IMAGE STEGANOGRAPHY

Mini Project Report submitted in partial fulfillment of the requirements for the award of the Degree of B.E in Computer Science and Engineering.

By

Janapati Himaja	160620733018
Kadarla Sharvani	160620733022
Pulakurthi Anaghaa Reddy	160207333043

Under the Guidance of

Dr Y.V.S.S Pragathi
Professor
Department of Computer Science & Engineering

Department of Computer Science and Engineering

Stanley College of Engineering & Technology for Women (Autonomous)

Chapel Road, Abids, Hyderabad – 500001
(Affiliated to Osmania University, Hyderabad, Approved by AICTE,
Accredited by NBA & NAAC with A Grade)
(2022-2023)

Stanley College of Engineering & Technology for Women

(Autonomous)

Chapel Road, Abids, Hyderabad – 500001

(Affiliated to Osmania University, Hyderabad, Approved by AICTE, Accredited by NBA & NAAC with A Grade)

CERTIFICATE

This is to certify that the mini project report entitled "**IMAGE STEGANOGRAPHY**"

being submitted by

Janapati Himaja	160620733018
Kadarla Sharvani	160620733022
Pulakurthi Anaghaa Reddy	160620733043

in partial fulfillment for the award of the Degree of Bachelor of Engineering in Computer Science & Engineering to the Osmania University, Hyderabad, is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Guide

Dr Y.V.S.S Pragathi

Professor, Dept of CSE

Head of the Department

Dr Y.V.S.S Pragathi

HoD, Dept of CSE

Project Coordinator

Dr D. Radhika

Associate Prof, Dept of CSE

External Examiner

DECLARATION

We hereby declare that the mini project work entitled "***Image Steganography",*** submitted to the Osmania University, Hyderabad is a record of original work done by us. This project work is submitted in partial fulfillment of the requirements for the award of the degree of B.E. in Computer Science and Engineering.

Janapati Himaja	160620733018
Kadarla Sharvani	160620733022
Pulakurthi Anaghaa Reddy	160620733043

ACKNOWLEDGEMENT

First of all, we are grateful to **The Almighty God** for establishing us to complete this Mini Project. We pay our respects and love to our parents, family members, and friends for their love and encouragement throughout our careers.

We wish to express our sincere thanks to Sri. Kodali Krishna Rao, Correspondent and Secretary Stanley College of Engineering & Technology for Women, for providing us with all the necessary facilities.

We place on record our sincere gratitude to Prof. Satya Prasad Lanka, Principal, for his constant encouragement.

We sincerely express our sincere thanks to our Head of the Department, Prof Y.V.S.S. Pragathi, for encouraging and allowing us to present the Mini Project on the topic of Image Steganography at our department premises for the partial fulfillment of the requirements leading to the award of the B.E. degree.

It is our privilege to express sincere regards to our project guide Dr Y.V.S.S. Pragathi, for the valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project.

We take this opportunity to thank all our faculty, who have directly or indirectly helped our project. Last but not least, we express our thanks to our friends for their cooperation and support.

ABSTRACT

Steganography is the practice of hiding a secret message which can be text or non-text files in non-text files like images. It preserves the integrity of a secret file so that apart from the desired receiver, no one else can know its presence. The objective of steganography is to maintain *secret communication* between individuals. In this Image Steganography, we aim to conceal the data into an image file by encoding and revealing it by decoding.

This GUI-based Python project uses *Tkinter, PIL* modules and the *Least Significant Bit (LSB) algorithm*. The least significant bit insertion is a standard and straightforward method to embed data in an image file. Tkinter uses the Python Pillow(PIL) package for image processing capabilities. Here we create a window with 2 buttons: *Encode* and *Decode*. It takes the image in which the secret message is to be encoded and the secret message as input. We are provided with an output of an *encrypted image*. Using the decode option, the desired receiver can *decode* it.

Keywords: *Steganography, secret communication, Tkinter, Least Significant Bit (LSB)*

algorithm, Encode, Decode.

