DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS23ICPC602 **DATE:**

COURSE NAME: Cryptography and Network Security Laboratory **CLASS: T. Y. BTech**

EXPERIMENT NO. 1

CO/LO: Design secure system using appropriate security mechanism

AIM / OBJECTIVE:

a. Implementation of Ceaser Cipher on alphanumeric data.
b. Implementation of Ceaser Cipher on gray scale image.

THEORY / CONCEPT / ALGORITHM:

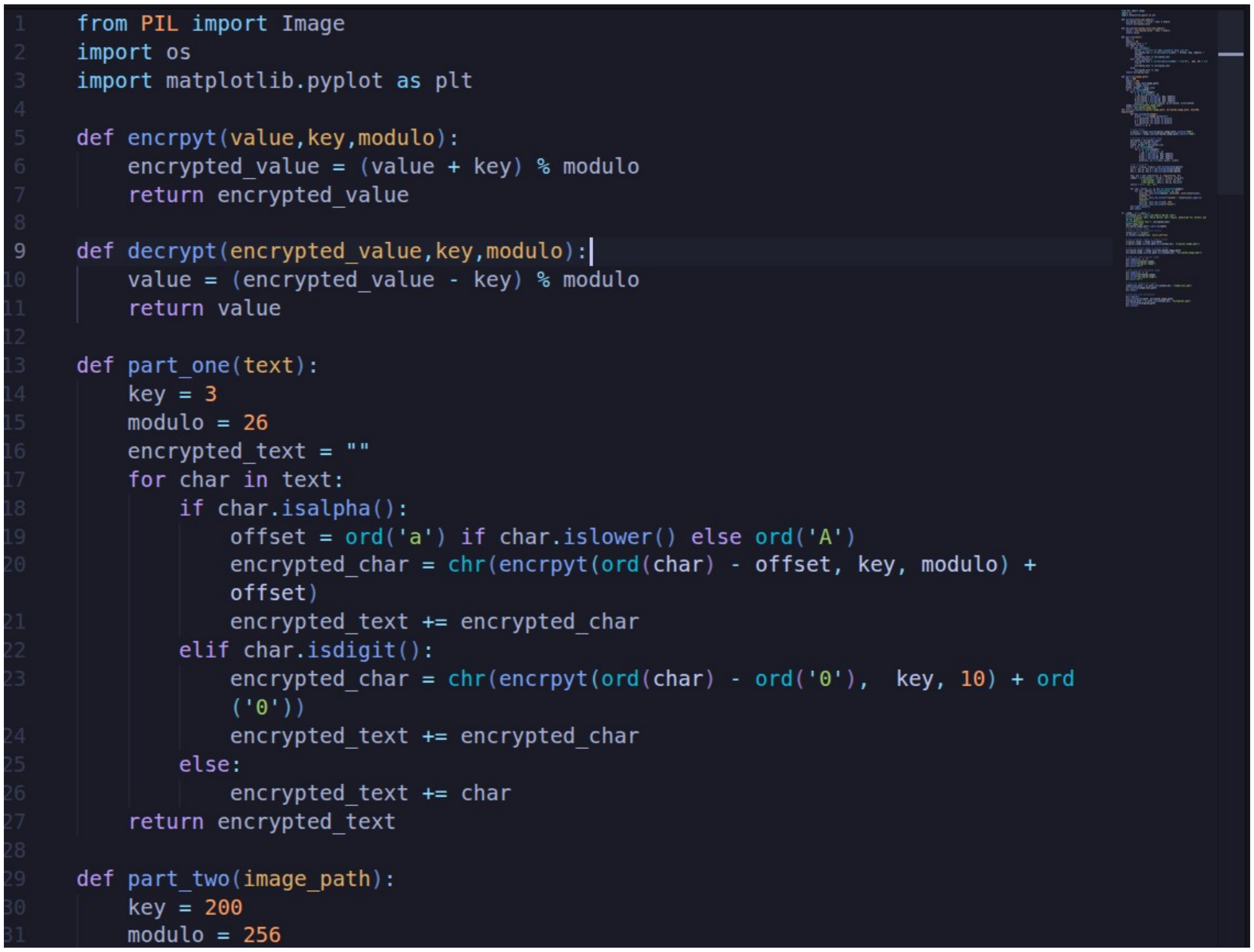
- The **Caesar Cipher** is one of the simplest and oldest methods of encrypting messages, named after Julius Caesar, who reportedly used it to protect his military communications. This technique involves shifting the letters of the alphabet by a fixed number of places. For example, with a shift of three, the letter ‘A’ becomes ‘D’, ‘B’ becomes ‘E’, and so on. Despite its simplicity, the Caesar Cipher formed the groundwork for modern cryptographic techniques.

