PRODIGY INFOTECH: CS_TASK_02

Pixel Manipulation For Image Encryption

Check out this project on my github : https://github.com/sajilsaju/PRODIGY_CS_02.git

REPORT SUBMITTED BY :

SAJIL MOHAMED V

+91 8714194649

sajilmohamed1234@gmail.com

https://www.linkedin.com/in/sajilmohamed1718/

https://github.com/sajilsaju/

Table Of Contents

1. Introduction

1.1 Project Overview

This project is a Python-based tool for encrypting and decrypting images using pixel manipulation techniques. The core functionality relies on XOR operations to securely alter the pixel values of an image, making it unreadable without the correct key. The tool allows users to input an image, apply encryption, and later decrypt the image using the same key.

Features

- Encryption: Securely encrypts image files by manipulating pixel values using a randomly generated key.
- Decryption: Recovers the original image from the encrypted file using the same key.
- Custom Output Paths: Allows users to specify where the encrypted and decrypted images are saved.
- Simple Interface: Easy-to-use command-line interface for seamless encryption and decryption.

2. Objectives

The main objective of this project to develop a simple image encryption tool using pixel manipulation which can perform operations like swapping pixel values or applying a basic mathematical operation to each pixel and also allow users to encrypt and decrypt their images.

- To develop a tool capable of encrypting and decrypting images through pixel manipulation.
- To understand and apply basic encryption principles to image data.

3. Methodology

3.1 Tools and Technologies Used

- Python 3
- **Pillow (PIL)**: A Python Imaging Library used for opening, manipulating, and saving image files.
- **NumPy**: A library for numerical operations, used for handling and manipulating pixel data.

3.2 Approach

- **Loading the Image**: The input image is loaded using the Pillow library and converted into a NumPy array to access and manipulate pixel values.
- **Key Generation**: A random key is generated, which is used to alter the pixel values. The key is resized to match the dimensions of the image to ensure consistency in encryption and decryption.
- **Encryption**: The XOR operation is applied to each pixel's RGB values using the generated key. This operation encrypts the image by altering the pixel values in a way that makes the image unrecognizable.
- **Saving the Encrypted Image**: The encrypted image is saved to the user-specified path.
- **Decryption**: The XOR operation is reapplied to the encrypted image using the same key, reversing the encryption process and restoring the original image.
- **Saving the Decrypted Image**: The decrypted image is saved to the user-specified path.

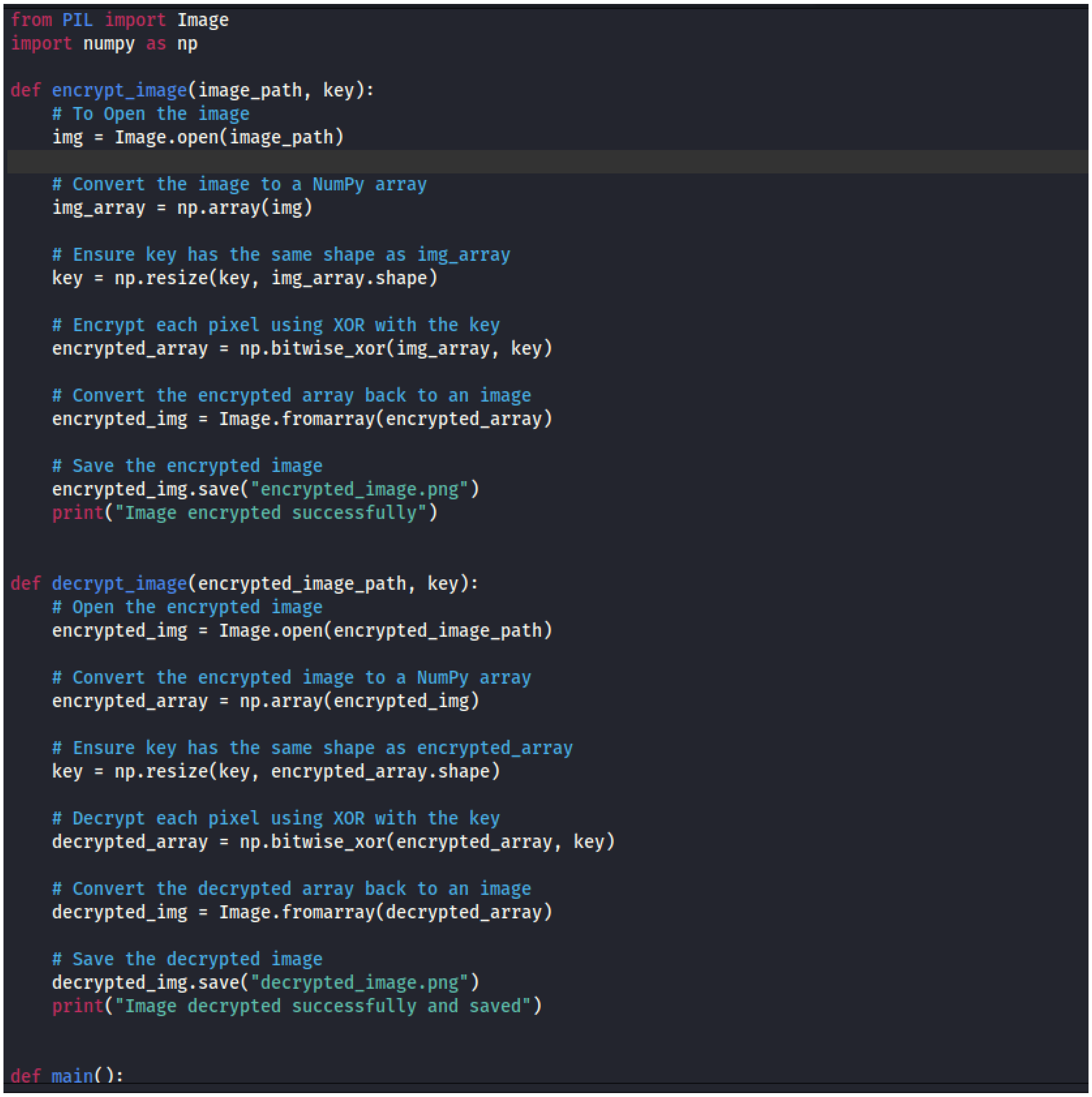
3.3 Program Code

This project were created using python programming language. This program uses the **Pillow** library for image processing.

Source code :

```
from PIL import Image

import numpy as np


def encrypt_image(image_path, key):


    # To Open the image

    img = Image.open(image_path)


    # Convert the image to a NumPy array

    img_array = np.array(img)



    # Ensure key has the same shape as img_array

    key = np.resize(key, img_array.shape)




    # Encrypt each pixel using XOR with the key

    encrypted_array = np.bitwise_xor(img_array, key)
    # Convert the encrypted array back to an image
    encrypted_img = Image.fromarray(encrypted_array)

    # Save the encrypted image
    encrypted_img.save("encrypted_image.png")
    print("Image encrypted successfully.")



def decrypt_image(encrypted_image_path, key):

    # Open the encrypted image
    encrypted_img = Image.open(encrypted_image_path)

    # Convert the encrypted image to a NumPy array
    encrypted_array = np.array(encrypted_img)

    # Ensure key has the same shape as encrypted_array
    key = np.resize(key, encrypted_array.shape)

    # Decrypt each pixel using XOR with the key
    decrypted_array = np.bitwise_xor(encrypted_array, key)
    # Convert the decrypted array back to an image
    decrypted_img = Image.fromarray(decrypted_array)

    # Save the decrypted image
    decrypted_img.save("decrypted_image.png")
    print("Image decrypted successfully.")

def main():

    print("Image Encryption and Decryption of imageusing Pixel Manipulation")

    #image_path = '/home/kali/Desktop/bmw.png'. Enter your image path as inpu
    image_path = input("Enter the path to the image file: ")

    # Generate a random key (you can use any integer as the key)
    key = np.random.randint(0, 256, size=(3,), dtype=np.uint8)

    # To Encrypt the image
    encrypt_image(image_path, key)

    # To Decrypt the image
    decrypt_image("encrypted_image.png", key)
```

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



```python
from PIL import Image
import numpy as np

def encrypt_image(image_path, key):
    # To Open the image
    img = Image.open(image_path)

    # Convert the image to a NumPy array
    img_array = np.array(img)

    # Ensure key has the same shape as img_array
    key = np.resize(key, img_array.shape)

    # Encrypt each pixel using XOR with the key
    encrypted_array = np.bitwise_xor(img_array, key)

    # Convert the encrypted array back to an image
    encrypted_img = Image.fromarray(encrypted_array)

    # Save the encrypted image
    encrypted_img.save("encrypted_image.png")
    print("Image encrypted successfully")


def decrypt_image(encrypted_image_path, key):
    # Open the encrypted image
    encrypted_img = Image.open(encrypted_image_path)

    # Convert the encrypted image to a NumPy array
    encrypted_array = np.array(encrypted_img)

    # Ensure key has the same shape as encrypted_array
    key = np.resize(key, encrypted_array.shape)

    # Decrypt each pixel using XOR with the key
    decrypted_array = np.bitwise_xor(encrypted_array, key)

    # Convert the decrypted array back to an image
    decrypted_img = Image.fromarray(decrypted_array)

    # Save the decrypted image
    decrypted_img.save("decrypted_image.png")
    print("Image decrypted successfully and saved")


def main():
```

4. Implementation

1. **Encryption Process**

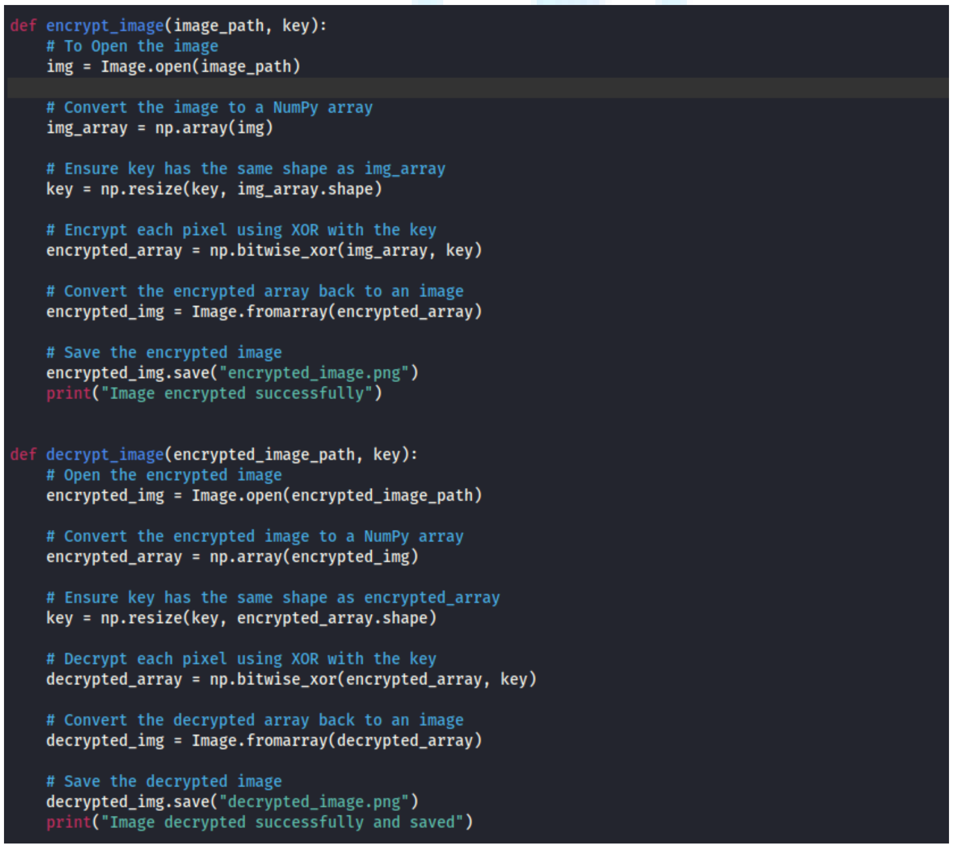
The encryption function accepts an image path, a key, and an output path as inputs. The image is loaded and converted to a NumPy array, and the XOR operation is performed on each pixel using the key. The modified array is then converted back into an image and saved.

2. **Decryption Process**

The decryption function uses the same key and applies the XOR operation to the encrypted image, restoring the original pixel values. The decrypted image is then saved to the specified path.

3. **Code Snippet**

Below is a brief overview of the core functions used in this project:

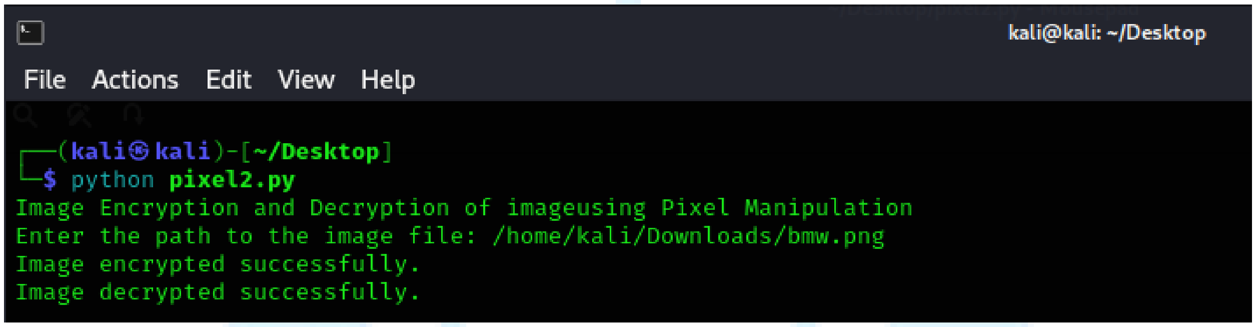


```python
def encrypt_image(image_path, key):
    # To Open the image
    img = Image.open(image_path)

    # Convert the image to a NumPy array
    img_array = np.array(img)

    # Ensure key has the same shape as img_array
    key = np.resize(key, img_array.shape)

    # Encrypt each pixel using XOR with the key
    encrypted_array = np.bitwise_xor(img_array, key)

    # Convert the encrypted array back to an image
    encrypted_img = Image.fromarray(encrypted_array)

    # Save the encrypted image
    encrypted_img.save("encrypted_image.png")
    print("Image encrypted successfully")


def decrypt_image(encrypted_image_path, key):
    # Open the encrypted image
    encrypted_img = Image.open(encrypted_image_path)

    # Convert the encrypted image to a NumPy array
    encrypted_array = np.array(encrypted_img)

    # Ensure key has the same shape as encrypted_array
    key = np.resize(key, encrypted_array.shape)

    # Decrypt each pixel using XOR with the key
    decrypted_array = np.bitwise_xor(encrypted_array, key)

    # Convert the decrypted array back to an image
    decrypted_img = Image.fromarray(decrypted_array)

    # Save the decrypted image
    decrypted_img.save("decrypted_image.png")
    print("Image decrypted successfully and saved")
```

5. Output Results

Successful Encryption and Decryption

The tool successfully encrypted and decrypted images, demonstrating the effectiveness of pixel manipulation techniques for securing image data. The encrypted image was unrecognizable, and the decryption process accurately restored the original image.



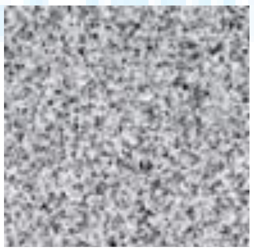
```
┌──(kali㉿kali)-[~/Desktop]
└─$ python pixel2.py
Image Encryption and Decryption of imageusing Pixel Manipulation
Enter the path to the image file: /home/kali/Downloads/bmw.png
Image encrypted successfully.
Image decrypted successfully.
```

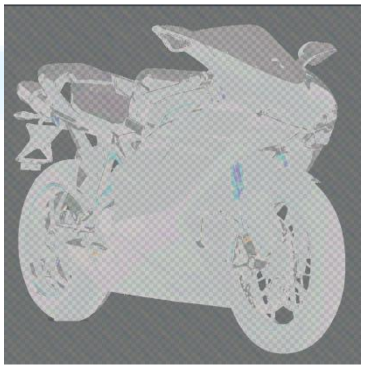
Sample output of the project

For test the working of the code, we can add a sample image to encrypt and decrypt by the pixel manipulation method.

#Encryption of the image:

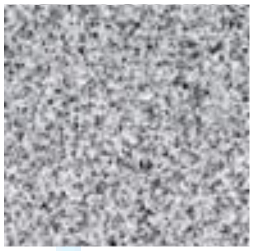
Image used for encryption


encryption process


Output of the encrypted image by pixel manipulation

#Decryption of the image :

Image to decrypt


decryption process

Output of the decrypted image

6. Knowledge Gained

- **Image Processing Basics:** Learned how to handle and manipulate image data using Python's Pillow library, converting images to and from NumPy arrays for pixel-level operations.
- **Pixel Manipulation Techniques:** Gained an understanding of how to alter pixel values for encryption purposes, using mathematical operations like XOR to secure image data.
- **Encryption Fundamentals:** Explored the fundamentals of encryption, including the importance of key management and the reversible nature of the XOR operation for both encryption and decryption.
- **Handling Randomized Keys:** Learned how to generate and manage randomized keys to ensure the encryption process is unique and secure for each image.
- **Custom Output Handling**: Gained experience in allowing users to specify output paths, improving the tool's usability.
- **Error Handling:** Improved skills in validating inputs and handling errors to create a robust and user-friendly tool.

7. Conclusion

The completion of this project marks a significant step in understanding and applying fundamental concepts of cybersecurity, particularly in the realm of image encryption. By leveraging pixel manipulation techniques and the XOR operation, I successfully developed a tool capable of encrypting and decrypting images with a high level of security. This project not only solidified my understanding of image processing and encryption algorithms but also enhanced my ability to implement practical cybersecurity solutions.

Through the challenges faced and the knowledge gained, I've gained valuable experience in handling real-world cybersecurity tasks. This project serves as a foundation for further exploration into more advanced encryption techniques, secure key management, and the development of user-friendly interfaces. The skills acquired here will undoubtedly contribute to my growth as a cybersecurity professional and my ability to protect digital assets effectively.

8. Challenges Faced

- **Key Management**

One of the primary challenges was ensuring that the key used for encryption could be securely stored or transmitted to enable accurate decryption. Future improvements could involve implementing more sophisticated key management techniques.

- **Image Compatibility**

Testing with different image formats and sizes was necessary to ensure that the tool worked consistently across various scenarios

9. Future Work

- **Advanced Encryption Algorithms**: Implement more complex encryption methods to enhance security.
- **Improved Key Management**: Develop a secure method for storing and transmitting keys.
- **User Interface**: Create a graphical user interface (GUI) to make the tool more accessible to non-technical users.
- **Cross-Platform Compatibility**: Ensure the tool works consistently across different operating systems and image formats.

10. References

- **Pillow (PIL) Documentation**: https://pillow.readthedocs.io/en/stable/
- **NumPy Documentation**: NumPy Documentation
- **XOR Encryption fundamentals**