Image Steganography - Python and Tkinter

Daniel Koztepe

Introduction:

This project demonstrates how to use image steganography (hiding data in images) in a simple and interactive graphical user interface (GUI) using the Python programming language and its Tkinter toolkit. This app gives users the ability to:

- Select and load an image
- Enter a secret message
- Embed the message into the image
- Extract and read hidden messages from stego-images

What is Steganography?

Image steganography, which is used in the digital age, modifies the Least Significant Bits (LSB) of pixel values to enable hidden messages within image files. Values that indicate the brightness of a particular colour are stored in pixels. Because LSB are so complex, humans are unable to notice this change, but with the correct software, they can find the encoded information.

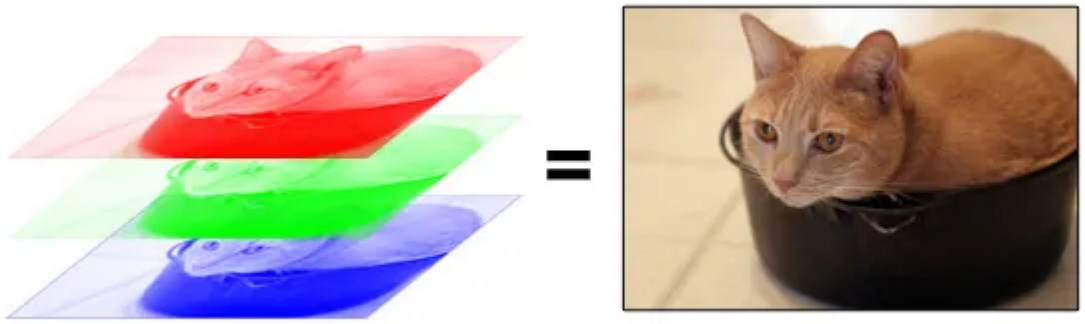
For example, there could be malicious codes which could be accidentally clicked on by employees or even the insider threats of a company transmitting sensitive data to prevent any trace of the leak.

How does LSB work?

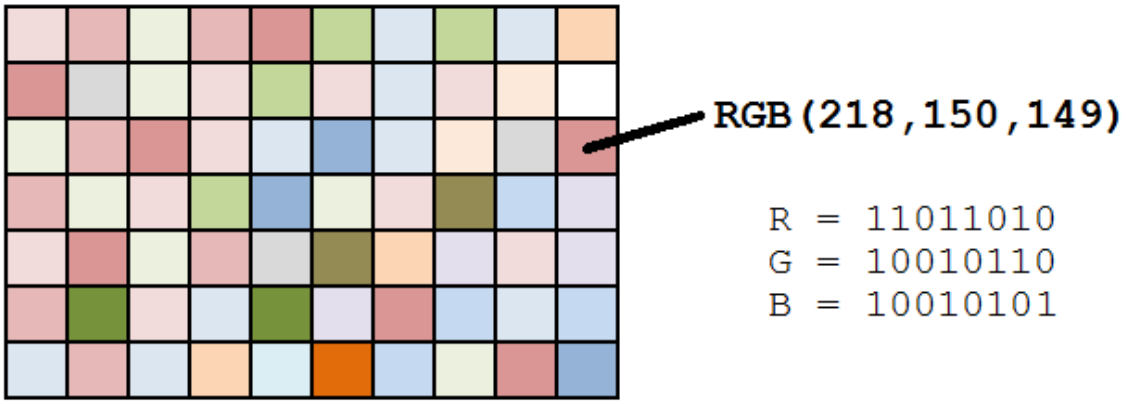
Before you can understand how LSB works, you must understand how a pixel works.

The colour of a pixel is determined by three channels (Red, Green, Blue), or RGB for short.

Computers, televisions, and even cameras use the RGB colour model to represent and display images. As humans are sensitive to these 3 colours, this enables us to create the illusion of realism.



Every channel can have values between 0 and 255, allowing you to generate any colour you want. The channel value increases with the intensity of the colour.

