```
!pip install qiskit
!pip install qiskit[visualization]
!pip install qiskit_aer
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.11/dist-packages (1.3.1)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11/dist-packages (from qiskit) (0.15.1)
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3.11/dist-packages (from qiskit) (1.26.4)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/dist-packages (from qiskit) (1.13.1)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-packages (from qiskit) (1.13.1)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-packages (from qiskit) (0.3.9)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.11/dist-packages (from qiskit) (2.8.2)
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from qiskit) (5.4.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from qiskit) (4.12.2)
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python3.11/dist-packages (from qiskit) (0.13.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.0->qiskit) (1.17.0)
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from stevedore>=3.0.0->qiskit) (6.1.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy>=1.3->qiskit) (1.3.0)
Requirement already satisfied: qiskit[visualization] in /usr/local/lib/python3.11/dist-packages (1.3.1)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (0.15.1)
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (1.26.4)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (1.13.1)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (1.13.1)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (0.3.9)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (2.8.2
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (5.4.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (4.12.2)
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (0.13.0
Requirement already satisfied: matplotlib>=3.3 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (3.10.0)
Requirement already satisfied: pydot in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (3.0.4)
Requirement already satisfied: Pillow>=4.2.1 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (11.1.0)
Requirement already satisfied: pylatexenc>=1.4 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (2.10)
Requirement already satisfied: seaborn>=0.9.0 in /usr/local/lib/python3.11/dist-packages (from qiskit[visualization]) (0.13.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3->qiskit[visualizati
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3->qiskit[visualization]
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3->qiskit[visualizat
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3->qiskit[visualizat
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3->qiskit[visualizatio
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3->qiskit[visualizati
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.0->qiskit[visualizatio
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.11/dist-packages (from seaborn>=0.9.0->qiskit[visualization])
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from stevedore>=3.0.0->qiskit[visualization])
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy>=1.3->qiskit[visualization
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn>=0.9.0->qiskit[vis
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn>=0.9.0->qiskit[v
Requirement already satisfied: qiskit_aer in /usr/local/lib/python3.11/dist-packages (0.16.0)
Requirement already satisfied: qiskit>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from qiskit_aer) (1.3.1)
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.11/dist-packages (from qiskit_aer) (1.26.4)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.11/dist-packages (from qiskit_aer) (1.13.1)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/dist-packages (from qiskit_aer) (5.9.5)
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (0.15.
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (1.13.1)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (0.3.9)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (5.4.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (4.12.
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python3.11/dist-packages (from qiskit>=1.1.0->qiskit_aer) (0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.0->qiskit>=1.1.0->qisk
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from stevedore>=3.0.0->qiskit>=1.1.0->qiskit_a
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy>=1.3->qiskit>=1.1.0->qiskit
```

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
from qiskit import*
from qiskit.visualization import plot_histogram
```

Input the 2 plain images

```
# Import image
img = cv.imread("Lena-Grayscale.jpg")
image = cv.imread("Baboon-grayscale.jpg")

# Acquire the dimensions of the original image
width = img.shape[1]
height = img.shape[0]
print(f'Original Dimension of Lena  : {height} x {width}')
width = image.shape[1]
height = image.shape[0]
```

```
print(f'Original Dimension of Baboon : {height} x {width}')
```

```
Original Dimension of Lena  : 512 x 512
Original Dimension of Baboon : 512 x 512
```

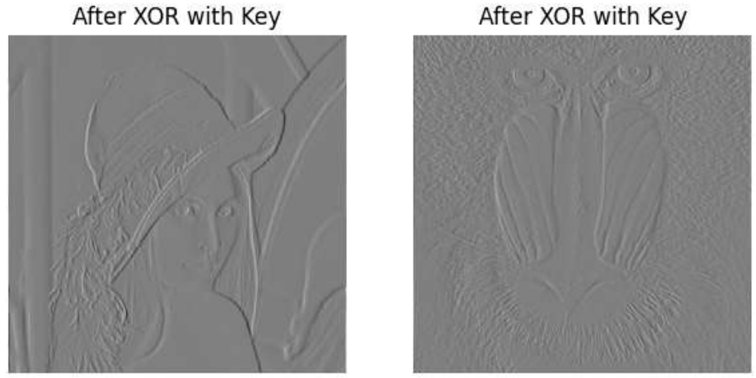
```
#  function to load grayscale image
def load_grayscale_image(image_path):
    img = cv.imread(image_path, cv.IMREAD_GRAYSCALE)
    return img


# function to convert image into quantum-ready format
def encode_image(image_data):
    height, width = image_data.shape
    num_qubits = height * width
    qc = QuantumCircuit(num_qubits)

    # Encoding each pixel's grayscale value as a qubit state
    for i in range(height):
        for j in range(width):
            pixel_val = image_data[i, j]
            qubit_index = i * width + j
            if pixel_val > 128:
                qc.x(qubit_index)  # Apply X gate to "activate" this pixel
    return qc


# Step 1: BRQI Quantum Image Preparation Optimization
def optimized_brqi_prep(qc, image_data):
    height, width = image_data.shape
    num_qubits = 3 * height * width  # 3 qubits per pixel

    # Create qubits for each pixel's bitplanes
    for i in range(height):
        for j in range(width):
            qubit_index = (i * width + j) * 3  # Each pixel has 3 qubits

            # Convert the pixel value to binary and assign to the corresponding qubits
            pixel_value = image_data[i, j]
            binary_value = format(pixel_value, '03b')

            # Apply X gates if the bit is 1
            for k, bit in enumerate(binary_value):
                if bit == '1':
                    qc.x(qubit_index + k)  # Set the qubit to 1 if the bit is 1

            # Apply Hadamard gates to each qubit for superposition
            for k in range(3):
                qc.h(qubit_index + k)

    return qc



# Step 2: Bit-plane based permutation
def classical_bit_plane_permutation(image_data):
    height, width = image_data.shape
    encrypted_image = image_data.copy()
    for i in range(height):
        for j in range(width):
            if (i + j) % 2 == 0:
                # Swap with neighbor pixel
                neighbor_i = i
                neighbor_j = (j + 1) % width
                encrypted_image[i, j], encrypted_image[neighbor_i, neighbor_j] = encrypted_image[neighbor_i, neighbor_j], encrypted_imag
            else:
                # Invert pixel value (NOT operation)
                encrypted_image[i, j] = 255 - encrypted_image[i, j]
    return encrypted_image

lena_image = load_grayscale_image("Lena-Grayscale.jpg")
baboon_image = load_grayscale_image("Baboon-grayscale.jpg")

height, width = lena_image.shape
height,width=baboon_image.shape

num_qubits = height * width
lena_qc = QuantumCircuit(num_qubits)
baboon_qc = QuantumCircuit(num_qubits)

# Step 1: Optimized BRQI Preparation
lena_encrypted_image = optimized_brqi_prep(lena_qc, lena_image)
```

```
baboon_encrypted_image = optimized_brqi_prep(baboon_qc, baboon_image)

# Step 2: Bit-plane based permutation
lena_encrypted_image_1 = classical_bit_plane_permutation(lena_image)
baboon_encrypted_image_1= classical_bit_plane_permutation(baboon_image)

plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(lena_encrypted_image_1, cmap='gray')
plt.title('After Bit-plane Permutation of lena image')
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(baboon_encrypted_image_1, cmap='gray')
plt.title('After Bit-plane Permutation of baboon image')
plt.axis('off')
plt.show()
```

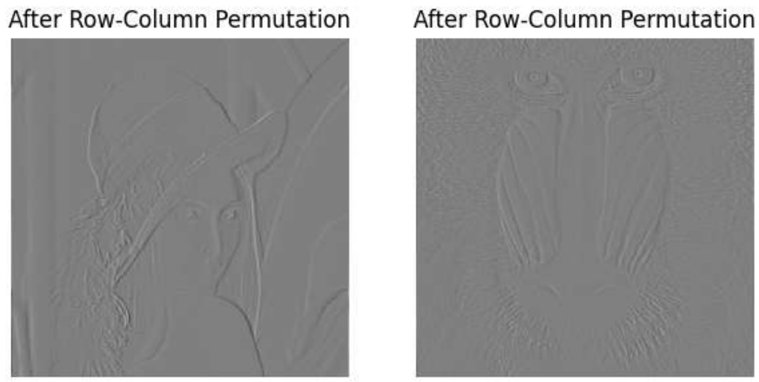
After Bit-plane Permutation of lena image

After Bit-plane Permutation of baboon image

```
# Step 3: Quantum XOR with chaotic quantum key image
# Function to generate quantum key image using Logistic Map
def generate_quantum_key_image(image_shape, r=4.0, x0=0.5): #r-chaotic constant,x-initial value that used to create a chaotic sequence
    height, width = image_shape
    num_iterations = height * width

    sequence = [] #To store the values generated by the Logistic Map
    xn = x0
    for _ in range(num_iterations):
        xn = r * xn * (1 - xn)
        sequence.append(xn)

    # Reshape the sequence to match image shape
    quantum_key_image = np.array(sequence).reshape(image_shape)

    # Threshold the values to create a binary image (0 or 255)
    quantum_key_image[quantum_key_image >= 0.5] = 255
    quantum_key_image[quantum_key_image < 0.5] = 0

    quantum_key_image = quantum_key_image.astype(np.uint8)#quantum_key_image is converted to 8-bit unsigned integers (np.uint8), which i

    return quantum_key_image

# Classical equivalent of xor_with_key_image (using quantum key image)
def classical_xor_with_key_image(image_data, quantum_key_image):
    encrypted_image = image_data.copy()
    encrypted_image = cv.bitwise_xor(image_data, quantum_key_image) # Perform element-wise XOR operation
    return encrypted_image



# Generate chaotic key using Logistic Map
chaotic_key = generate_quantum_key_image(lena_image.shape)
chaotic_key = generate_quantum_key_image(baboon_image.shape)
# Step 3: XOR with chaotic key image
lena_encrypted_image_2 = classical_xor_with_key_image(lena_encrypted_image_1, chaotic_key)
baboon_encrypted_image_2 = classical_xor_with_key_image(baboon_encrypted_image_1, chaotic_key)
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(lena_encrypted_image_2, cmap='gray')
plt.title('After XOR with Key')
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(baboon_encrypted_image_2, cmap='gray')
plt.title('After XOR with Key')
plt.axis('off')
plt.show()
```




```
# Step 4: Row-column based permutation
# Classical equivalent of row_column_permutation
def classical_row_column_permutation(image_data):
    height, width = image_data.shape
    encrypted_image = image_data.copy()
    for i in range(height):
        for j in range(width):
            # Row-based permutation
            if i % 2 == 0:
                neighbor_i = (i + 1) % height
                neighbor_j = j
                encrypted_image[i, j], encrypted_image[neighbor_i, neighbor_j] = encrypted_image[neighbor_i, neighbor_j], encrypted_imag
            else:
                encrypted_image[i, j] = 255 - encrypted_image[i, j]
            # Column-based permutation
            if j % 2 == 0:
                neighbor_i = i
                neighbor_j = (j + 1) % width
                encrypted_image[i, j], encrypted_image[neighbor_i, neighbor_j] = encrypted_image[neighbor_i, neighbor_j], encrypted_imag
    return encrypted_image
# Step 4: Row-column based permutation
lena_encrypted_image_3 = classical_row_column_permutation(lena_encrypted_image_2)
baboon_encrypted_image_3 = classical_row_column_permutation(baboon_encrypted_image_2)
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(lena_encrypted_image_3, cmap='gray')
plt.title('After Row-Column Permutation')
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(baboon_encrypted_image_3, cmap='gray')
plt.title('After Row-Column Permutation')
plt.axis('off')
plt.show()
```

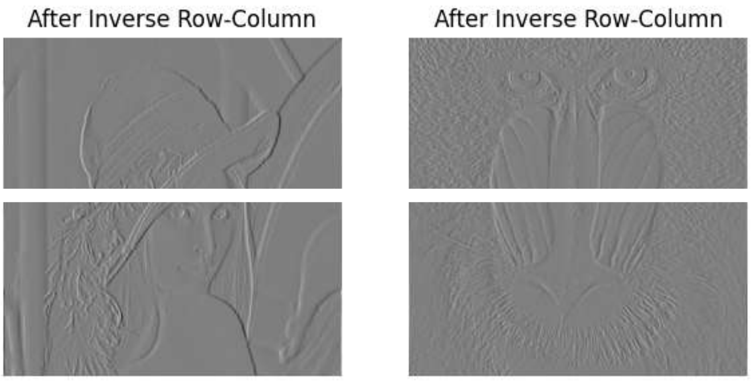


Start coding or generate with AI.

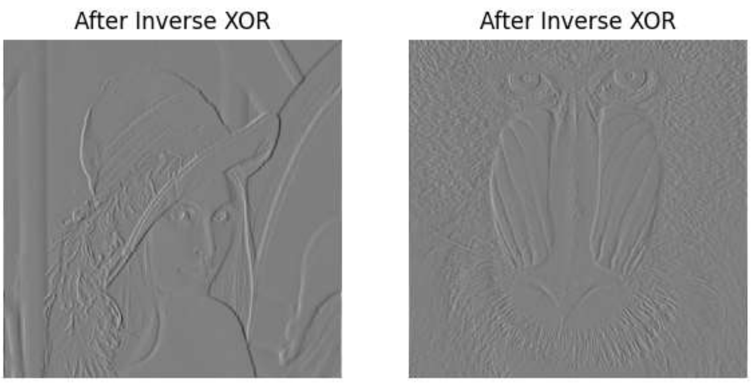
Start coding or generate with AI.

```
# Decryption functions (inverse of encryption functions)
def classical_inverse_row_column_permutation(image_data):
    # Inverse of classical_row_column_permutation
    height, width = image_data.shape
    decrypted_image = image_data.copy()
    for i in range(height - 1, -1, -1):
        for j in range(width - 1, -1, -1):
            # Inverse column-based permutation
```

```
            if j % 2 == 0:
                neighbor_i = i
                neighbor_j = (j + 1) % width
                decrypted_image[i, j], decrypted_image[neighbor_i, neighbor_j] = decrypted_image[neighbor_i, neighbor_j], decrypted_imag
            # Inverse row-based permutation
            if i % 2 == 0:
                neighbor_i = (i + 1) % height
                neighbor_j = j
                decrypted_image[i, j], decrypted_image[neighbor_i, neighbor_j] = decrypted_image[neighbor_i, neighbor_j], decrypted_imag
            else:
                decrypted_image[i, j] = 255 - decrypted_image[i, j]
    return decrypted_image

lena_decrypted_image_3 = classical_inverse_row_column_permutation(lena_encrypted_image_3)
baboon_decrypted_image_3 = classical_inverse_row_column_permutation(baboon_encrypted_image_3)

plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(lena_decrypted_image_3, cmap='gray')
plt.title('After Inverse Row-Column')
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(baboon_decrypted_image_3, cmap='gray')
plt.title('After Inverse Row-Column')
plt.axis('off')
plt.show()
```




```
lena_decrypted_image_4 = classical_xor_with_key_image(lena_decrypted_image_3, chaotic_key)
baboon_decrypted_image_4 = classical_xor_with_key_image(baboon_decrypted_image_3, chaotic_key)

plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(lena_decrypted_image_4, cmap='gray')
plt.title('After Inverse XOR')
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(baboon_decrypted_image_4, cmap='gray')
plt.title('After Inverse XOR')
plt.axis('off')
plt.show()
```

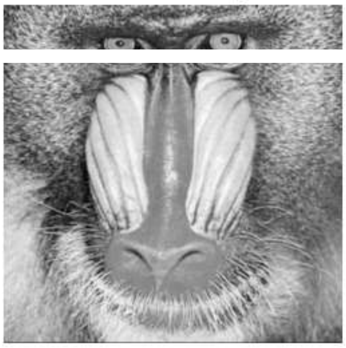



```
def classical_inverse_bit_plane_permutation(image_data):
    # Inverse of classical_bit_plane_permutation
    height, width = image_data.shape
    decrypted_image = image_data.copy()
```

```
    for i in range(height - 1, -1, -1):
        for j in range(width - 1, -1, -1):
            if (i + j) % 2 == 0:
                # Inverse swap with neighbor pixel
                neighbor_i = i
                neighbor_j = (j + 1) % width
                decrypted_image[i, j], decrypted_image[neighbor_i, neighbor_j] = decrypted_image[neighbor_i, neighbor_j], decrypted_imag
            else:
                # Inverse invert pixel value (NOT operation)
                decrypted_image[i, j] = 255 - decrypted_image[i, j]
    return decrypted_image
lena_decrypted_image_3 = classical_inverse_bit_plane_permutation(lena_decrypted_image_4)
baboon_decrypted_image_3 = classical_inverse_bit_plane_permutation(baboon_decrypted_image_4)

plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(lena_decrypted_image_3, cmap='gray')
plt.title('After Inverse Bit-plane')
plt.axis('off')
plt.subplot(1, 4, 2)
plt.imshow(baboon_decrypted_image_3, cmap='gray')
plt.title('After Inverse Bit-plane')
plt.axis('off')
plt.show()
```

After Inverse Bit-plane

After Inverse Bit-plane



Start coding or generate with AI.