- To cipher a given text, we need an integer value, known as a shift which indicates the number of positions each letter of the text has been moved down. The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption and decryption of a letter by a shift n can be described mathematically as.

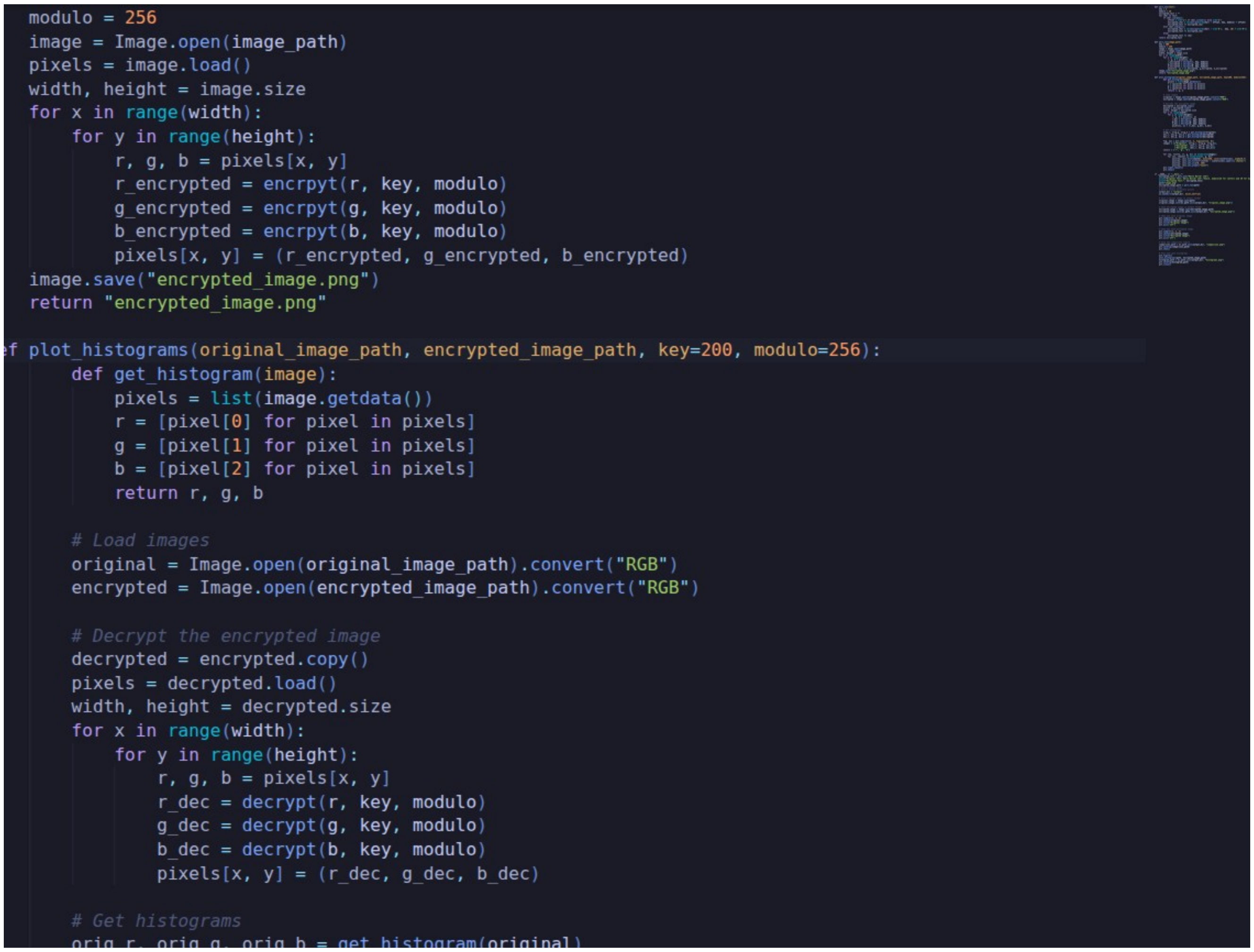
  $E_n(x)=(x+n) \bmod 26$ (Encryption Phase with shift n)