Tables of Contents

List of Figures

CHAPTER 1: INTRODUCTION

1.1 About the Project

“A picture is worth a thousand words".Yes, we can have words in a picture. Today any form of data such as text, image, audio or video can be digitized, and it is possible to insert secret binary information into the data during the digitisation process. One such technique is Steganography. The word steganography originates in Greek and means “covered writing”.Steganography is concealing messages or information within other nonsecret text or data. Steganography works have been done on different transmission media like images, video, text, or audio.

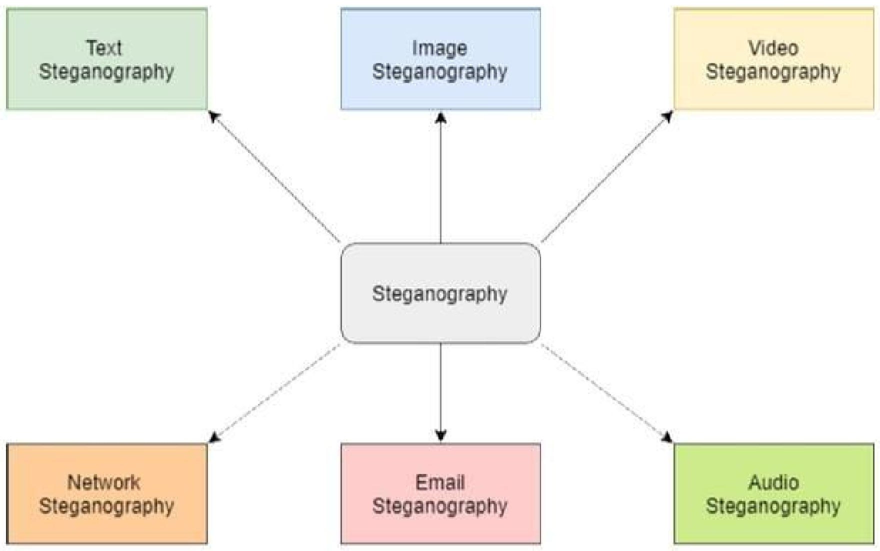


Image Steganography, as the name implies, is the process of concealing data within an image file. The image chosen for this purpose is referred to as the cover image, and the image obtained after steganography is referred to as the stego image. There are lots of ways to hide information inside an image. Common approaches include:

- Least Significant Bit Insertion
- Masking and Filtering
- Redundant Pattern Encoding
- Encrypt and Scatter
- Coding and Cosine Transformation

In this project, we will use the Least Significant Bit Insertion. The least significant bit insertion is the standard technique used in Steganography. In the LSB method, an image is used. A picture is more than strings and a string of bytes. Each byte in an image represents different colors. The last few bits in a color byte are less significant than the first few bits. Therefore only two bits differ in the last few bits representing a color indistinguishable from human eyes. In this method, the least significant bits of a cover image are altered such that we can embed information.

1.2 Objectives of the Project

The objective of this project on steganography specifically focuses on text hiding in images and image hiding in images using the LSB algorithm, is to implement a steganographic technique that allows for the covert embedding and extraction of textual information within images, as well as the hiding of one image within another image using the LSB Substitution algorithm. The main goal is to explore and demonstrate the feasibility of these techniques in steganography and assess their effectiveness in terms of security, capacity, and visual quality.

The code aims to achieve the following objectives:

1. Text Hiding: Users can enter a text message and select a cover image. The code should embed the text message within the cover image using the LSB algorithm, ensuring minimal visual distortion. The resulting stego image is saved for further use.

2. Image Hiding: Enable users to select a cover and secret images. The code should embed the secret image within the cover image using the LSB algorithm, maintaining the visual quality of the cover image as much as possible. The resulting stego image is saved for future reference.

3. Graphical User Interface (GUI): A user-friendly Tkinter library interface allows users to browse and select the cover and secret images conveniently. The GUI should include buttons for text hiding and image hiding, as well as entry fields for entering the text message.

4. Flexibility and Usability: Ensure the code is flexible enough to handle different image formats (e.g., PNG, JPEG) and allow users to hide text or images of varying sizes. The code should handle potential errors, such as messages or photos exceeding the embedding capacity, and provide informative error messages.

5. Visual Integrity: Minimizing any noticeable visual changes or distortions in the resulting stego image. The code should prioritize maintaining the quality and visual fidelity of the cover image while hiding the secret information.

