Task-2 Pixel Manipulation for Image Encryption

Report By

Leburi SriRam

Intern At

Prodigy Infotech

July 07,2024

Table of Contents:

Abstract

Introduction

Requirements

Understanding Pixel Manipulation,Image Encryption

Operations

Program

Conclusion

Reference

Abstract:

Develop a simple image encryption tool using pixel manipulation, By performing operations like swapping pixel values or applying a basic mathematical operation to each pixel. Allow users to encrypt and decrypt images.

Introduction:

As an intern at prodigy infotech, my task is clear: Develop a simple image encryption tool using pixel manipulation, By performing operations like swapping pixel values or applying a basic mathematical operation to each pixel. Allow users to encrypt and decrypt images.

This task provides me with unique opportunity to enhance my pixel manipulation,Image encryption and programming skills, focusing on the encryption and decryption of images using Pixel manipulation. Learn to create a tool to encrypt and decrypt images with the support of prodigy infotech and access to knowledge and resources necessary for the endeavor. Together, we embark on a journey to uncover the intricacies of encryption and decryption techniques and operations for pixel manipulation driven by a passion for cyber security, a commitment to ethical practices and an eagerness to learn and explore through self paced.

Requirements:

System or PC

Jupyter notebook or any interpreter

Tkinter and Pillow packages should be installed

Understanding Pixel Manipulation, Image Encryption

Pixel Manipulation:

Pixel manipulation is altering or modifying pixels of an image to achieve a desired result. The process of modifying the RGB values of each pixel in an image. A digital image is nothing more than data—numbers indicating variations of red, green, and blue at a particular location on a grid of pixels. Most of the time, we view these pixels as miniature rectangles sandwiched together on a computer screen. With a little creative thinking and some lower level manipulation of pixels with code, however, we can display that information in a myriad of ways.

We are familiar with the idea of each pixel on the screen having an X and Y position in a two dimensional window. However, the array pixels has only one dimension, storing color values in linear sequence. Images are stored digitally as an array of pixels. A pixel (short for picture element) is the smallest element of an image. Every pixel has a color and the way colors are defined is called the color space.The most used color space is the RGB color space. In this model, every color is defined by three values, one for red, one for green, and one for blue.

Image Encryption :

Image encryption, fundamentally defined as the process of transforming a plain image into a coded form that can only be deciphered by its intended recipient , has gained increasing importance in response to the growing prevalence of image applications and the transmission of images over the internet and open networks. The critical information embedded within these images necessitates secure protection.

In this task we just encrypted the image by manipulating the pixels (altering or modifying the pixel values) of an image. Image encryption works by transforming the pixel values or visual data of an image into a scrambled or encrypted form, making it unintelligible to unauthorized individuals. This transformation is achieved using mathematical algorithms and cryptographic techniques.

The encryption algorithm is applied to each block of the image. The algorithm uses the encryption key to perform mathematical operations on the pixel values within the block. This process alters the pixel values based on the algorithm's specific rules, making it difficult to understand the original content. After encrypting each block, the entire image becomes a scrambled version. The pixel values are rearranged and modified based on the encryption algorithm and the key used. The resulting encrypted image appears as a random arrangement of pixels or patterns.

Common Image Encryption Algorithms

There are several common encryption algorithms used in various applications, domains, and image encryption and decryption project work. Here are some widely-used encryption algorithms:

Advanced Encryption Standard (AES)

AES is a symmetric key encryption algorithm that is adopted by the U.S. government as a standard for encrypting sensitive information. It supports key sizes of 128, 192, and 256 bits and operates on fixed-size blocks.

Data Encryption Standard (DES)

DES is a symmetric key encryption algorithm that has been widely used in the past, but it is now considered relatively weak for modern security requirements. It operates on a 64-bit block size and supports a 56-bit key.

Triple DES (3DES) is an enhancement of DES that applies the algorithm multiple times for increased security.

Blowfish

Blowfish is a symmetric key block cipher that operates on a variable block size and supports key sizes ranging from 32 to 448 bits.

It is known for its flexibility and relatively fast encryption and decryption speed. Blowfish is commonly used in various applications and protocols.

RSA (Rivest-Shamir-Adleman)

RSA is an asymmetric key encryption algorithm named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. Secure data transmission and key exchange are its major applications.

It is based on the mathematical properties of large prime numbers and uses different keys for encryption and decryption. RSA is based on the mathematical properties of large prime numbers and uses different keys to decrypt and encrypt a photo.

Elliptic Curve Cryptography (ECC)

ECC is an asymmetric key encryption algorithm that operates based on the mathematics of elliptic curves. It provides strong security with relatively shorter key lengths compared to other asymmetric algorithms like RSA.

ECC is commonly used in applications where resource constraints, such as bandwidth or computational power, are a concern.

Decryption:

Decryption is the process of taking an encrypted message and restoring it to its original format. Once completed, it can be read again by anyone. It’s like solving a puzzle, where the right combination of codes or passwords is used to reveal the original message.

Decryption Techniques and Algorithms:

Symmetric Decryption

Symmetric decryption is like a secret handshake between two people. It uses the same key for both encrypting and decrypting data, ensuring that only those with the key can access the information. Popular symmetric algorithms include AES, DES, and 3DES, which are used to secure private conversations and other sensitive data.

Asymmetric Decryption

Asymmetric decryption takes a different approach, using two keys instead of one. Imagine a locked mailbox where anyone can drop mail in, but only the owner can unlock and retrieve it. This is how asymmetric decryption works – it uses a public key for encryption and a private key for decryption.

Public-Key Cryptography

Public-key cryptography is the backbone of secure communication and data protection in the digital world. It involves the use of a pair of keys – a public key and a private key – to encrypt and decrypt messages. Anyone can encrypt a message using the recipient’s public key, but only the recipient who possesses the corresponding private key can decrypt it.

Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is a widely-used encryption standard that offers robust security and performance. It supports key lengths of 128-bit, 192-bit, and 256-bit, providing varying levels of security depending on the needs of the user. AES is resistant to all known attacks except brute force, making it a trusted choice for securing sensitive data.

RSA Algorithm

The RSA algorithm is a popular public-key encryption and decryption method that relies on the mathematical complexity of factoring large numbers. With RSA, a user generates a pair of keys – a public key for encryption and a private key for decryption. The user can share the public key with anyone, allowing them to encrypt messages that can only be decrypted by the user with the corresponding private key.

Encryption stratergies:

- Privacy and security
- Regulations
- Secure internet browsing

Operations:

Swapping Pixel values

Pixel swapping in the context of pixel manipulation refers to the process of exchanging the values of pixels in an image. This can be done in various ways, such as swapping the color channels of each pixel (e.g., swapping the red and blue values) or swapping the positions of two or more pixels in the image.

Example of Color Channel Swapping

One common method is to swap the color channels within a single pixel. For example, you can swap the red and blue channels of each pixel to create an encrypted version of the image.

Original Pixel Color Channels:

Red: R

Green: G

Blue: B

Swapped Pixel Color Channels:

Red: B

Green: G

Blue: R

Example of Pixel Position Swapping

Another method is to swap the positions of two pixels in the image. For example, you can take the pixel at position (i, j) and swap it with the pixel at position (j, i).

Original Pixel Positions:

Pixel at position (i, j): P1

Pixel at position (j, i): P2

Swapped Pixel Positions:

Pixel at position (i, j): P2

Pixel at position (j, i): P1

Python Program:

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

Function to load an image :

```
def load_image(path):
    img = Image.open(path)
    return img
```

Function to encrypt the image (swap red and green channels)

```
def encrypt_image(img):
    encrypted_img = img.copy()
    pixels = encrypted_img.load()

    for i in range(encrypted_img.size[0]):
        for j in range(encrypted_img.size[1]):
            r, g, b = pixels[i, j]
            pixels[i, j] = (g, r, b)  # Swap red and green channels

    return encrypted_img
```

Function to decrypt the image (swap green and red channels)

```
def decrypt_image(encrypted_img):
    decrypted_img = encrypted_img.copy()
    pixels = decrypted_img.load()

    for i in range(decrypted_img.size[0]):
        for j in range(decrypted_img.size[1]):
            g, r, b = pixels[i, j]
            pixels[i, j] = (r, g, b)  # Swap green and red channels back

    return decrypted_img
```

Function to save image

```
def save_image(img, path):
    img.save(path)
```

Function to open and load image

```
def open_file():
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg;*.jpeg;*.png")])
    if file_path:
        img = load_image(file_path)
        img.show()
        return img
```

Function to encrypt and display

```
def encrypt_and_show():
    global img, encrypted_img
    img = open_file()
    if img:
        encrypted_img = encrypt_image(img)
        encrypted_img.show()
```

Function to decrypt and display

```
def decrypt_and_show():
    if encrypted_img:
        decrypted_img = decrypt_image(encrypted_img)
        decrypted_img.show()
        return decrypted_img
    else:
        messagebox.showerror("Error", "No encrypted image to decrypt")
```

Function to save encrypted image

```
def save_encrypted_image():
    if encrypted_img:
        file_path = filedialog.asksaveasfilename(defaultextension=".jpg",
filetypes=[("JPEG files", "*.jpg"), ("All files", "*.*")])
        if file_path:
            save_image(encrypted_img, file_path)
    else:
        messagebox.showerror("Error", "No encrypted image to save")
```

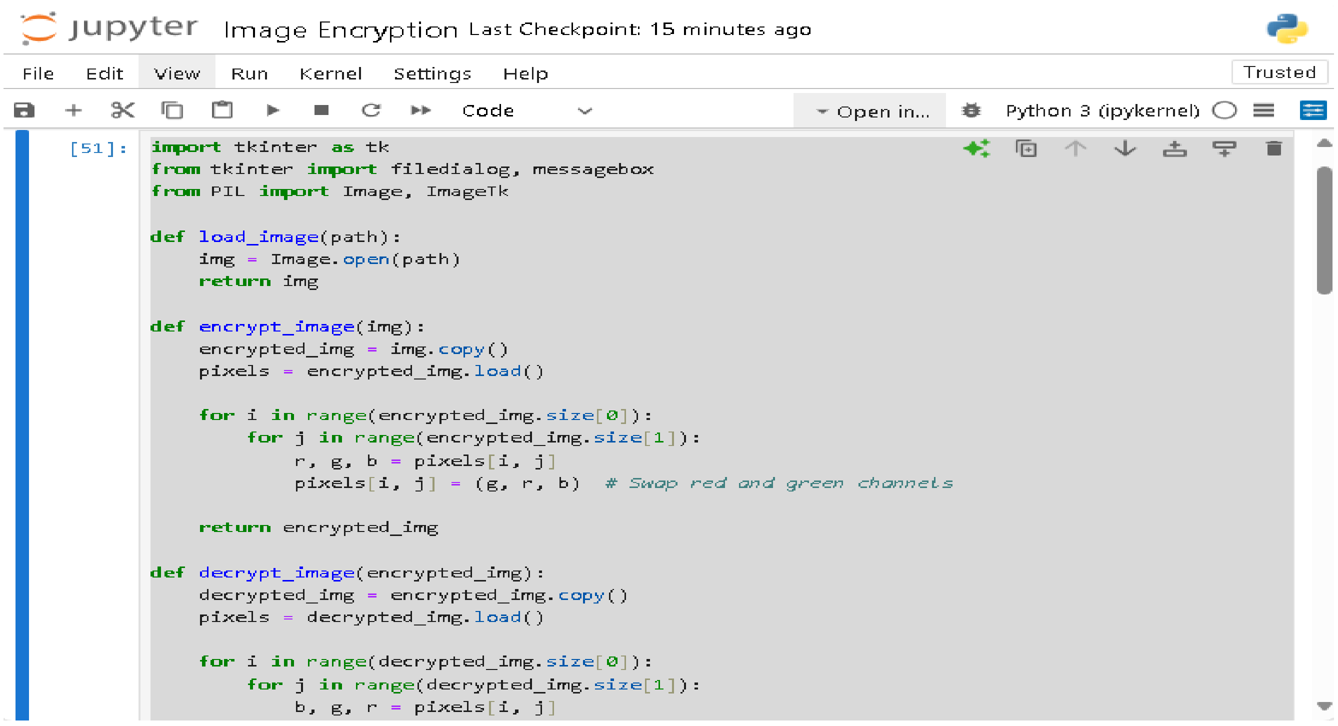
Function to save decrypted image

```
def save_decrypted_image():
    decrypted_img = decrypt_and_show()
    if decrypted_img:
        file_path = filedialog.asksaveasfilename(defaultextension=".jpg", filetypes=[("JPEG files", "*.jpg"), ("All files", "*.*")])
        if file_path:
            save_image(decrypted_img, file_path)
    else:
        messagebox.showerror("Error", "No decrypted image to save")
```

GUI Setup

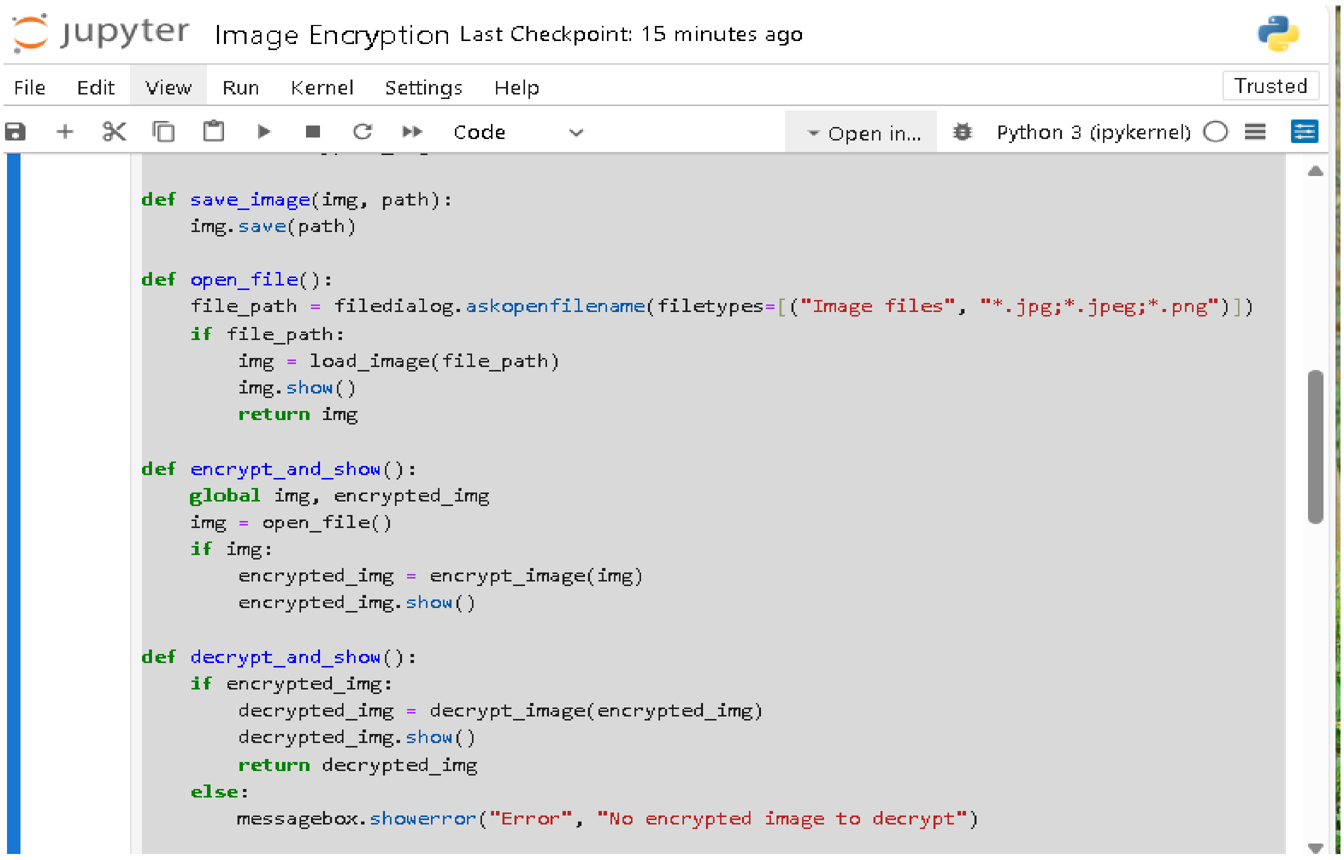
```
root = tk.Tk()
root.title("Image Encryption Tool")


frame = tk.Frame(root)
frame.pack(padx=10, pady=10)


open_button = tk.Button(frame, text="Image to Encrypt ", command=encrypt_and_show)
open_button.pack(pady=5)
save_encrypted_button = tk.Button(frame, text="Save Encrypted Image", command=save_encrypted_image)
save_encrypted_button.pack(pady=5)
decrypt_button = tk.Button(frame, text="Decrypt Image", command=decrypt_and_show)
decrypt_button.pack(pady=5)
save_decrypted_button = tk.Button(frame, text="Save Decrypted Image", command=save_decrypted_image)
save_decrypted_button.pack(pady=5)


root.mainloop()
```

Screen shots:



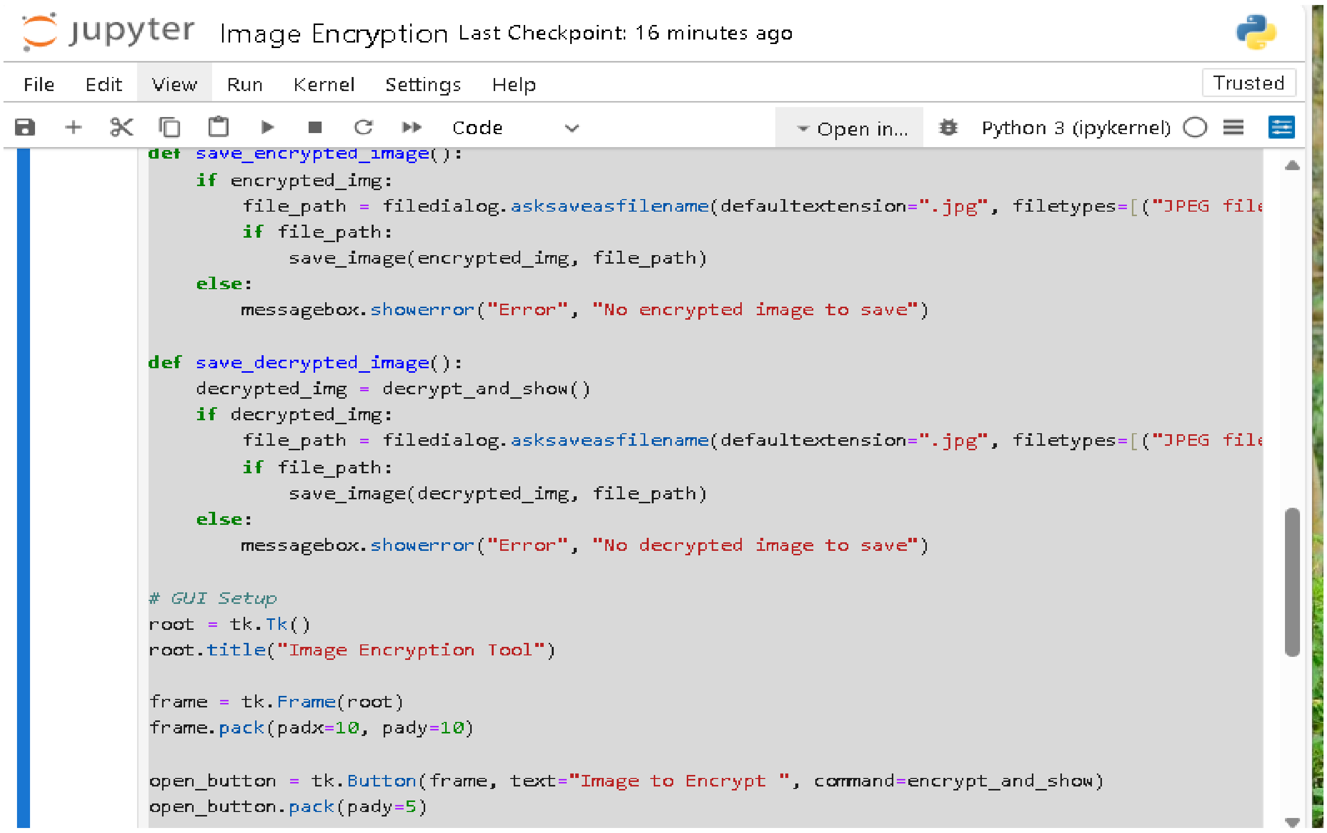
Jupyter Image Encryption Last Checkpoint: 15 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
Open in...
Python 3 (ipykernel)

```
[51]: import tkinter as tk
      from tkinter import filedialog, messagebox
      from PIL import Image, ImageTk

      def load_image(path):
          img = Image.open(path)
          return img

      def encrypt_image(img):
          encrypted_img = img.copy()
          pixels = encrypted_img.load()

          for i in range(encrypted_img.size[0]):
              for j in range(encrypted_img.size[1]):
                  r, g, b = pixels[i, j]
                  pixels[i, j] = (g, r, b)  # Swap red and green channels

          return encrypted_img

      def decrypt_image(encrypted_img):
          decrypted_img = encrypted_img.copy()
          pixels = decrypted_img.load()

          for i in range(decrypted_img.size[0]):
              for j in range(decrypted_img.size[1]):
                  b, g, r = pixels[i, j]
```

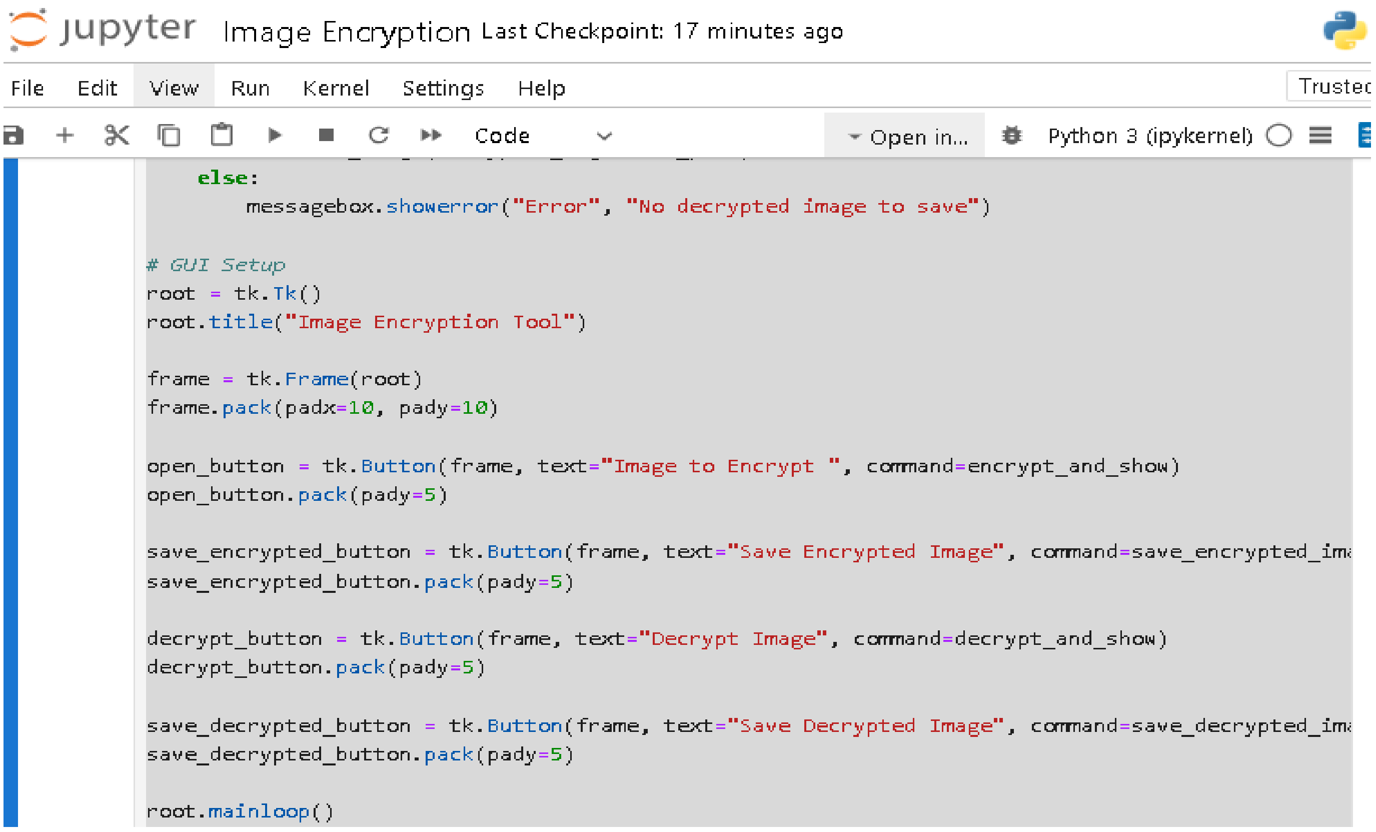


Jupyter Image Encryption Last Checkpoint: 15 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
Open in...
Python 3 (ipykernel)

```
def save_image(img, path):
    img.save(path)

def open_file():
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg;*.jpeg;*.png")])
    if file_path:
        img = load_image(file_path)
        img.show()
        return img

def encrypt_and_show():
    global img, encrypted_img
    img = open_file()
    if img:
        encrypted_img = encrypt_image(img)
        encrypted_img.show()

def decrypt_and_show():
    if encrypted_img:
        decrypted_img = decrypt_image(encrypted_img)
        decrypted_img.show()
        return decrypted_img
    else:
        messagebox.showerror("Error", "No encrypted image to decrypt")
```

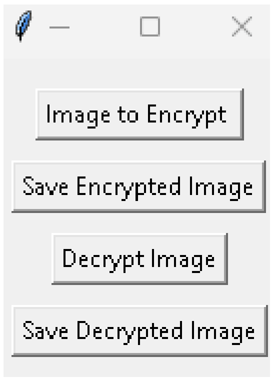

Jupyter
Image Encryption Last Checkpoint: 16 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
Open in...
Python 3 (ipykernel)

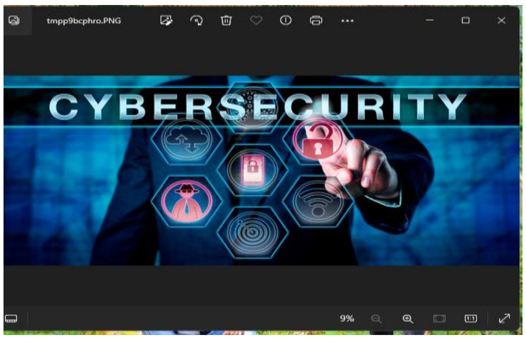
```
def save_encrypted_image():
    if encrypted_img:
        file_path = filedialog.asksaveasfilename(defaultextension=".jpg", filetypes=[("JPEG file
        if file_path:
            save_image(encrypted_img, file_path)
    else:
        messagebox.showerror("Error", "No encrypted image to save")

def save_decrypted_image():
    decrypted_img = decrypt_and_show()
    if decrypted_img:
        file_path = filedialog.asksaveasfilename(defaultextension=".jpg", filetypes=[("JPEG file
        if file_path:
            save_image(decrypted_img, file_path)
    else:
        messagebox.showerror("Error", "No decrypted image to save")

# GUI Setup
root = tk.Tk()
root.title("Image Encryption Tool")

frame = tk.Frame(root)
frame.pack(padx=10, pady=10)

open_button = tk.Button(frame, text="Image to Encrypt ", command=encrypt_and_show)
open_button.pack(pady=5)
```


Jupyter
Image Encryption Last Checkpoint: 17 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
Open in...
Python 3 (ipykernel)

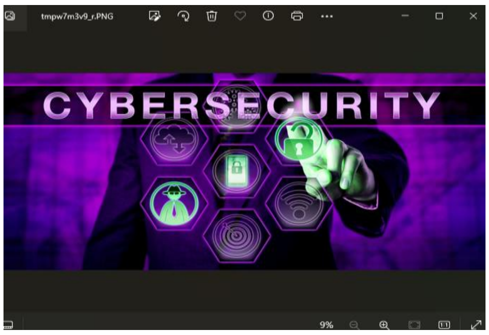
```
    else:
        messagebox.showerror("Error", "No decrypted image to save")

# GUI Setup
root = tk.Tk()
root.title("Image Encryption Tool")

frame = tk.Frame(root)
frame.pack(padx=10, pady=10)

open_button = tk.Button(frame, text="Image to Encrypt ", command=encrypt_and_show)
open_button.pack(pady=5)

save_encrypted_button = tk.Button(frame, text="Save Encrypted Image", command=save_encrypted_ima
save_encrypted_button.pack(pady=5)

decrypt_button = tk.Button(frame, text="Decrypt Image", command=decrypt_and_show)
decrypt_button.pack(pady=5)

save_decrypted_button = tk.Button(frame, text="Save Decrypted Image", command=save_decrypted_ima
save_decrypted_button.pack(pady=5)

root.mainloop()
```

Output:

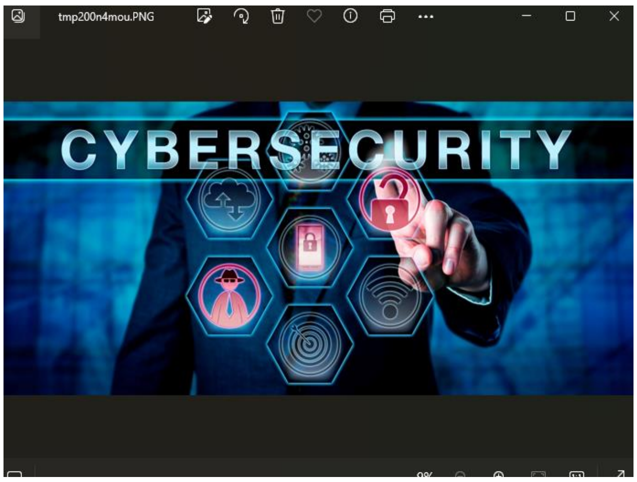




Original image



encrypted



decrypted

Conclusion:

This comprehensive guide has outlined the image encryption tool using pixel manipulation by performing pixel swapping operation by creating python program that can encrypt and decrypt image using pixel values swapping operation by allowing users to input an image to perform encryption and decryption.

Pixel manipulation is altering or modifying pixels of an image to achieve a desired result. The process of modifying the RGB values of each pixel in an image. A digital image is nothing more than data—numbers indicating variations of red, green, and blue at a particular location on a grid of pixels. Image encryption, fundamentally defined as the process of transforming a plain image into a coded form that can only be deciphered by its intended recipient , has gained increasing importance in response to the growing prevalence of image applications and the transmission of images over the internet and open networks.

During this task allotted by Prodigy Infotech. I have gained valuable insights into importance of encryption, decryption techniques and algorithms in securing data.And some new knowledge I gained in depth about the operations involved in this.The support and resources provided by Prodigy infotech have been instrumental in enhancing my understanding and skills in Cyber security.

Reference:

https://cloudinary.com/guides/image-effects/image-manipulation-history-concepts-and-a-complete-guide

https://processing.org/tutorials/pixels

https://www.knowledgehut.com/blog/security/encrypting-images#common-image-encryption-algorithms%C2%A0

https://youtu.be/IloURepHrww?si=JkEEFmqfIS1FJ1QU