  $D_n(x)=(x-n) \bmod 26$ (Decryption Phase with shift n)

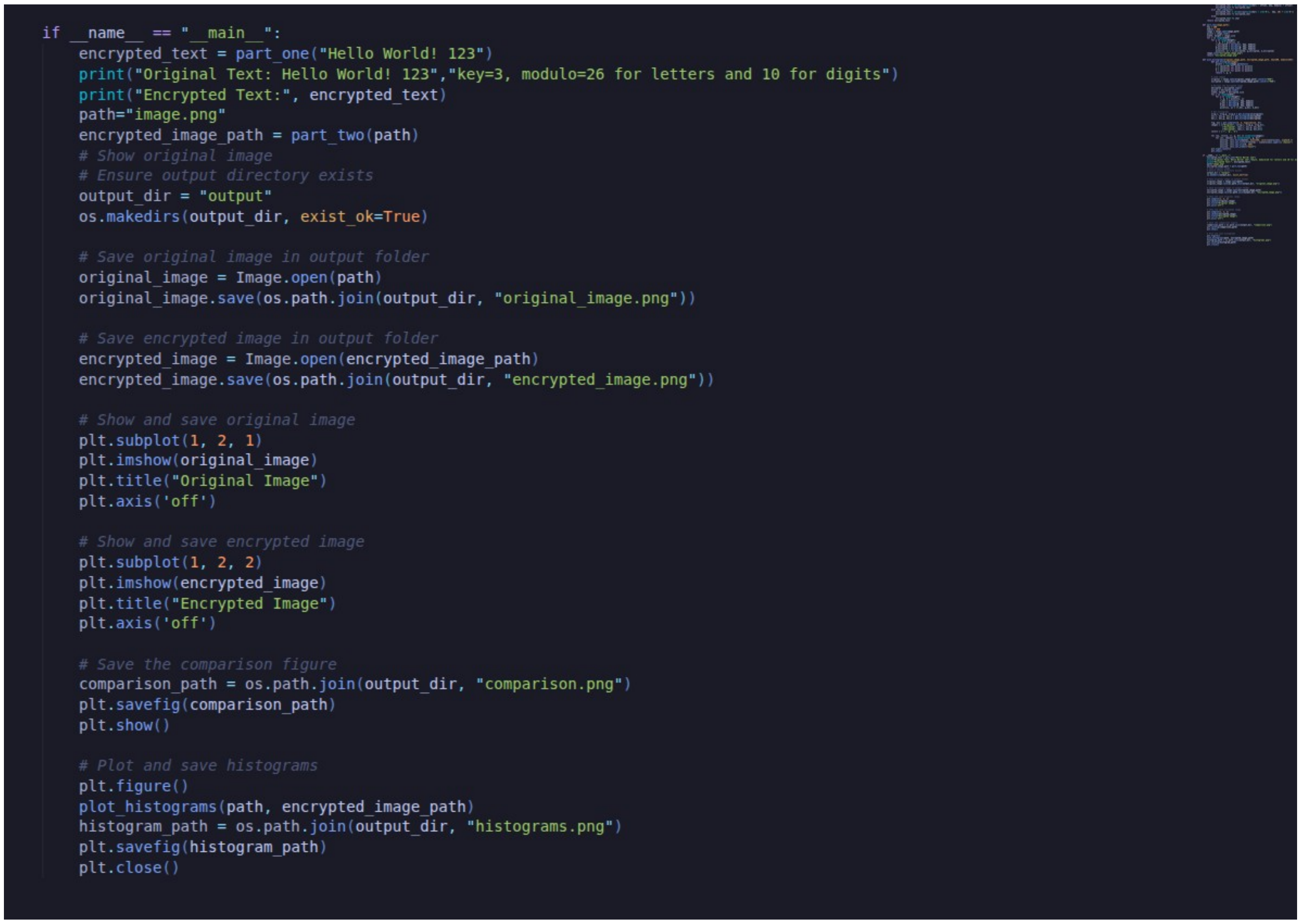
SOURCE CODE:



```
from PIL import Image
import os
import matplotlib.pyplot as plt

def encrpyt(value,key,modulo):
    encrypted_value = (value + key) % modulo
    return encrypted_value

def decrypt(encrypted_value,key,modulo):
    value = (encrypted_value - key) % modulo
    return value

def part_one(text):
    key = 3
    modulo = 26
    encrypted_text = ""
    for char in text:
        if char.isalpha():
            offset = ord('a') if char.islower() else ord('A')
            encrypted_char = chr(encrpyt(ord(char) - offset, key, modulo) +
            offset)
            encrypted_text += encrypted_char
        elif char.isdigit():
            encrypted_char = chr(encrpyt(ord(char) - ord('0'),  key, 10) + ord
            ('0'))
            encrypted_text += encrypted_char
        else:
            encrypted_text += char
    return encrypted_text

def part_two(image_path):
    key = 200
    modulo = 256
```