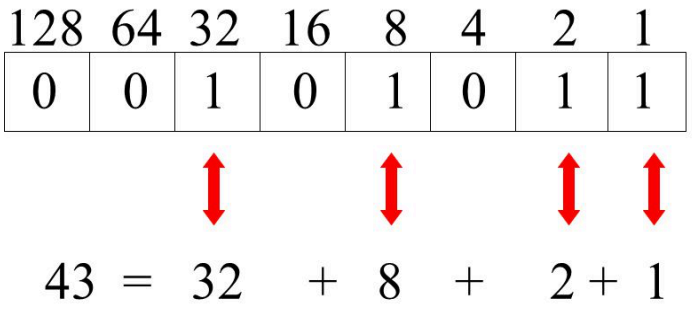


As you can see in the image above, for each value (R,G,B) all represent a binary code.

Binary

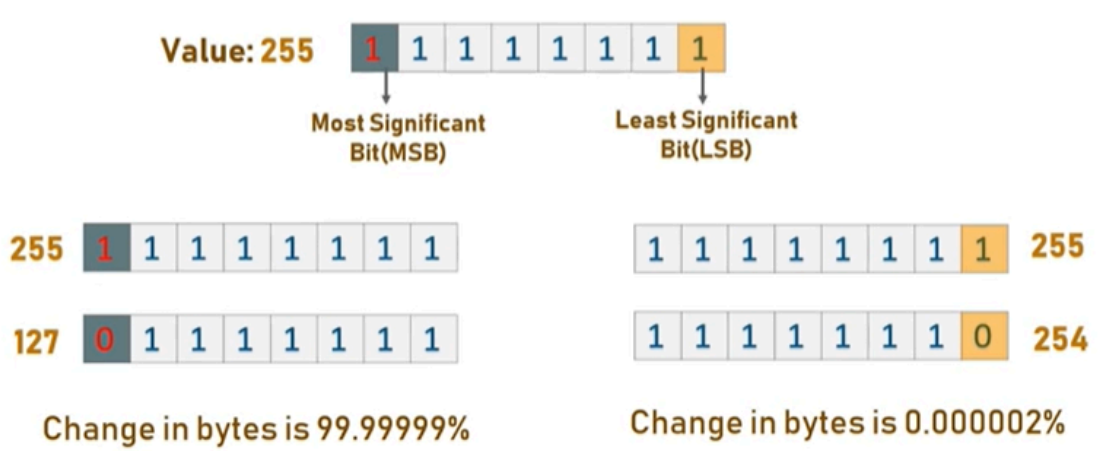
Binary coding, often known as Base-2, is a technique that only uses the integers 0 and 1. It's a language that enables the computer to process commands, text, and images, among other types of data. A sequence of 0s and 1s makes up each piece of data.

However, since each RGB value is 8-bit, meaning that 8 binary values can be stored, we need to know which bits are more crucial than the others when using binary codes.



The leftmost bit (value 128) is the most important one because it greatly affects the final value. The decimal value will change from 255 to 127, for instance, if I change the leftmost bit from 1 to 0 (11111111 to 01111111).

However, the bit on the right is the less significant bit because it has a value of 1. For example, if the leftmost bit changes from 1 to 0 (11111111 to 11111110) the decimal value changes from 255 to 254.



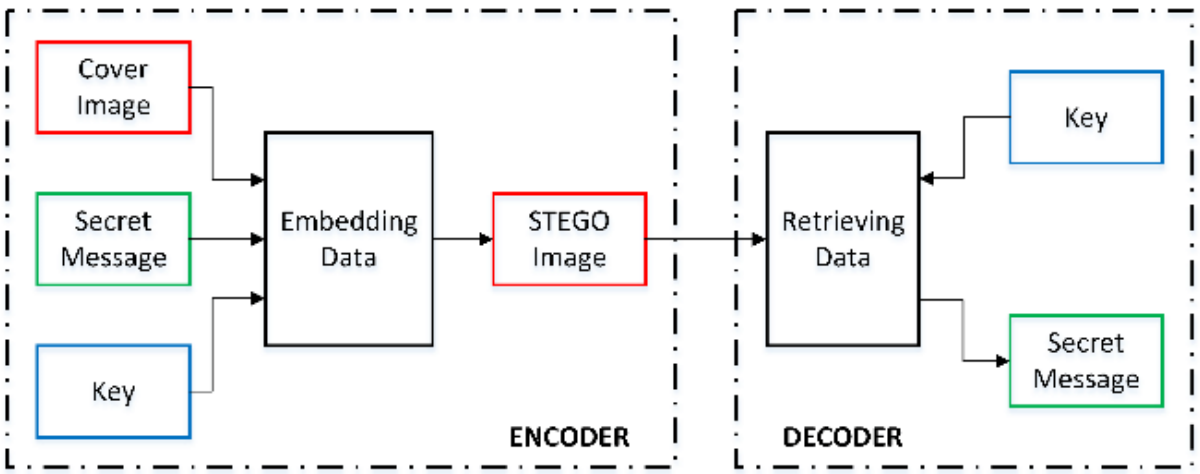
In summary, changing the rightmost bit will allow me to make a minor visual difference in the final image. This is the key behind image steganography that I will be developing.

