# *Pixel Manipulation for Image Encryption*

Pallabi Poria

06/10/2025

## Aim

Develop a simple Image encryption tool using pixel manipulation. You can perform operations like swapping pixel values or applying a basic mathematical operation to each pixel. Allow users to encrypt and decrypt images.

## Objective

- To understand how data encryption can be applied to images using basic algorithms.
- To design a Python-based application that allows users to encrypt and decrypt any image.
- To visually demonstrate how pixel-level changes affect an image's confidentiality.
- To ensure that the decrypted image restores the original appearance and colors accurately.

## Description

This project demonstrates how simple mathematical operations can secure digital images.

The user can select an image and apply encryption using a numeric key. The pixel values (RGB channels) are modified based on this key, resulting in an unreadable image.

When the same key is used in decryption mode, the algorithm reverses the operation, restoring the original-colored image.

The GUI is built using Tkinter, while NumPy and Pillow (PIL) handle image processing.

## Features

- Encrypt any image using a custom numeric key.
- Decrypt the imsge back to its original form and color.
- User-friendly graphical interface for easy use.
- Works with common image formats like .jpg, .png, .jpeg.
- Ensures confidentiality through reversible pixel-level manipulation.

## Algorithm/Logic

1. Input Image: The user selects an image from the file system.
2. Read Pixels: Convert the image into a NumPy array to access RGB pixel values.
3. Encryption Process:
   - Each pixel value is modifiied using a key (e.g., *(pixel + key) % 256*).
   - This ensures pixel values stay within the valid range (0-255).
4. Decryption Process:
   - The inverse operation (pixel – key ) % 256 is applied using the same key.
   - This restores the original pixel values.
5. Output: Save and display the encrypted/decrypted image.

## Python Code

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import numpy as np

# --- Encryption ---
def encrypt_image(image_path, key, output_path):
    image = Image.open(image_path).convert('RGB')
    pixels = np.array(image, dtype=np.int32)  # int32 to prevent overflow

    # --- Pixel Shuffling ---
    flat_pixels = pixels.reshape(-1, 3)
    np.random.seed(key)
    indices = np.arange(flat_pixels.shape[0])
    np.random.shuffle(indices)
    shuffled_pixels = flat_pixels[indices]

    # --- Pixel Value Encryption ---
    encrypted_pixels = (shuffled_pixels + key) % 256
    encrypted_image =
Image.fromarray(encrypted_pixels.astype(np.uint8).reshape(pixels.shape), 'RGB')
    encrypted_image.save(output_path)
    messagebox.showinfo("Success", f"Image encrypted and saved as {output_path}")

# --- Decryption ---
def decrypt_image(image_path, key, output_path):
    image = Image.open(image_path).convert('RGB')
    pixels = np.array(image, dtype=np.int32)

    # --- Reverse Pixel Value Encryption ---
    decrypted_pixels = (pixels - key) % 256
    flat_pixels = decrypted_pixels.reshape(-1, 3)

    # --- Reverse Pixel Shuffling ---
    np.random.seed(key)
    indices = np.arange(flat_pixels.shape[0])
    np.random.shuffle(indices)

    # Compute inverse permutation
```

```
    inverse_indices = np.zeros_like(indices)
    inverse_indices[indices] = np.arange(len(indices))

    original_flat = flat_pixels[inverse_indices]
    original_image = original_flat.reshape(pixels.shape)

    decrypted_image = Image.fromarray(original_image.astype(np.uint8), 'RGB')
    decrypted_image.save(output_path)
    messagebox.showinfo("Success", f"Image decrypted and saved as {output_path}")

# --- GUI ---
def select_file():
    path = filedialog.askopenfilename(filetypes=[("Image Files",
".png;.jpg;.jpeg;.bmp")])
    if path:
        entry_file.delete(0, tk.END)
        entry_file.insert(0, path)
        img = Image.open(path).convert('RGB')
        img.thumbnail((200, 200))
        img_tk = ImageTk.PhotoImage(img)
        lbl_preview.img = img_tk
        lbl_preview.config(image=lbl_preview.img)

def perform_encrypt():
    path = entry_file.get()
    if not path:
        messagebox.showerror("Error", "Please select an image")
        return
    try:
        key = int(entry_key.get())
    except:
        messagebox.showerror("Error", "Key must be an integer")
        return
    output_path = filedialog.asksaveasfilename(defaultextension=".png",
filetypes=[("PNG","*.png")])
    if output_path:
        encrypt_image(path, key, output_path)

def perform_decrypt():
    path = entry_file.get()
    if not path:
        messagebox.showerror("Error", "Please select an encrypted image")
        return
```

```
    try:
        key = int(entry_key.get())
    except:
        messagebox.showerror("Error", "Key must be an integer")
        return
    output_path = filedialog.asksaveasfilename(defaultextension=".png",
filetypes=[("PNG","*.png")])
    if output_path:
        decrypt_image(path, key, output_path)