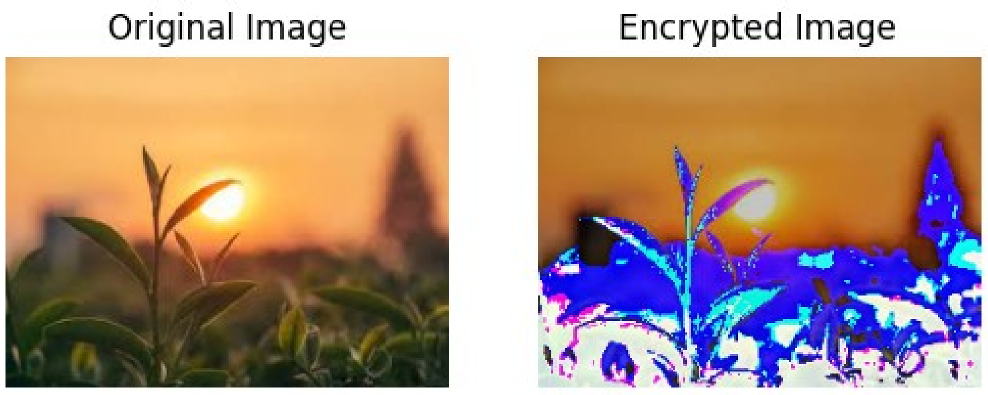


```
    modulo = 256
    image = Image.open(image_path)
    pixels = image.load()
    width, height = image.size
    for x in range(width):
        for y in range(height):
            r, g, b = pixels[x, y]
            r_encrypted = encrpyt(r, key, modulo)
            g_encrypted = encrpyt(g, key, modulo)
            b_encrypted = encrpyt(b, key, modulo)
            pixels[x, y] = (r_encrypted, g_encrypted, b_encrypted)
    image.save("encrypted_image.png")
    return "encrypted_image.png"

ef plot_histograms(original_image_path, encrypted_image_path, key=200, modulo=256):
        def get_histogram(image):
            pixels = list(image.getdata())
            r = [pixel[0] for pixel in pixels]
            g = [pixel[1] for pixel in pixels]
            b = [pixel[2] for pixel in pixels]
            return r, g, b

        # Load images
        original = Image.open(original_image_path).convert("RGB")
        encrypted = Image.open(encrypted_image_path).convert("RGB")

        # Decrypt the encrypted image
        decrypted = encrypted.copy()
        pixels = decrypted.load()
        width, height = decrypted.size
        for x in range(width):
            for y in range(height):
                r, g, b = pixels[x, y]
                r_dec = decrypt(r, key, modulo)
                g_dec = decrypt(g, key, modulo)
                b_dec = decrypt(b, key, modulo)
                pixels[x, y] = (r_dec, g_dec, b_dec)

        # Get histograms
        orig_r, orig_g, orig_b = get_histogram(original)
```

