**Practical No: 1**

**Practical Name: Study and implement a program for Caesar Cipher**

**Roll No: 22BCP250**

**Name: Anusi Patel**

**Division and batch: Div4 G7**

**Original Approach:**

**Introduction:** The Caesar Cipher is one of the simplest and oldest encryption techniques, named after Julius Caesar, who reportedly used it to secure his military communications. This method is a type of substitution cipher, where each letter in the plaintext is replaced by a letter a fixed number of positions down the alphabet**.**

**How It Works:**

1. Shift or Key: The core concept of the Caesar Cipher is the shift or key, which determines how many positions each letter in the plaintext will be moved. For example, with a shift of 3:
   * A becomes D
   * B becomes E
   * C becomes F
   * ... and so on, wrapping around the alphabet so that Z becomes C.
2. Encryption Process: To encrypt a message, you take each letter of the plaintext and shift it according to the key. Spaces and non-alphabetic characters remain unchanged. For instance, the plaintext "HELLO" with a shift of 3 would be encrypted to "KHOOR".
3. Decryption Process: To decrypt the message, the process is reversed. Each letter is shifted back by the same number of positions. Using the previous example, "KHOOR" would be decrypted back to "HELLO".

**Example:**

Let's consider the plaintext "ATTACK AT DAWN" with a shift of 3:

* A → D
* T → W
* T → W
* A → D
* C → F
* K → N
* (space remains unchanged)
* A → D
* T → W
* (space remains unchanged)
* D → G
* A → D
* W → Z
* N → Q

Thus, "ATTACK AT DAWN" becomes "DWWDFN DW GDZQ".

**Source Code:**

import random

def encrypt\_caesar\_cipher(text, key):

alphabet\_upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

alphabet\_lower = 'abcdefghijklmnopqrstuvwxyz'

result = ""

for char in text:

if char in alphabet\_upper:

P = alphabet\_upper.index(char)

C = (P + key) % 26

result += alphabet\_upper[C]

elif char in alphabet\_lower:

P = alphabet\_lower.index(char)

C = (P + key) % 26

result += alphabet\_lower[C]

elif char.isspace():

result += " "

return result

def decrypt\_caesar\_cipher(text, key):

alphabet\_upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

alphabet\_lower = 'abcdefghijklmnopqrstuvwxyz'

result = ""

for char in text:

if char in alphabet\_upper:

C = alphabet\_upper.index(char)

P = (C - key) % 26

result += alphabet\_upper[P]

elif char in alphabet\_lower:

C = alphabet\_lower.index(char)

P = (C - key) % 26

result += alphabet\_lower[P]

elif char.isspace():

result += " "

return result

plaintext = input("Enter the text to be encrypted: ")

key = random.randint(1, 25)

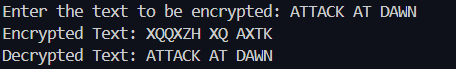
encrypted\_text = encrypt\_caesar\_cipher(plaintext, key)

print(f"Encrypted Text: {encrypted\_text}")

decrypted\_text = decrypt\_caesar\_cipher(encrypted\_text, key)

print(f"Decrypted Text: {decrypted\_text}")

**Output:**

****

**Revised Approach:**

**Introduction:**

The provided code implements a modified version of the **Caesar cipher**, which is a type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet. This implementation enhances the basic Caesar cipher by introducing a dynamic key that varies based on the position of each character in the text.

**Drawbacks of the Caesar Cipher:**

1. **Simplicity**: The Caesar cipher is relatively easy to break, especially with modern computational power. A brute-force attack can try all 25 possible shifts quickly.
2. **Frequency Analysis**: Since it is a substitution cipher, the frequency of letters in the ciphertext will still reflect the frequency of letters in the plaintext. This makes it vulnerable to frequency analysis attacks.
3. **Limited Key Space**: The key space is limited to 25 possible shifts (for the English alphabet), making it less secure compared to more complex ciphers.

**Improvements Over Basic Caesar Cipher:**

The provided code improves the security of the basic Caesar cipher by:

* **Dynamic Key Usage**: Instead of a fixed shift, it uses a key that varies based on the character's position, making it harder to predict the shifts.
* **Random Key Generation**: The use of a randomly generated key adds layer of complexity, as the same plaintext will encrypt to different ciphertexts each time.

**Example:**

Let's use the example text "ATTACK AT DAWN" to illustrate how the code works.