Overall, this code aims to provide a functional and user-friendly solution for steganography, allowing users to hide text or images within other images using the LSB algorithm while providing a graphical interface for ease of use.

1.3 Scope of the Project

Our project can hide unlimited data. It can handle any text(alphabets, numerics, special characters). It provides a better way to hide large images, except for both the cover and hidden images being the same size. If they are of different size, it is resized, which may lead to loss of information.
This project is designed for encoding and decoding data, which can be performed on a wide range of devices with the executable environment alone. It restricts the user from performing encoding or decoding with a different algorithm. The chance of storing the results into users folders is an advantage.

1.4 Advantages

1. User-Friendly GUI:
The mini project boasts a user-friendly Graphical User Interface (GUI) built with tkinter, enabling easy interaction for users. They can effortlessly select cover images, secret texts, or images and perform steganography operations without technical expertise.

2. Versatility in Data Hiding:
Users can conceal various types of information by supporting text and image-hiding techniques, offering flexibility for different use cases.

3. Secure Data Hiding:
The project enhances data security by enabling users to hide sensitive information within cover images, providing confidentiality during communication.

4. Encoding and Decoding Functionality:
Users can encode secret data into cover images and later decode the hidden content, ensuring seamless retrieval of the original information.

5. Efficient Image Processing:
The integration of OpenCV, a powerful computer vision library, ensures efficient image processing and manipulation, enhancing performance.

6. Modular Structure for Future Enhancements:
The mini project's modular structure allows for easy integration of additional steganography methods or advanced image processing techniques, providing room for future improvements.

7. Educational Tool:
Serving as a practical learning opportunity, the project imparts valuable insights into steganography concepts, binary representation, GUI development, and image processing.

8. Practical Applications:
The project's data-hiding capabilities offer practical applications in secure communication and confidential data sharing.

1.5 Disadvantages

1. Limited Capacity for Data:
Image Steganography has limited capacity to hide large amounts of data within images, as it relies on modifying pixel values, which can lead to reduced image quality.

2. Susceptibility to Attacks:
Steganographic images may be vulnerable to detection or attacks by experienced adversaries using specialized techniques or algorithms.

3. Data Loss and Corruption:
Embedding secret data into cover images may lead to the loss or corruption of hidden information if the steganographic image undergoes compression or editing.

4. Incompatibility with Certain Formats:
Not all image formats are suitable for steganography, potentially limiting the choice of cover images.

5. Complexity of Decoding Process:
The process of decoding hidden information from steganographic images may require specialized knowledge and tools.

1.6 Applications

1. Secure Communication:
Image Steganography finds applications in secure communication by concealing sensitive messages within innocent-looking images, safeguarding data from unauthorized access.

2. Data Privacy and Confidentiality:
The mini-project can be utilized to protect data privacy and confidentiality, ensuring that confidential information remains hidden from prying eyes.

3. Digital Watermarking:
Image Steganography can be applied to embed digital watermarks into images, helping to verify the authenticity and ownership of digital media.

4. Covert Communication:
Steganography enables covert communication, where secret messages can be exchanged in plain sight without attracting suspicion.

5. Copyright Protection:
The project's steganographic techniques can be used for copyright protection, embedding copyright information into digital media to prevent unauthorized use.

6. Forensics and Investigations:
Image Steganography plays a role in digital forensics and investigations, allowing experts to analyze hidden information in suspect images.

1.7 Hardware and Software requirements

Hardware Requirements:

- A PC or laptop that can satisfy software requirements

Software Requirements:

- Python IDLE(Version 3.7 or more)
- Python libraries and modules (like OpenCV, NumPy, Tkinter, Random and Pillow)

CHAPTER 2: PROPOSED ARCHITECTURE

2.1. Overview

The proposed architecture for the Image Steganography mini-project involves building a Python application that enables users to hide secret text or images within cover images and decode the hidden information. The architecture consists of three main components: GUI (Graphical User Interface), Encoding, and Decoding.