# --- GUI Layout ---
root = tk.Tk()
root.title("RGB-Safe Image Encryption Tool")
root.geometry("400x550")

tk.Label(root, text="Select Image:").pack(pady=5)
entry_file = tk.Entry(root, width=40)
entry_file.pack(pady=5)
tk.Button(root, text="Browse", command=select_file).pack(pady=5)

tk.Label(root, text="Enter Key (integer):").pack(pady=5)
entry_key = tk.Entry(root, width=20)
entry_key.pack(pady=5)

tk.Button(root, text="Encrypt", width=15, command=perform_encrypt).pack(pady=10)
tk.Button(root, text="Decrypt", width=15, command=perform_decrypt).pack(pady=5)

lbl_preview = tk.Label(root)
lbl_preview.pack(pady=10)

root.mainloop()
```

## Code Explanation

**Importing Libraries:**

- tkinter – used for creating GUI interface.
- filedialog – used for selecting files from your system.
- messagebox – used for displaying success or error messages.
- PIL (Pillow) – used for handling and processing images.
- numpy – used for pixel-level mathematical operations.

**Encrypt Function (encrypt_image),**:

1. Takes user input as the encryption key.
2. Opens the selected image and converts it into an RGB NumPy array.
3. Each pixel's RGB value is modified as (pixel + key) % 256, ensuring pixel values stay between 0–255.
4. Converts the encrypted array back to an image and saves it as encrypted_image.png.
5. Displays a success message after encryption.

**Decrypt Function (decrypt_image)**

1. Takes the same key for decryption.
2. Opens the encrypted image and converts it to a NumPy array.
3. Applies the inverse operation (pixel - key) % 256 to restore original pixels.
4. Saves and shows a message for successful decryption.

**Graphical User Interface (GUI)**

1. Uses Tkinter window titled "Image Encryption Tool"
2. Has a text entry for key input (hidden by "*").
3. Includes two buttons — Encrypt Image and Decrypt Image.
4. Allows easy use without needing command-line interaction.

**Encryption Logic:**

The core algorithm is based on pixel manipulation. By adding a numeric key to every pixel's RGB value, the original image becomes distorted and unreadable. Only the correct key can reverse the transformation.

**Decryption Logic:**

Decryption applies the reverse operation — subtracting the key. This restores all pixel values to their original state, bringing back the true colors and appearance of the image.

**Why % 256?:**

RGB values in an image range from 0 to 255. Using % 256 ensures that after any addition or subtraction, the result stays within this valid range.

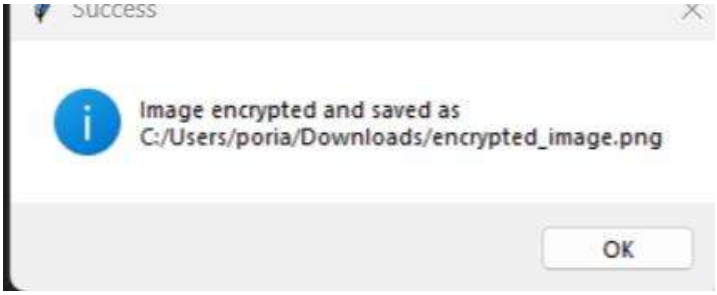
## Sample Output

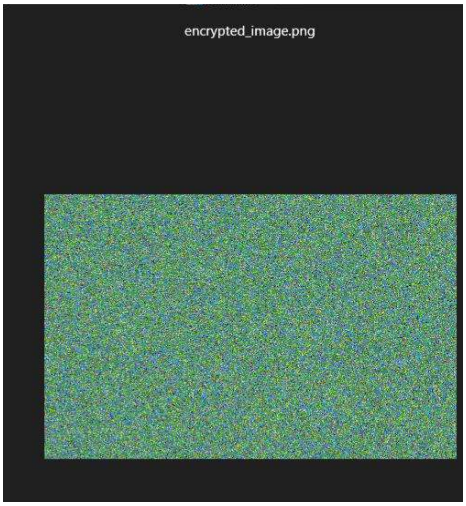
Input: A normal image file, e.g., ship_wreck_480x300.png

**Encryption:**

Key = 50


Image saved as encrypted_image.png (appears distorted and noisy)




**Decryption:**

Same key = 50

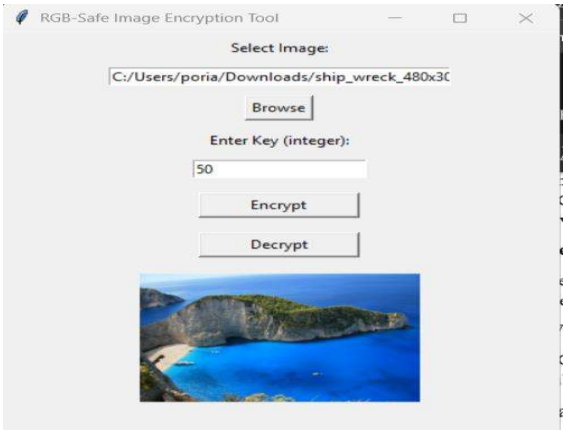
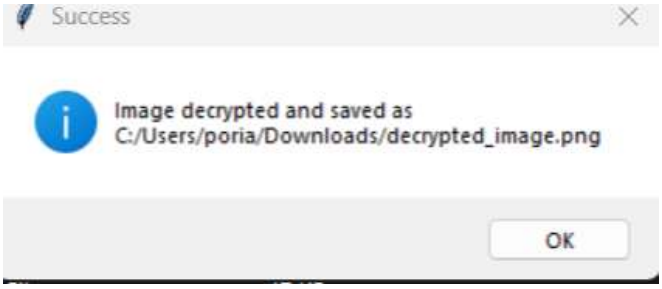
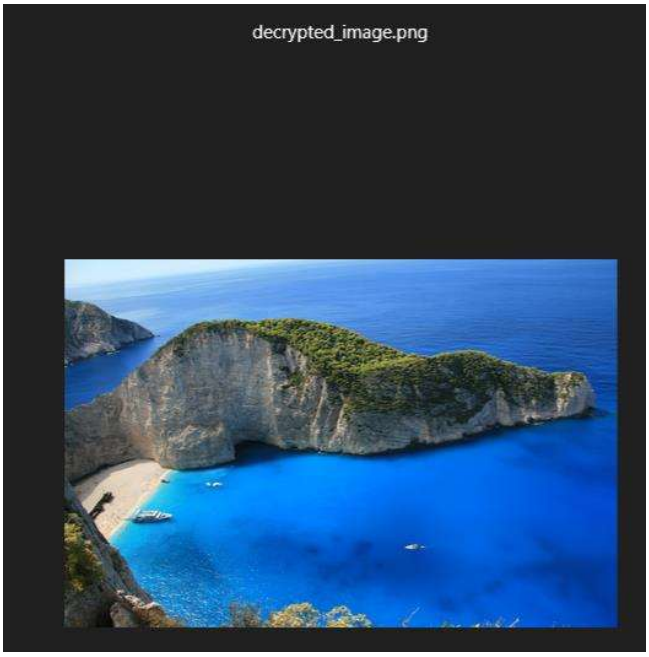



Image saved as decrypted_image.png (returns to the original colorful image)