No Message

Attack at midnight

This is crucial for all employees as placing them in cybersecurity training will teach them how to spot phishing emails that contain malicious files, such as an unusually large file size. This will benefit the company by safeguarding their valuable data and preventing any financial losses.

Algorithm



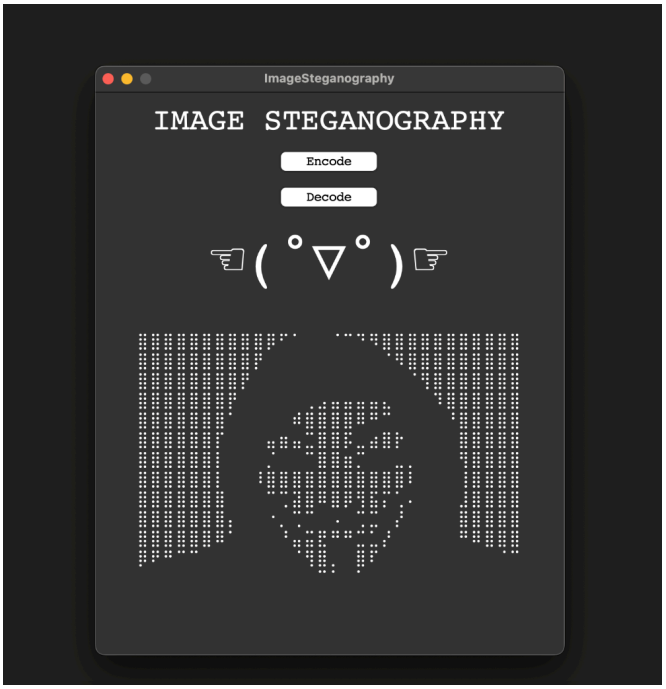
Key Features:

- Clean, user-friendly GUI with clear navigation
- Design element for better user experience
- User-friendly file dialogs for easy navigation
- Implement proper error handling

Libraries Used:

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

Home Page:



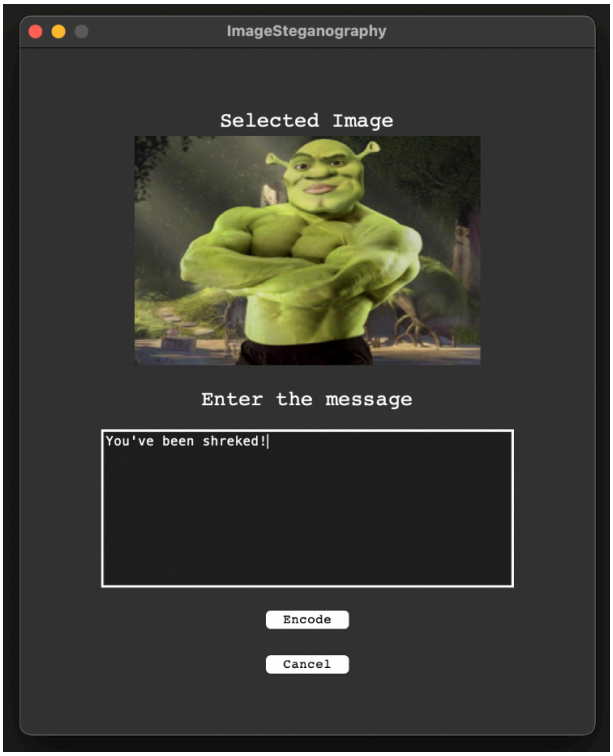

```
class SteganographyApp: # Creating the GUI
    def __init__(self):
        self.root = tk.Tk()
        self.root.title('Image Steganography')
        self.root.geometry('500x600')
        self.create_main_window()

    def create_main_window(self): # Clears the widgets
        for widget in self.root.winfo_children():
            widget.destroy()

        tk.Label(self.root, text='IMAGE STEGANOGRAPHY', # The home page title
                 font=('courier', 20, 'bold')).pack(pady=10)

        tk.Button(self.root, text='Encoding an Image', # Button for encoding
messages in images
                  command=self.encode_message,
                  font=('courier', 12), width=20).pack(pady=5)

        tk.Button(self.root, text='Decoding an Image', # Button for decoding
messages in images
                  command=self.decode_message,
                  font=('courier', 12), width=20).pack(pady=5)

        tk.Label(self.root, text='☜(˚▽˚)☞',  # This is just for the home page
design
                 font=('courier', 40)).pack(pady=10)
```

```
   # This is just for the home page design as well
       art = '''
                                                  '''

       tk.Label(self.root, text=art,
                font=('courier', 20), justify='left').pack(pady=5)
```

Selecting File to Encode*:*

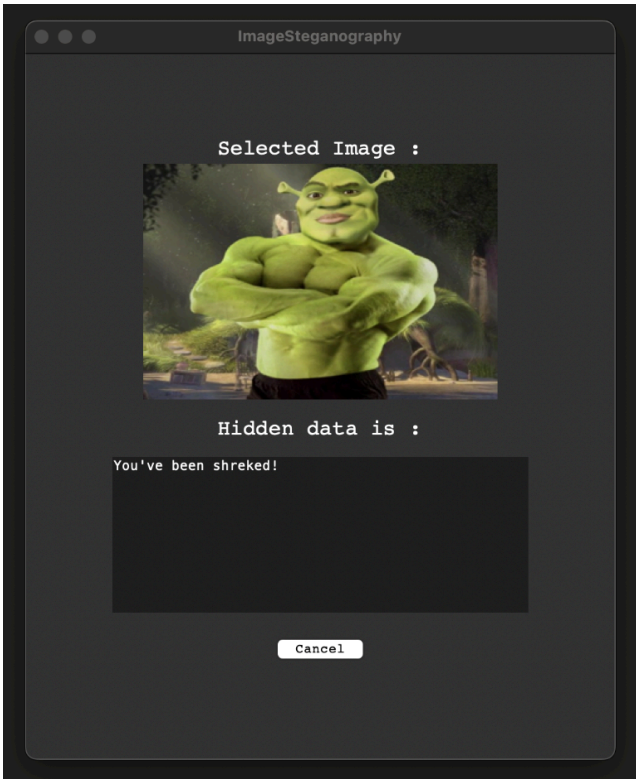

```
   def encode_message(self): # Allowing only image files to be selected when
encoding
       image_path = filedialog.askopenfilename(
           title="Select Image",
           filetypes=[('Image files', '*.png *.jpg *.jpeg')])

       if not image_path: # Exit the path if no image was selected
           return
```

Encoding Page:




```
message = tk.simpledialog.askstring("Message", "Enter message to hide:") # User's
input of their hidden message
        if not message:
            return

        try:
            img = Image.open(image_path) # Encoded message in the image
            encoded_img = self.encode_image(img, message)

            save_path = filedialog.asksaveasfilename(
                defaultextension=".png",
                filetypes=[('PNG files', '*.png')])

            if save_path: # Allows the user to save the encoded file
                encoded_img.save(save_path)
                messagebox.showinfo("Success", "Message hidden successfully!")
        except Exception as e: # Error if encoded incorrectly
            messagebox.showerror("Error", f"Encoding failed: {str(e)}")
```

Decoding Page:




```
   def decode_message(self): # Allowing only image files to be selected when
decoding
        image_path = filedialog.askopenfilename(
            title="Select Image with Hidden Message",
            filetypes=[('Image files', '*.png *.jpg *.jpeg')])

        if not image_path: # Exits if a file wasn't selected
            return

        try:
            img = Image.open(image_path)
            message = self.decode_image(img)

            msg_window = tk.Toplevel(self.root) # The Decode Page
            msg_window.title("Hidden Message")
            msg_window.geometry("400x200")

            tk.Label(msg_window, text="Hidden Message:", # Title of the decode page
                     font=('Arial', 12, 'bold')).pack(pady=10)
            # The text widget
            text_widget = tk.Text(msg_window, height=8, width=50)
            text_widget.insert('1.0', message)
            text_widget.config(state='disabled') # Prevents the user to change the
message (read-only)
            text_widget.pack(pady=10)
```

```
        except Exception as e: # Error message if the decoding fails
            messagebox.showerror("Error", f"Decoding failed: {str(e)}")
```

Algorithm Process of Encoding the Image:

```
    def encode_image(self, img, message):
        # Encoding a message into an image using LSB (Least Significant Bit) steganography
        message += chr(0)
        binary_message = ''.join(format(ord(char), '08b') for char in message) # Converts the characters to a 8 bit binary

        pixels = list(img.getdata()) # Collects the pixel data from image

        if len(binary_message) > len(pixels) * 3: # Checks if hidden message is too long for the image
            raise ValueError("Message too long for this image")

        # Encode message into pixels
        data_index = 0
        new_pixels = [] # Modified pixels

        for pixel in pixels: # Grayscale images coverted to RGB tuple
            if isinstance(pixel, int):
                pixel = (pixel, pixel, pixel)

            new_pixel = list(pixel[:3])  # Take only RGB values

            # Modify RGB values
            for i in range(3):
                if data_index < len(binary_message):
                    # Set LSB to message bit
                    new_pixel[i] = (new_pixel[i] & 0xFE) | int(binary_message[data_index])
                    data_index += 1

            new_pixels.append(tuple(new_pixel)) # Modified pixel to new pixel

        # Create new image with the modified pixels
        new_img = Image.new(img.mode, img.size)
        new_img.putdata(new_pixels)
        return new_img
```

Algorithm Process of Decoding the Image:

```
    def decode_image(self, img):
```

```
        pixels = list(img.getdata()) # Collects the pixel data from the original
image
        binary_message = "" # Allows me to store the extracted binary data

        # Extract LSBs of each colour channel
        for pixel in pixels:
            if isinstance(pixel, int):
                pixel = (pixel, pixel, pixel)

            # Only extracting the LSB
            for i in range(3):
                binary_message += str(pixel[i] & 1)

        # Convert binary to text
        message = ""
        for i in range(0, len(binary_message), 8): # Process 8 bits at a time
            byte = binary_message[i:i+8]
            if len(byte) == 8: # Confirms if the bytes are accurate
                char = chr(int(byte, 2)) # Converts binary to character
                if char == chr(0):
                    break
                message += char
        return message
```

To Run the Application:

```
    def run(self):
        self.root.mainloop() #starts the loop
# Runs the application
if __name__ == "__main__":
    app = SteganographyApp()
    app.run()
```

Conclusion:

Using Python and Tkinter, this project has effectively demonstrated how to build image steganography. A simple and easy-to-use graphical user interface has been made accessible for selecting between options for encoding and decoding hidden messages in images. The user of Tkinter has allowed me to provide accessibility for the users to enable a seamless experience with their interactions such as file selections and message input. Along with the factors, the Least Significant Bit (LSB) has ensured minimal visual distortion with the RGB channels to generate the integrity of the hidden message.

By creating a teaching tool that simulates real-world cyberattacks, I am able to educate myself and others. While I intend to further improve encryption for messages that are encoded in the future, the project has accessibility for technical and non-technical users.

Project File: https://github.com/Danielkoz/Python-Projects

References

Steganography
Image Link
https://www.echomark.com/post/how-steganography-can-help-prevent-internal-information-theft
650 × 233
https://medium.com/advanced-deep-learning/decoding-image-representation-understanding-the-structure-of-rgb-images-6a211eb8800d
960 × 720
https://www.britannica.com/technology/binary-code
826 × 315
653 × 256
1,600 × 675
https://www.kaspersky.com/resource-center/definitions/what-is-steganography