1. **Encryption Process:**

* **Key = “1234567890”**
  + For each character in "ATTACK AT DAWN":
    - A (index 0):
      * K = int(key[0^2 % len(key)]) = int(key[0]) = 1
      * New index: (0 + 1) % 26 = 1 → B
    - T (index 19):
      * K = int(key[1^2 % len(key)]) = int(key[1]) = 2
      * New index: (19 + 2) % 26 = 21 → V
    - T (index 19):
      * K = int(key[2^2 % len(key)]) = int(key[4]) = 5
      * New index: (19 + 5) % 26 = 24 → Y
    - A (index 0):
      * K = int(key[3^2 % len(key)]) = int(key[9]) = 9
      * New index: (0 + 9) % 26 = 9 → J
    - C (index 2):
      * K = int(key[4^2 % len(key)]) = int(key[16]) = 6
      * New index: (2 + 6) % 26 = 8 → I
    - K (index 10):
      * K = int(key[5^2 % len(key)]) = int(key[25]) = 5
      * New index: (10 + 5) % 26 = 15 → P
    - Space: remains as " ".
    - A (index 0):
      * K = int(key[6^2 % len(key)]) = int(key[36]) = 6
      * New index: (0 + 6) % 26 = 6 → G
    - T (index 19):
      * K = int(key[7^2 % len(key)]) = int(key[49]) = 9
      * New index: (19 + 9) % 26 = 2 → C
    - Space: remains as " ".
    - D (index 3):
      * K = int(key[8^2 % len(key)]) = int(key[64]) = 4
      * New index: (3 + 4) % 26 = 7 → H
    - A (index 0):
      * K = int(key[9^2 % len(key)]) = int(key[81]) = 1
      * New index: (0 + 1) % 26 = 1 → B
    - W (index 22):
      * K = int(key[10^2 % len(key)]) = int(key[100]) = 0
      * New index: (22 + 0) % 26 = 22 → W
    - N (index 13):
      * K = int(key[11^2 % len(key)]) = int(key[121]) = 1
      * New index: (13 + 1) % 26 = 14 → O
  + Encrypted Text: "BVYJIP GC HBWO"

1. **Decryption Process:**
   * The decryption function will reverse the process using the same key, yielding the original text.

**Source Code:**

import random

def encrypt\_caesar\_cipher(text, key):

alphabet\_upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

alphabet\_lower = 'abcdefghijklmnopqrstuvwxyz'

result = ""

for i in range(len(text)):

char = text[i]

if char in alphabet\_upper:

if char != ' ':

C = alphabet\_upper.index(char)

K = int(key[i \*\* 2 % len(key)])

P = (C + K) % 26

result += alphabet\_upper[P]

else:

result += " "

elif char in alphabet\_lower:

if char != ' ':

C = alphabet\_lower.index(char)

K = int(key[i \*\* 2 % len(key)])

P = (C + K) % 26

result += alphabet\_lower[P]

else:

result += " "

else:

result += char

return result

def decrypt\_caesar\_cipher(text, key):

alphabet\_upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

alphabet\_lower = 'abcdefghijklmnopqrstuvwxyz'

result = ""

for i in range(len(text)):

char = text[i]

if char in alphabet\_upper:

if char != ' ':

C = alphabet\_upper.index(char)

K = int(key[i \*\* 2 % len(key)])

P = (C - K) % 26

result += alphabet\_upper[P]

else:

result += " "

elif char in alphabet\_lower:

if char != ' ':

C = alphabet\_lower.index(char)

K = int(key[i \*\* 2 % len(key)])

P = (C - K) % 26

result += alphabet\_lower[P]

else:

result += " "

else:

result += char

return result

def generate\_random\_key(length):

nums = '0123456789'

key = ""

for \_ in range(length):

key += random.choice(nums)

return key

text = input("Enter the text to encrypt: ")

key = generate\_random\_key(len(text.replace(" ", "")))

encrypted\_text = encrypt\_caesar\_cipher(text, key)

decrypted\_text = decrypt\_caesar\_cipher(encrypted\_text, key)

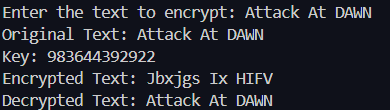
print("Original Text:", text)

print("Key:", key)

print("Encrypted Text:", encrypted\_text)

print("Decrypted Text:", decrypted\_text)

**Output:**

****

**Conclusion:**

The revised approach provides a valuable learning opportunity by introducing concepts of randomness and dynamic key generation in cryptography. However, while these modifications offer some security improvements, they are insufficient for protecting sensitive data in real-world applications. The exercise highlights the importance of key complexity and the limitations of simple substitution ciphers, paving the way for exploring more advanced and secure cryptographic methods.

**References:**

1. [**IEEE Xplore Full-Text PDF:**](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8553910)(Deepanshu Gautam, Parth Sharma, Poonam Saini, Chandan Agrawal, Dr. Munish Mehta from Department of Computer Applications NIT Kurukshetra)
2. [Modified\_Vigenere\_cipher\_algorithm\_based[1].pdf](file:///C:\Users\Anushi%20Patel\AppData\Local\Microsoft\Windows\INetCache\IE\9VN21C40\Modified_Vigenere_cipher_algorithm_based%5b1%5d.pdf) (Thamer Hassan Hameed, Haval Tariq Sadeeq from University of Duhok, Duhok, Iraq)