IMAGE ENCRYPTION USING PIXEL MANIPULATION

---

1. Introduction

Digital images store information as pixel values arranged in a 2D grid. Each pixel in an RGB image consists of three colour components: Red, Green, and Blue, each ranging from 0 to 255.
This project demonstrates **basic image encryption techniques** using **pixel manipulation**, without relying on advanced cryptographic libraries.

The goal is to:

- Understand how images are represented at the pixel level
- Apply reversible operations to encrypt and decrypt images
- Compare the visual impact of different pixel manipulation strategies

---

2. Objectives

The main objectives of this project are:

1. To implement image encryption using a **basic mathematical operation** on pixel values
2. To implement image encryption using **pixel swapping (position permutation)**
3. To make the encrypted image visually unreadable
4. To ensure the encryption process is **fully reversible**
5. To provide a **command-line interface (CLI)** for ease of use

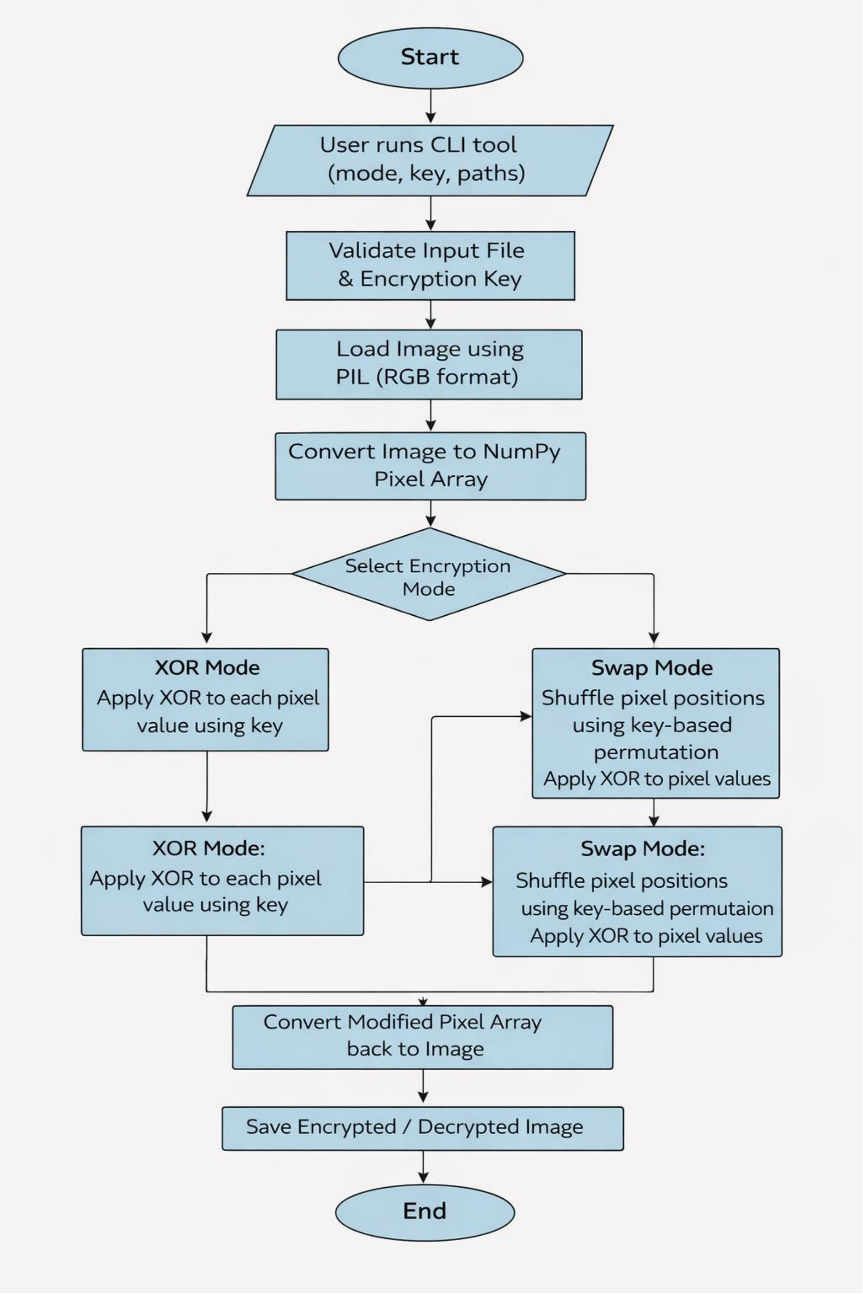
---

3. Tools and Technologies Used

- **Programming Language:** Python 3
- **Libraries:**
    - Pillow (PIL) – Image loading and saving
    - NumPy – Pixel-level manipulation
    - argparse – Command-line argument handling
- **Platform:** Windows / Linux

---

4. System Architecture



---

5. Encryption Modes Implemented

5.1 XOR Mode (Value-Based Encryption)

Description:
In XOR mode, a bitwise XOR operation is applied to each pixel's RGB values using a secret key.

Operation:

Encrypted_Pixel = Original_Pixel XOR Key

Properties:

- Applies a basic mathematical (bitwise) operation to each pixel
- Fully reversible using the same key
- Preserves spatial structure
- Encrypted image may still be partially readable

Purpose:
Demonstrates pixel-value manipulation as required by the task.

5.2 Swap Mode (Position-Based Encryption)

Description:
In swap mode, pixel positions are shuffled using a key-based permutation, followed by XOR value encryption.

Steps:

1. Flatten the pixel array
2. Generate a permutation using the key
3. Rearrange pixel positions
4. Apply XOR to pixel values

Properties:

- Destroys spatial relationships between pixels
- Makes the image visually unreadable
- Fully reversible using the same key
- Stronger visual encryption than XOR alone

Purpose:
Demonstrates pixel swapping and spatial manipulation.

---

6. Command-Line Interface (CLI)

Syntax:

python image_crypto.py --mode <xor|swap> --input <input_image> --output <output_image> --key <0-255>

Example – XOR Mode:

python image_crypto.py --mode xor --input input.png --output encrypted.png --key 123

Example – Swap Mode:

python image_crypto.py --mode swap --input input.png --output encrypted.png --key 123

Running the command again with the same key decrypts the image.

---

7. Results and Observations

XOR Mode:

- Image colours are altered
- Overall structure remains visible
- Suitable for demonstrating mathematical pixel manipulation

Swap Mode:

- Image appears as random noise
- No visible structure or text
- Successfully makes the image unreadable

Decryption:

- Restores the original image exactly
- Pixel-perfect reconstruction confirmed

---

8. Advantages and Limitations

Advantages:

- Simple and easy to understand
- No external cryptographic libraries
- Fully reversible
- Educational and beginner-friendly

Limitations:

- Not secure against real-world attacks
- Uses a single static key
- Designed for learning, not production security

9. Conclusion

This project successfully demonstrates image encryption using pixel manipulation techniques.
The XOR mode fulfils the requirement of applying a basic mathematical operation to each pixel, while the swap mode enhances security by destroying spatial relationships.
The project achieves its educational objectives and provides a clear understanding of image-level encryption concepts.