2.2. GUI Component

The GUI component provides users with an interactive and user-friendly interface. It is implemented using the Tkinter library, allowing easy integration with the Python application. The main window of the GUI presents the following functionalities:

a) Text in Image Steganography Button:
- Upon clicking this button, users can select a cover image and a secret text file to embed the text into the cover image.

b) Image in Image Steganography Button:
- When users click this button, they can choose a cover image and a secret image to hide the image within the cover image.

c) Decode Text Button:
- This button allows users to select a steganographic image containing hidden text and decode the text from the image.

d) Decode Image Button:
- Users can choose a steganographic image containing a hidden image and decode the image from the cover image.

The GUI provides file dialogs for image and text file selection, status messages, and pop-up alerts to keep users informed about the application's progress and outcomes.

2.3. Encoding Component

The Encoding component handles the process of embedding the secret information into the cover images. It includes two sub-components for text and image hiding:

a) Text in Image Steganography:
- This sub-component takes the selected cover image and secret text as inputs.
- The secret text is converted to binary form, and an End of Text marker is appended.
- The binary bits of the secret text are embedded into the least significant bits of the cover image's pixel values until the entire message is hidden.
- If the cover image's capacity is insufficient, a warning message is displayed to the user.

b) Image in Image Steganography:
- This sub-component takes the selected cover image and secret image as inputs.
- The 4 most significant bits (MSBs) of each pixel in the secret image are merged with the 4

MSBs of each pixel in the cover image.
- The result is a steganographic image where the secret image is concealed within the cover image.

2.4. Decoding Component

The Decoding component is responsible for extracting the hidden information from the steganographic images. It includes two sub-components for text and image decoding:

a) Decode Hidden Text:
- The sub-component takes a steganographic image as input.
- It analyzes the image's pixel values and extracts the least significant bits of each pixel to reconstruct the binary representation of the hidden text.
- The binary representation is searched for the End of Text marker to determine the end of the hidden text.
- The hidden text is extracted and displayed to the user.

b) Decode Hidden Image:
- This sub-component takes a steganographic image as input.
- It processes the pixel values to retrieve the 4 MSBs from each pixel, which constitute the binary representation of the hidden image.
- The binary representation is used to reconstruct the hidden image, which is then displayed to the user.

CHAPTER 3: IMPLEMENTATION

3.1 Algorithm

The LSB (Least Significant Bit) algorithm is a widely used and simple steganographic technique for hiding information within a digital signal, such as an image, audio, or video file, without significantly affecting its perceptual quality. The technique is based on manipulating each byte or pixel's least significant bit (LSB) in the signal to embed the hidden information.

Here's a general overview of how the LSB algorithm works:

1. Binary Representation: The data you want to hide (e.g., text, image, or other files) is converted into binary form. A sequence of bits represents each character or pixel value.

2. Cover Signal: The "cover signal" is the original digital signal (e.g., an image) you want to use to hide the data. It can be any digital file where you can modify individual bits without causing noticeable changes to the human eye or ear.

3. LSB Replacement: The least significant bit is replaced with the first bit(i.e. the least significant bit) of the hidden data for each byte or pixel in the cover signal. The changes are typically imperceptible since the LSB has the most minor effect on the overall value.

4. Embedding Data: Continue replacing the LSB of each byte or pixel in the cover signal with the bits of the hidden data until all the data has been embedded.

5. Save the Modified Signal: The modified signal (e.g., the modified image) is saved once all the data has been hidden.

To extract the hidden data from the modified signal, you perform the reverse process:

1. Access the LSB: Extract the least significant bit for each byte or pixel in the modified signal.

2. Reconstruct Hidden Data: Combine the extracted bits to reconstruct the hidden data.

It's important to note that the LSB algorithm is an essential and easy-to-implement technique. However, it is not highly secure against more advanced steganalysis techniques or attacks. More sophisticated steganographic methods and encryption should be considered for more robust security. Additionally, depending on the nature of the cover signal and the amount of data to be hidden, the hidden information may be vulnerable to loss if the cover signal undergoes heavy compression or editing.

3.2 Code Implementation

3.2.1. Installation of modules/packages: We need to install modules like OpenCV, NumPy, Tkinter, Random and Pillow beforehand.

(i) Import the cv2 package, which is the name of the OpenCV module: OpenCV is a Python library that allows you to perform image processing and computer vision tasks. It provides many features, including object detection, face recognition, and tracking.

Command: pip install opencv-python

(ii) Import the Numpy package: NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds robust data structures to Python that guarantee efficient calculations with arrays and matrices, and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

Command: pip install numpy

(iii) Import Tkinter: Tkinter is the built-in Python module to create GUI applications. It is one of the most commonly used modules for building GUI applications in Python, as it is simple and easy to work with.

Command: pip install tkinter

(iv) From tkinter importing filedialog, messagebox modules:

a) filedialog is one of the tkinter modules that provides classes and library functions to create file/directory selection windows. You can use filedialog, where you must ask the user to browse a file or a directory from the system.
b) The tkMessageBox module is used to display message boxes in your applications. This module provides several functions that you can use to display an appropriate message. Some of these functions are showinfo, showwarning, showerror, askquestion, askokcancel, askyesno, and askretryignore.

(v) Import Random: The Python Random module is a built-in module for generating random integers in Python. These are fake random numbers that do not possess true randomness. We can therefore use this module to generate random numbers, display a random item for a list or string, and so on. Random module in Python requires no installation as it comes built-in with Python and can be used directly by importing the required functions or the whole module into the working file.

(vi) PIL: The Python Imaging Library best suits image archival and batch-processing applications. Python pillow package can create thumbnails, convert from one format to another, print images, etc.

Command: pip install Pillow

a) ImageTK: The ImageTk module supports creating and modifying Tkinter BitmapImage and PhotoImage objects from PIL images.
b) Image: The Image module provides a class with the same name to represent a PIL image. The module also provides several factory functions, including loading images

from files and creating new images.

```
import cv2
import numpy as np
import tkinter as tk
from tkinter import filedialog, messagebox
import random
from PIL import ImageTk, Image
```

3.2.2. Function for encoding the secret text into the cover image: ‘encode_text’ aims to hide a personal text within an image using steganography.

(i) Function ‘encode_text’ takes two parameters: `cover_img` (the image to hide the text in) and `secret_text` (the text to be encoded).

(ii) The ‘secret_text’ is converted into binary format. Each character in the ‘secret_text’ is first converted to its ASCII value using ‘ord(c)’, then that ASCII value is converted to an 8-bit binary representation using ‘format(..., '08b')’. The resulting binary values are concatenated to form a single binary string, `secret_text_binary`.

(iii) An "End of Text" marker is appended to the ‘secret_text_binary’ string. This marker is a binary sequence (‘1111111111111110’) used to indicate the end of the hidden text during the decoding process.

(iv) The variable, ‘text_index’, keeps track of the current position within the ‘secret_text_binary’ string.

(v) The nested loops iterate over each pixel in the cover image. The ‘cover_img’ is assumed to be a three-dimensional array representing the image, with dimensions ‘[height, width, channels]’.

(vi) The ‘if’ condition checks if there is remaining text to be encoded. If ‘text_index’ is less than the length of ‘secret_text_binary’, more text must be encoded.

(vii) The following few lines modify the least significant bit (LSB) of each colour channel of the current pixel. The LSB of the pixel value is replaced with the corresponding bit from ‘secret_text_binary’ at the ‘text_index’ position. This process is done by converting the pixel value to an 8-bit binary representation (‘binary_value’), replacing the last bit with the desired bit from ‘secret_text_binary’, and then converting the modified binary value back to an integer. The modified pixel value is assigned back to ‘cover_img’.

(viii) After modifying a pixel, the 'text_index' is incremented to move to the next bit in 'secret_text_binary'.

(ix) If there is no more text to encode ('text_index' is greater than or equal to the length of `secret_text_binary`), the loop breaks and stops modifying the image pixels.

(x) The last 'if' condition is checked after each channel iteration ('k' loop) to break out of the colour channel loop if no more text is encoded.

```python
# Encode the secret text into the cover image
def encode_text(cover_img, secret_text):
    secret_text_binary = ''.join(format(ord(c), '08b') for c in secret_text)
    secret_text_binary += '1111111111111110'  # End of Text marker
    text_index = 0
    for i in range(cover_img.shape[0]):
        for j in range(cover_img.shape[1]):
            for k in range(cover_img.shape[2]):
                if text_index < len(secret_text_binary):
                    # Modify the last bit of the cover image pixel value
                    binary_value = format(cover_img[i][j][k], '08b')
                    modified_binary_value = binary_value[:-1] + secret_text_binary[text_index]
                    cover_img[i][j][k] = int(modified_binary_value, 2)
                    text_index += 1
                else:
                    break
            if text_index >= len(secret_text_binary):
                break
    if text_index < len(secret_text_binary):
        messagebox.showwarning("Steganography", "Insufficient cover image capacity to hide the entire message")
```

3.2.3. Function for resizing secret image: The 'resize_images' function is responsible for resizing the secret image to match the dimensions of the cover image. Here's how it works:

(i) The function takes two parameters: 'cover_img' and 'secret_img'

(ii) It retrieves the dimensions of the cover image using the 'shape' attribute. The shape attribute returns a tuple with three values: the image's height, width, and number of channels. In this case, we use `_` to disregard the number of channels because we are only interested in the height and width.

(iii) The `cv2.resize` function is then used to resize the secret image.

(iv) Finally, the function returns both the cover image and the resized secret image as a tuple: `(cover_img, secret_img)`.

```
def resize_images(cover_img, secret_img):
    # Get the dimensions of the cover image
    cover_height, cover_width, _ = cover_img.shape
    # Resize the secret image to match the dimensions of the cover image
    secret_img = cv2.resize(secret_img, (cover_width, cover_height))
    return cover_img, secret_img
```

3.2.4. Function for encoding the secret image into the cover image: 'encode_image' performs image steganography by hiding one image within another.

(i) Function named 'encode_image' takes two parameters: 'cover_img' (the image in which the secret image will be hidden) and 'secret_img' (the image to be encoded).

(ii) The three nested for loops iterate over each pixel of the 'secret_img'. The 'secret_img' is assumed to be a three-dimensional array representing the image, with dimensions [height, width, channels]. The loop variables 'i' and 'j' represent the coordinates of each pixel, and `'l' represents the colour channel (red, green, or blue).

(iii) The following two lines convert the 'cover_img' and 'secret_img' pixel values into 8-bit binary representations. The 'format(..., '08b')' function converts the integer pixel values to binary strings with leading zeros to ensure they are 8 bits long.

(iv) The four most significant bits (MSBs) of 'v1' and 'v2' are combined to create a new binary string 'v3'. The MSBs from 'v1' are obtained using slicing 'v1[:4]', and the MSBs from 'v2' are obtained using 'v2[:4]'. The resulting binary string 'v3' contains the combined MSBs of the two images.

(v) The combined binary value 'v3' is converted back to an integer using 'int(v3, 2)', and the resulting value is assigned to the corresponding pixel in the 'cover_img'. This operation replaces the four least significant bits (LSBs) of the 'cover_img' with the MSBs of the 'secret_img'.

(vi) After the execution of the nested loops, the 'cover_img' will have been modified to hide the information from the 'secret_img' within it. The resulting image will appear similar to the original 'cover_img' but with subtle modifications in the LSBs of the pixel values to accommodate the information from the 'secret_img'.

```
# Encode the secret image into the cover image
def encode_image(cover_img, secret_img):
    # Check if the dimensions of cover_img and secret_img are the same
    if cover_img.shape != secret_img.shape:
        messagebox.showerror("Error", "Cover image and secret image dimensions do not match")
        return
    for i in range(secret_img.shape[0]):
        for j in range(secret_img.shape[1]):
            for l in range(3):
                if i >= cover_img.shape[0] or j >= cover_img.shape[1]:
                    # Skip if the indices are out of bounds for cover_img
                    continue
                # v1 and v2 are 8-bit pixel values of img1 and img2 respectively
                v1 = format(cover_img[i][j][l], '08b')
                v2 = format(secret_img[i][j][l], '08b')
                # Taking 4 MSBs of each image
                v3 = v1[:4] + v2[:4]
                cover_img[i][j][l] = int(v3, 2)
```

3.2.5. Function for decoding the secret text from the cover image: The function named `decode_text` aims to extract a hidden text message from a steganographic image.

(i) First line of code opens a file dialog window to select an image file (in PNG format). The chosen file path is stored in the ‘filepath’ variable.

