# Install required libraries

!pip install phe opencv-python-headless matplotlib scikit-image

# Import necessary libraries

import cv2

import numpy as np

from phe import paillier

from skimage.metrics import peak\_signal\_noise\_ratio as psnr, structural\_similarity as ssim, mean\_squared\_error as mse

import matplotlib.pyplot as plt

import os

# Function Definitions

def generate\_keys():

public\_key, private\_key = paillier.generate\_paillier\_keypair()

return public\_key, private\_key

def scale\_image(image):

rows, cols = image.shape

scaled\_image = np.ones((2 \* rows - 1, 2 \* cols - 1), dtype=int)

scaled\_image[::2, ::2] = image

return scaled\_image

def divide\_into\_blocks(image):

blocks = []

rows, cols = image.shape

for i in range(rows - 2):

for j in range(cols - 2):

block = image[i:i+3, j:j+3]

blocks.append(block)

return blocks

def embed\_data(blocks, secret\_data, public\_key):

embedded\_blocks = []

secret\_index = 0

for block in blocks:

block = block.copy()

central\_pixel = block[1, 1]

if secret\_index < len(secret\_data):

secret\_bit = int(secret\_data[secret\_index])

if (central\_pixel % 2 == 0 and secret\_bit == 1) or (central\_pixel % 2 == 1 and secret\_bit == 0):

central\_pixel += 1

block[1, 1] = central\_pixel

secret\_index += 1

encrypted\_block = np.vectorize(lambda x: public\_key.encrypt(int(x)))(block)

embedded\_blocks.append(encrypted\_block)

return embedded\_blocks

def extract\_data(embedded\_blocks, private\_key):

recovered\_data = ""

recovered\_blocks = []

for block in embedded\_blocks:

decrypted\_block = np.vectorize(lambda x: private\_key.decrypt(x))(block)

recovered\_blocks.append(decrypted\_block)

central\_pixel = decrypted\_block[1, 1]

recovered\_data += str(central\_pixel % 2)

return recovered\_data, np.array(recovered\_blocks)

def reconstruct\_image(blocks, shape):

reconstructed = np.zeros(shape, dtype=int)

block\_size = blocks[0].shape[0]

idx = 0

for i in range(0, shape[0] - block\_size + 1, 2):

for j in range(0, shape[1] - block\_size + 1, 2):

reconstructed[i:i+block\_size, j:j+block\_size] = blocks[idx]

idx += 1

return reconstructed

# Try to Download Lena image (TIFF format)

try:

!wget -q https://www.cs.cmu.edu/~chuck/lennapg/lena.tiff -O lena.tiff

except:

print("Failed to download Lena image. Please upload manually.")

# Check if the file exists

if not os.path.exists("lena.tiff"):

print("Please upload the Lena image manually to the Colab environment.")

raise FileNotFoundError("Lena image not found. Upload `lena.tiff` manually.")

# Load Lena image

lena\_image = cv2.imread("/content/Lena.tiff", cv2.IMREAD\_GRAYSCALE)

# Verify image loading

if lena\_image is None:

raise FileNotFoundError("The Lena image was not loaded correctly. Check the file path or format.")

else:

print("Image loaded successfully. Shape:", lena\_image.shape)

# Secret data

secret\_data = "1010101010101010" # Example binary secret data

# Generate Paillier keys

public\_key, private\_key = generate\_keys()

# Scale the image

scaled\_image = scale\_image(lena\_image)

# Divide into blocks

blocks = divide\_into\_blocks(scaled\_image)

# Embed the data

embedded\_blocks = embed\_data(blocks, secret\_data, public\_key)

# Extract the data and recover the image

recovered\_data, recovered\_blocks = extract\_data(embedded\_blocks, private\_key)

# Reconstruct the recovered image

recovered\_image = reconstruct\_image(recovered\_blocks, scaled\_image.shape)

# Quality Metrics

psnr\_value = psnr(lena\_image, recovered\_image[:lena\_image.shape[0], :lena\_image.shape[1]])

mse\_value = mse(lena\_image, recovered\_image[:lena\_image.shape[0], :lena\_image.shape[1]])

ssim\_value = ssim(lena\_image, recovered\_image[:lena\_image.shape[0], :lena\_image.shape[1]])

# Display Results

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(lena\_image, cmap='gray')

plt.subplot(1, 2, 2)

plt.title("Recovered Image")

plt.imshow(recovered\_image[:lena\_image.shape[0], :lena\_image.shape[1]], cmap='gray')

plt.show()

# Print Quality Metrics

print(f"PSNR: {psnr\_value:.2f} dB")

print(f"MSE: {mse\_value:.2f}")

print(f"SSIM: {ssim\_value:.4f}")