a message using Caesar cipher
def caesar_encrypt(text, shift):
    encrypted_text = ""  # Initialize an empty string to store the encrypted result
    for char in text:  # Loop through each character in the input text
        if char.isalpha():  # Check if the character is a letter
            # Determine ASCII offset: 65 for uppercase letters, 97 for lowercase
            ascii_offset = 65 if char.isupper() else 97
            # Shift the character by the given value and wrap around using modulo 26
            encrypted_char = chr((ord(char) - ascii_offset + shift) % 26 + ascii_offset)
            # Add the encrypted character to the result string
            encrypted_text += encrypted_char
        else:
            # If character is not a letter (punctuation, number, space), keep it unchanged
            encrypted_text += char
    return encrypted_text  # Return the final encrypted string

# Function to decrypt a message using Caesar cipher
def caesar_decrypt(ciphertext, shift):
    # Decryption is just encryption with a negative shift
    return caesar_encrypt(ciphertext, -shift)

# Main program block
if __name__ == "__main__":
    message = "Hello, World!"  # Original message to encrypt
    shift_value = 3  # Number of positions to shift in the Caesar cipher
    encrypted = caesar_encrypt(message, shift_value)  # Encrypt the message
    print("Encrypted:", encrypted)  # Print encrypted message
    decrypted = caesar_decrypt(encrypted, shift_value)  # Decrypt the message
    print("Decrypted:", decrypted)  # Print decrypted message (should match original)
```

Encrypted Output:

```
[Running] python -u "c:\Users\Shan Net Point\Desktop\IS
Encrypted: Khoor, Zruog!
```

Decrypted Output:

```
Decrypted: Hello, World!
```

2. Security Analysis of Caesar Cipher

The Caesar Cipher is a classic substitution cipher dating back to Julius Caesar, used for simple message obfuscation. However, in the context of modern information security (aligned with CLO4 on data security), it has significant limitations:

- **Strengths:**
    - Simplicity: Easy to implement and understand, making it educational for learning encryption basics.

    - Efficiency: O(n) time complexity where n is message length, suitable for small messages.
    - Preserves message length and structure, which can be useful in certain low-security scenarios.

- **Weaknesses and Vulnerabilities:**
    - **Brute-Force Attack:** There are only 25 possible shifts (1-25; shift=0 or 26 is identity). An attacker can try all shifts in seconds, even manually. For example, decrypting with every shift reveals the plaintext when coherent text appears.
    - **Frequency Analysis:** English letters have known frequencies (e.g., 'E' ~12.7%). An attacker can map shifted frequencies to standard ones to guess the shift without trying all.
    - **Known-Plaintext Attack:** If part of the plaintext is known (e.g., common words like "the"), the shift can be deduced easily.
    - **No Key Variability:** The "key" is just the shift, lacking complexity compared to modern ciphers like AES, which use large keys and multiple rounds.
    - **Not Secure for Sensitive Data:** In data security, it fails against confidentiality in CIA triad (Confidentiality, Integrity, Availability). It doesn't protect against interception in networks or storage.
    - **Historical Context:** While used in ancient times, it's obsolete today. Modern alternatives like symmetric (AES) or asymmetric (RSA) ciphers provide provable security under computational assumptions.
- **Recommendations for Improvement:**
    - Use in combination with other ciphers (e.g., Vigenère for polyalphabetic substitution).
    - For real-world security, adopt standards like TLS/SSL for data transmission or encrypted storage.
    - In this implementation, adding input validation (e.g., ensure shift is integer) could prevent runtime errors, but it doesn't enhance cryptographic strength.

This analysis highlights why Caesar Cipher is more a teaching tool than a practical security measure in the field of data security.