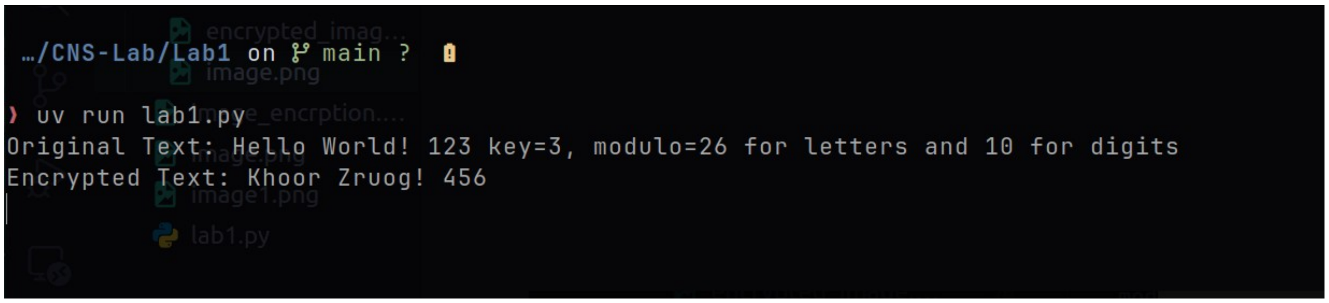


```python
if __name__ == "__main__":
    encrypted_text = part_one("Hello World! 123")
    print("Original Text: Hello World! 123","key=3, modulo=26 for letters and 10 for digits")
    print("Encrypted Text:", encrypted_text)
    path="image.png"
    encrypted_image_path = part_two(path)
    # Show original image
    # Ensure output directory exists
    output_dir = "output"
    os.makedirs(output_dir, exist_ok=True)

    # Save original image in output folder
    original_image = Image.open(path)
    original_image.save(os.path.join(output_dir, "original_image.png"))

    # Save encrypted image in output folder
    encrypted_image = Image.open(encrypted_image_path)
    encrypted_image.save(os.path.join(output_dir, "encrypted_image.png"))

    # Show and save original image
    plt.subplot(1, 2, 1)
    plt.imshow(original_image)
    plt.title("Original Image")
    plt.axis('off')

    # Show and save encrypted image
    plt.subplot(1, 2, 2)
    plt.imshow(encrypted_image)
    plt.title("Encrypted Image")
    plt.axis('off')

    # Save the comparison figure
    comparison_path = os.path.join(output_dir, "comparison.png")
    plt.savefig(comparison_path)
    plt.show()

    # Plot and save histograms
    plt.figure()
    plot_histograms(path, encrypted_image_path)
    histogram_path = os.path.join(output_dir, "histograms.png")
    plt.savefig(histogram_path)
    plt.close()
```

