The **Pixel Manipulation for Image Encryption** code step by step, line by line, to make it easy to understand.

### ****Full Code****

from PIL import Image

import numpy as np

def encrypt\_image(image\_path, key):

"""

Encrypts an image using pixel manipulation.

:param image\_path: Path to the input image

:param key: Encryption key (integer)

:return: Encrypted image as a PIL Image object

"""

# Load the image

img = Image.open(image\_path)

img\_array = np.array(img) # Convert image to a NumPy array

# Perform encryption: Add the key to each pixel value

encrypted\_array = (img\_array + key) % 256 # Ensure values remain in the 0-255 range

# Convert the encrypted array back to an image

encrypted\_img = Image.fromarray(encrypted\_array.astype('uint8'))

return encrypted\_img

def decrypt\_image(encrypted\_image, key):

"""

Decrypts an encrypted image using pixel manipulation.

:param encrypted\_image: Encrypted image as a PIL Image object

:param key: Decryption key (integer)

:return: Decrypted image as a PIL Image object

"""

encrypted\_array = np.array(encrypted\_image) # Convert image to a NumPy array

# Perform decryption: Subtract the key from each pixel value

decrypted\_array = (encrypted\_array - key) % 256 # Ensure values remain in the 0-255 range

# Convert the decrypted array back to an image

decrypted\_img = Image.fromarray(decrypted\_array.astype('uint8'))

return decrypted\_img

# Main scriptif \_\_name\_\_ == "\_\_main\_\_":

import os

# Input: Image file and encryption key

image\_path = input("Enter the path to the image file: ")

if not os.path.exists(image\_path):

print("Error: File not found!")

exit()

key = int(input("Enter the encryption key (integer): "))

# Encrypt the image

print("Encrypting the image...")

encrypted\_img = encrypt\_image(image\_path, key)

encrypted\_img.save("encrypted\_image.png")

print("Encrypted image saved as 'encrypted\_image.png'.")

# Decrypt the image

print("Decrypting the image...")

decrypted\_img = decrypt\_image(encrypted\_img, key)

decrypted\_img.save("decrypted\_image.png")

print("Decrypted image saved as 'decrypted\_image.png'.")

### ****Explanation****

#### ****1. Importing Libraries****

from PIL import Image

import numpy as np

* from PIL import Image:  
  This imports the Image module from the Pillow library (PIL). It is used to open, manipulate, and save image files.
* import numpy as np:  
  This imports the numpy library and gives it an alias np. NumPy is a powerful library for working with arrays and performing mathematical operations efficiently.

#### ****2. Encrypt Image Function****

def encrypt\_image(image\_path, key):

"""

Encrypts an image using pixel manipulation.

:param image\_path: Path to the input image

:param key: Encryption key (integer)

:return: Encrypted image as a PIL Image object

"""

This function, encrypt\_image, takes two parameters:

* 1. image\_path: The file path of the image to be encrypted.
  2. key: An integer that acts as the encryption key.

**Returns**: An encrypted version of the image as a PIL Image object.

#### ****3. Loading and Converting the Image****

img = Image.open(image\_path)

img\_array = np.array(img)

* Image.open(image\_path):  
  Opens the image file specified by image\_path.
* np.array(img):  
  Converts the image into a NumPy array. This array contains the RGB (or grayscale) pixel values of the image, making it easier to perform pixel-level manipulations.

#### ****4. Encryption Process****

encrypted\_array = (img\_array + key) % 256

* (img\_array + key):  
  Adds the encryption key (key) to every pixel value in the array. For example, if a pixel value is 120 and the key is 50, the new value will be 170.
* % 256:  
  Ensures the resulting pixel values remain within the range of 0-255 (valid pixel range for 8-bit images).

#### ****5. Convert Encrypted Array Back to Image****

encrypted\_img = Image.fromarray(encrypted\_array.astype('uint8'))

* encrypted\_array.astype('uint8'):  
  Converts the data type of the array to uint8 (unsigned 8-bit integer), which is the standard format for image pixels.
* Image.fromarray():  
  Converts the NumPy array back into a PIL Image object.

#### ****6. Decrypt Image Function****

def decrypt\_image(encrypted\_image, key):

"""

Decrypts an encrypted image using pixel manipulation.

:param encrypted\_image: Encrypted image as a PIL Image object

:param key: Decryption key (integer)

:return: Decrypted image as a PIL Image object

"""

The decrypt\_image function takes:

* 1. encrypted\_image: The encrypted image (as a PIL Image object).
  2. key: The same encryption key used during encryption.

**Returns**: The decrypted image as a PIL Image object.

#### ****7. Decryption Process****

decrypted\_array = (encrypted\_array - key) % 256

* (encrypted\_array - key):  
  Subtracts the encryption key from each pixel value to restore the original pixel values.
* % 256:  
  Ensures the values remain within the 0-255 range, avoiding negative pixel values.

#### ****8. Main Script****

if \_\_name\_\_ == "\_\_main\_\_":

import os

* if \_\_name\_\_ == "\_\_main\_\_":  
  This ensures the script runs only when executed directly, not when imported as a module.
* import os:  
  Imports the os module for checking if the file path exists.

#### ****9. User Input****

image\_path = input("Enter the path to the image file: ")

if not os.path.exists(image\_path):

print("Error: File not found!")

exit()

* input():  
  Prompts the user to enter the path of the image file.
* os.path.exists(image\_path):  
  Checks if the file exists at the given path. If it doesn’t, the script exits with an error message.

#### ****10. Encryption and Saving****

key = int(input("Enter the encryption key (integer): "))

print("Encrypting the image...")

encrypted\_img = encrypt\_image(image\_path, key)

encrypted\_img.save("encrypted\_image.png")print("Encrypted image saved as 'encrypted\_image.png'.")

* key = int(input()):  
  Asks the user to input an encryption key (integer).
* encrypted\_img.save("encrypted\_image.png"):  
  Saves the encrypted image to a file named encrypted\_image.png.

#### ****11. Decryption and Saving****

print("Decrypting the image...")

decrypted\_img = decrypt\_image(encrypted\_img, key)

decrypted\_img.save("decrypted\_image.png")print("Decrypted image saved as 'decrypted\_image.png'.")

* decrypt\_image():  
  Decrypts the encrypted image using the same key.
* decrypted\_img.save("decrypted\_image.png"):  
  Saves the decrypted image to a file named decrypted\_image.png.

### ****How It Works****

1. **Encryption**:
   * Adds the key to each pixel value, wrapping values using % 256.
2. **Decryption**:
   * Subtracts the key from each pixel value, restoring the original image.
3. **Output**:
   * The encrypted image is saved as encrypted\_image.png, and the decrypted image is saved as decrypted\_image.png.