## Project Justification

In today's digital world, images are widely shared for communication, authentication, and personal use. However, this also raises serious privacy concerns, as sensitive images can easily be accessed or misused without proper protection.

This project addresses that challenge by introducing a simple yet effective image encryption tool based on pixel-level manipulation. Using basic mathematical operations and a secret key, the system securely transforms the image so it cannot be viewed without decryption.

The project also features a user-friendly Python GUI, making encryption accessible even to non-technical users. It effectively demonstrates the core concept of symmetric encryption in a practical, visual form — making it both educational and useful for anyone interested in cybersecurity and data protection.

## Conclusion and Scope

**Conclusion:**

The Image Encryption Tool using Pixel Manipulation effectively demonstrates how image data can be securely protected through simple yet powerful pixel-level transformations. By adding or subtracting a secret key from each pixel's value, the tool ensures that the original image is completely distorted and can only be recovered using the same key — showcasing the essence of symmetric encryption.

Through Python's **NumPy** and **Pillow** libraries, the project efficiently processes images, while **Tkinter** provides an intuitive and user-friendly interface.

Overall, the project successfully meets the internship objectives by delivering a practical, secure, and interactive encryption system. It reinforces the importance of data privacy and highlights how even basic cryptographic principles can be applied to real-world digital security solutions.

**Scope:**

This project has wide educational and practical significance in the field of cybersecurity and data protection.

- **Educational Use:**
  It serves as a great learning resource for students and beginners to understand the fundamentals of image encryption, cryptography concepts, and Python GUI development.
- **Security Awareness:**
  The tool emphasizes the importance of protecting visual data before sharing or storing it, promoting awareness about digital privacy and secure communication.

- **Further Enhancement:**
  Future versions of the tool can integrate stronger encryption algorithms such as **AES** or **RSA** for improved security. Enhancements like password-based key generation, multi-layer encryption, pixel shuffling, and integration with cloud storage or secure file transfer systems could extend its usability to professional and enterprise environments.
- **Practical Use:**
  Individuals and small-scale users can use this lightweight tool to safeguard personal photos or confidential images before sharing them online.
- **Research & Development:**
  The project provides a solid foundation for exploring advanced cryptographic techniques and image data protection methods, encouraging further research and innovation in cybersecurity.