Image Encryption

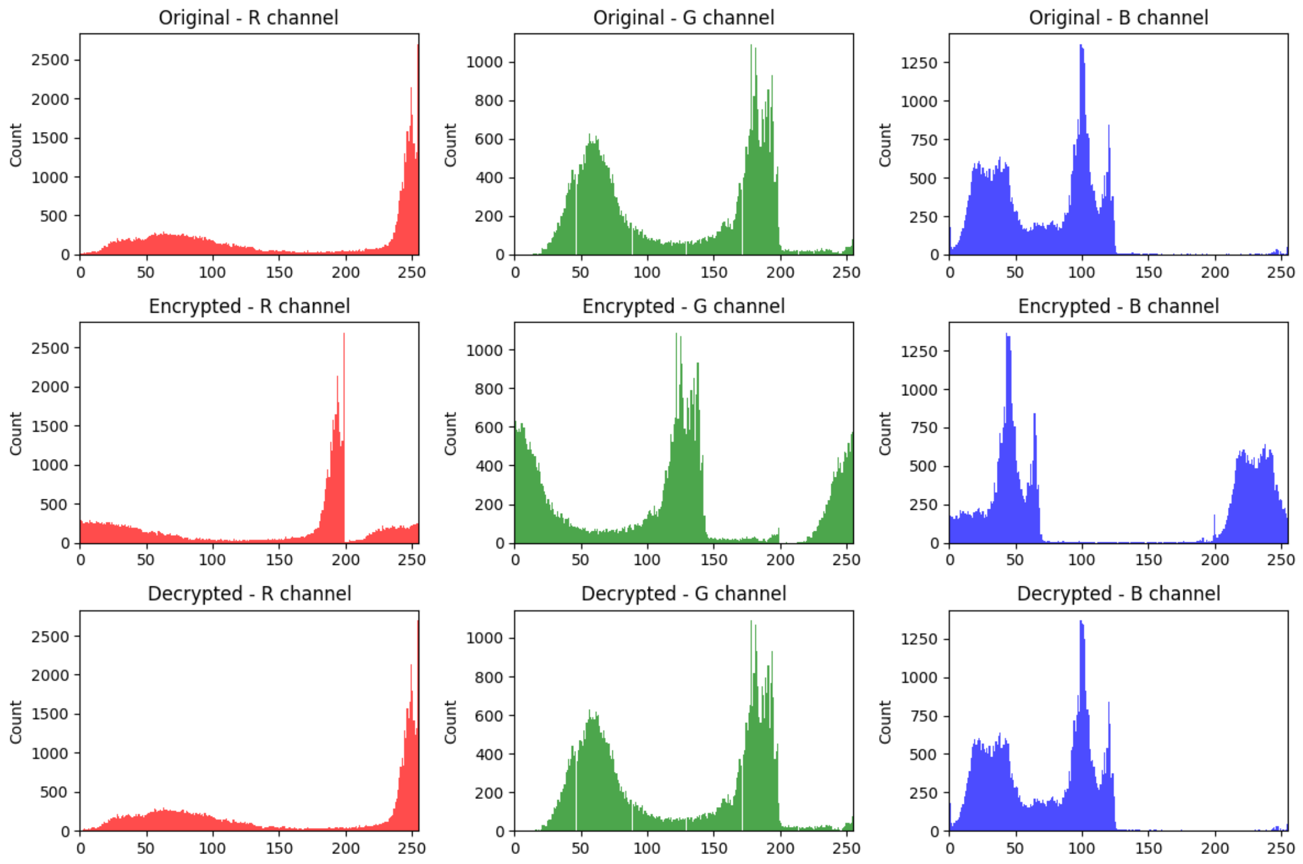


Text Encryption



```
…/CNS-Lab/Lab1 on main ?

❯ uv run lab1.py
Original Text: Hello World! 123 key=3, modulo=26 for letters and 10 for digits
Encrypted Text: Khoor Zruog! 456
```

Histogram



QUESTIONS:

1. Perform a frequency analysis of the encrypted alphanumeric data and compare it to the frequency of the original data.
2. Generate histograms of the pixel values for the original, encrypted, and decrypted images.
3. Show encryption and decryption with the help of an example
4. Cryptanalysis of Caesar Cipher

CONCLUSION: conclude based on the questions asked.

REFERENCES: