TASK 2
Image Encryption and Transformation Tool
Report

1. Introduction

In the current digital age, image security is crucial in various domains including medical imaging, military applications, confidential documentation, and private communications. This project presents a Python-based tool that allows users to perform image **encryption and decryption** using three different techniques:

1. **Pixel-level encryption** using XOR and flip operations.
2. **Fourier Transform-based encryption**, where the image is transformed into the frequency domain and can be reconstructed.
3. **Laplacian transform**, a one-way filtering technique useful for edge detection.

Each method serves a different purpose—ranging from basic image scrambling to frequency-domain analysis and feature enhancement.

---

2. Theory and Background

2.1. Pixel-Level Encryption

Pixel-level encryption uses simple reversible operations:

- **XOR operation** (^): A bitwise logical operation that is commonly used in symmetric key cryptography.
- **Vertical flip** (np.flipud): Inverts the order of pixel rows.

These operations are chosen for their simplicity and reversibility, making encryption and decryption symmetric.

Algorithm:

Let I be the original image and K be the key (an integer between 0–255).

- **Encryption**:
  E=flipud(I)⊕KE = \text{flipud}(I) \oplus K
- **Decryption**:
  D=flipud(E⊕K)D = \text{flipud}(E \oplus K)

Where ⊕\oplus denotes the XOR operation.

2.2. Fourier Transform Encryption

The **Discrete Fourier Transform (DFT)** converts an image from the **spatial domain** to the **frequency domain**, which reveals the image's frequency characteristics. It is typically used for compression and filtering, but can also be used for obfuscation by storing the transformed data securely.

The **2D DFT** of an image f(x, y) of size M×N is defined as:

F(u,v)=∑x=0M−1∑y=0N−1f(x,y)⋅e−j2π(uxM+vyN)F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot e^{-j2\pi \left(\frac{ux}{M} + \frac{vy}{N} \right)}

Its **Inverse DFT** reconstructs the original image:

f(x,y)=1MN∑u=0M−1∑v=0N−1F(u,v)·ej2π(uxM+vyN)f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) \cdot e^{j2\pi \left(\frac{ux}{M} + \frac{vy}{N} \right)}

Workflow:

- **Encrypt**: Apply FFT2, save result to .npy, visualize magnitude using:

Magnitude=20·log10(|F(u,v)|+1)\text{Magnitude} = 20 \cdot \log_{10}(|F(u, v)| + 1)

- **Decrypt**: Load .npy and apply IFFT2 to reconstruct.

This method is *lossless* if the original complex spectrum is stored.

2.3. Laplacian Filtering

The **Laplacian operator** is a second-order derivative that highlights regions of rapid intensity change and is often used for **edge detection**.

In discrete form, the Laplacian kernel is:

[0−10−14−10−10]\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}

The Laplacian is computed by convolving this kernel with the grayscale image:

L(x,y)=∂2f∂x2+∂2f∂y2L(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}

This transform is **one-way** and cannot be reversed, thus not suitable for encryption but useful for feature extraction.

---

3. Methodology and Implementation

3.1. Technologies Used

- **Python 3**
- **NumPy**: For mathematical operations and array handling.
- **Pillow (PIL)**: For reading and saving images.
- **OpenCV**: For image filtering and processing.
- **Matplotlib (optional)**: For plotting during debugging or visualization.

3.2. Modular Design

Functions:

- load_image(path, as_gray=False): Reads image as array.
- save_image(array, mode, path): Saves image in proper format.
- encrypt_pixels, decrypt_pixels: Uses XOR and vertical flipping.
- encrypt_fourier, decrypt_fourier: Transforms to frequency domain.

- apply_laplacian: Computes edge map.

3.3. User Interface (CLI)

The program offers a **command-line interface** where users can select:

1. Encryption method
2. Encrypt/Decrypt
3. File paths and parameters (e.g., keys)

The tool checks for input validity and guides the user with appropriate prompts.

---

4. Algorithm Flow

🔄 Pixel Encryption Algorithm

1. Input image and key.
2. Convert image to NumPy array.
3. Flip image vertically.
4. Apply XOR with the key.
5. Save result.

🔄 Fourier Encryption Algorithm

1. Input grayscale image.
2. Apply 2D FFT: np.fft.fft2(img)
3. Save spectrum (.npy).

4. Display magnitude for visualization (optional).
5. For decryption, load .npy and apply ifft2.

🧠 **Laplacian Algorithm**

1. Input grayscale image.
2. Apply Laplacian filter via cv2.Laplacian.
3. Normalize and save result.

---

5. Security Analysis

Pros:

- XOR encryption is lightweight and fast.
- Fourier spectrum encryption can obfuscate content at a higher level.
- Modular design allows future integration of advanced cryptographic methods.

Cons:

- XOR alone is vulnerable to brute force if the key space is small.
- Fourier method requires storing large .npy files for decryption.
- Laplacian is not reversible, so not a true encryption.

---

6. Use Cases

- Educational tool for understanding basic image encryption
- Pre-processing for advanced watermarking or steganography
- Lightweight encryption for non-critical image data
- Frequency-domain analysis and visualization

CODE

```
import numpy as np
from PIL import Image
import cv2
import os
def load_image(path, as_gray=False):
    """Load an image from the given path."""
    try:
        if as_gray:
            img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
            return img, "L"
        else:
```

```
        img = Image.open(path)
        return np.array(img), img.mode
    except Exception as e:
        print(f"❌ Error loading image: {e}")
        return None, None

def save_image(array, mode, path):
    """Save an image array to disk."""
    try:
        if mode == "L":
            cv2.imwrite(path, array)
        else:
            img = Image.fromarray(array.astype('uint8'), mode)
            img.save(path)
        print(f"✅ Image saved as: {path}")
    except Exception as e:
        print(f"❌ Error saving image: {e}")

# ------------------------------
# Pixel-Level Encryption
# ------------------------------

def encrypt_pixels(img_array, key):
    """Encrypt image using XOR and flip."""
    return np.flipud(img_array) ^ key
```

```
def decrypt_pixels(img_array, key):
    """Decrypt image by reversing encryption steps."""
    return np.flipud(img_array ^ key)

# ------------------------------
# Fourier Transform Encryption
# ------------------------------

def encrypt_fourier(img_array, spectrum_file):
    """Encrypt by applying FFT and saving the spectrum."""
    f = np.fft.fft2(img_array)
    np.save(spectrum_file, f)  # Save complex data
    magnitude = 20 * np.log(np.abs(np.fft.fftshift(f)) + 1)
    return magnitude.astype(np.uint8)

def decrypt_fourier(spectrum_file):
    """Decrypt by applying inverse FFT from saved spectrum."""
    try:
        f = np.load(spectrum_file)
        img_back = np.fft.ifft2(f)
        img_back = np.abs(img_back)
        return np.uint8(img_back)
    except Exception as e:
        print(f"❌ Error decrypting Fourier data: {e}")
```

```
        return None

# ------------------------------
# Laplacian (One-Way Filter)
# ------------------------------

def apply_laplacian(img_array):
    """Apply Laplacian edge detection."""
    lap = cv2.Laplacian(img_array, cv2.CV_64F)
    abs_lap = np.absolute(lap)
    return np.uint8(abs_lap)

# ------------------------------
# Input Validation
# ------------------------------

def get_int_input(prompt, min_val=0, max_val=255):
    """Prompt the user for a valid integer in range."""
    while True:
        try:
            val = int(input(prompt))
            if min_val <= val <= max_val:
                return val
            else:
                print(f"Please enter a number between {min_val} and {max_val}.")
```

```
        except ValueError:
            print("Invalid input. Enter a valid integer.")

# ------------------------------
# Main Interface
# ------------------------------

def main():
    print("\n=== 🔐 Image Encryption Tool ===")
    print("1. Pixel-based Encryption/Decryption")
    print("2. Fourier Transform Encryption/Decryption")
    print("3. Laplacian Transform (edge detection only)")

    choice = input("Choose method (1/2/3): ").strip()

    if choice == "1":
        operation = input("Encrypt (e) or Decrypt (d)? ").strip().lower()
        path = input("Enter image path: ").strip()
        key = get_int_input("Enter encryption key (0-255): ")
        output = input("Output image filename: ").strip()

        img_array, mode = load_image(path)
        if img_array is None: return
```

```
        result = encrypt_pixels(img_array, key) if operation == 'e' else
decrypt_pixels(img_array, key)
        save_image(result, mode, output)

    elif choice == "2":
        operation = input("Encrypt (e) or Decrypt (d)? ").strip().lower()

        if operation == 'e':
            path = input("Enter grayscale image path: ").strip()
            spectrum_file = input("Enter path to save spectrum data (e.g.,
spectrum.npy): ").strip()
            output = input("Output (magnitude image) filename: ").strip()

            img_array, mode = load_image(path, as_gray=True)
            if img_array is None: return

            result = encrypt_fourier(img_array, spectrum_file)
            save_image(result, "L", output)

        elif operation == 'd':
            spectrum_file = input("Enter .npy spectrum file path: ").strip()
            output = input("Output decrypted image filename: ").strip()

            result = decrypt_fourier(spectrum_file)
            if result is not None:
                save_image(result, "L", output)
```

```
        else:
            print("❌ Invalid operation. Use 'e' or 'd'.")

    elif choice == "3":
        path = input("Enter image path: ").strip()
        output = input("Output image filename: ").strip()

        img_array, mode = load_image(path, as_gray=True)
        if img_array is None: return

        lap = apply_laplacian(img_array)
        save_image(lap, "L", output)

    else:
        print("❌ Invalid choice. Please enter 1, 2, or 3.")

# ------------------------------
if __name__ == "__main__":
    main()
```

OUTPUT

Sample image

Pixel manipulation

Fourier transform

Code outputs

1)

```
=== 🔐 Image Encryption Tool ===
1. Pixel-based Encryption/Decryption
2. Fourier Transform Encryption/Decryption
3. Laplacian Transform (edge detection only)
Choose method (1/2/3): 1
Encrypt (e) or Decrypt (d)? e
Enter image path: D:\Internship project\Task 2/image.jpg
Enter encryption key (0-255): 100
Output image filename: output.png
✅ Image saved as: output.png
PS D:\Internship project\Task 2>
```

2)

```
=== 🔐 Image Encryption Tool ===
1. Pixel-based Encryption/Decryption
2. Fourier Transform Encryption/Decryption
3. Laplacian Transform (edge detection only)
Choose method (1/2/3): 2
Encrypt (e) or Decrypt (d)? e
Enter grayscale image path: D:\Internship project\Task 2/image.jpg
Enter path to save spectrum data (e.g., spectrum.npy): spectrum.npy
Output (magnitude image) filename: output.png
✅ Image saved as: output.png
```

7. Conclusion

This project successfully implements a flexible and beginner-friendly **Image Encryption and Processing Tool** that supports both **basic and intermediate-level image transformation techniques**. The integration of spatial-domain (XOR-based), frequency-domain (Fourier Transform), and feature-domain (Laplacian) approaches makes it a versatile learning platform for cybersecurity and image processing enthusiasts.

🔧 **Future Improvements:**

- GUI using tkinter or PyQt
- Password-derived keys using hashlib
- AES integration using pycryptodome
- Watermarking and steganography support
- Network-based transmission module