(ii) If a filepath is selected (i.e., the ‘filepath’ variable is not empty), the selected image is loaded using the `cv2.imread` function from the OpenCV library. The loaded image is stored in the ‘img’ variable.

(iii) The nested loops iterate over each pixel of the ‘img’. Similar to the previous code snippets, `img` is assumed to be a three-dimensional array representing the image, with dimensions [height, width, channels]. For each pixel, the least significant bit (LSB) of each colour channel is extracted by converting the pixel value to an 8-bit binary representation (‘binary_value’) and concatenating the last bit (‘binary_value[-1]’) to the ‘binary_message’ string.

(iv) The ‘end_index’ variable is assigned the index of the "End of Text" marker (‘1111111111111110’) within the ‘binary_message’ string. The ‘find’ function is used to locate the first occurrence of the marker.

(v) If the "End of Text" marker is found (‘end_index’ is not -1), the binary message before the marker is extracted and stored in the ‘secret_text_binary’ variable. The ‘secret_text_binary’ is then converted back to its original text form by splitting it into 8-bit segments (‘byte’) and converting each element to its corresponding ASCII character using ‘chr(int(byte, 2))’. The resulting text is stored in the ‘secret_text’ variable. Finally, a message box is shown with the decoded secret message. If the "End of Text" marker is not found, a warning message box is shown, indicating that no hidden message was found in the image. In summary, the ‘decode_text’ function prompts the user to select a steganographic image, extracts the hidden binary message by considering the LSBs of the image pixels, decodes the binary message into a text message, and displays the decoded message using a message box.

```
# Decode the secret message from the steganographic image-text
def decode_text():
    filepath = filedialog.askopenfilename(filetypes=[("Image Files", "*.png")])
    if filepath:
        img = cv2.imread(filepath)
        binary_message = ""
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                for k in range(img.shape[2]):
                    binary_value = format(img[i][j][k], '08b')
                    binary_message += binary_value[-1]
        end_index = binary_message.find('1111111111111110')
        if end_index != -1:
            secret_text_binary = binary_message[:end_index]
            secret_text = ""
            for i in range(0, len(secret_text_binary), 8):
                byte = secret_text_binary[i:i + 8]
                secret_text += chr(int(byte, 2))
            messagebox.showinfo("Decoded Message", "Secret Message:\n" + secret_text)
        else:
            messagebox.showwarning("Decoding Error", "No hidden message found in the image")
```

3.2.6. Function for decoding the secret image from the cover image: The function named ‘decode_image’ attempts to extract a hidden picture from a steganographic image.

(i) The first line opens a file dialog window to select an image file (in PNG format). The chosen file path is stored in the ‘filepath’ variable.

(ii) If a filepath is selected, the selected image is loaded using the ‘cv2.imread’ function from the OpenCV library. The loaded image is stored in the ‘img’ variable. The variables ‘width’ and ‘height’ are assigned the image's dimensions.

(iii) An empty image, ‘img2’, is created with the exact dimensions as the original image. This image will be used to store the extracted hidden image.

(iv) The code then iterates over each original image pixel using nested loops. For each pixel, it further iterates over the three color channels (red, green, and blue).

(v) The nested loops iterate over each original image pixel. Each pixel's least significant 4 bits (LSBs) of each colour channel are replaced with randomly generated bits. The variable `v1` stores the original 8-bit binary representation of the pixel value. The variable `v3` is created by combining the last 4 bits of `v1` with four randomly generated bits using ‘chr(random.randint(0, 1)+48)*4’, where adding 48 converts it to the ASCII code of '0' or '1'. The resulting binary string `v3` is converted back to an integer using ‘int(v3, 2)’ , and the value is assigned to the corresponding pixel in `img2`. This process modifies the LSBs of the pixel values in `img2` i.e assigns the integer value to the corresponding pixel in the blank image (`img2[i][j][l]`).

(vi) The modified image `img2` is saved as a PNG file named 'pic3_re.png'. Then, the saved image is opened using the `Image.open` function from the PIL library, and it is displayed in a new window using `img.show()`.

In summary, the `decode_image` function prompts the user to select a steganographic image, modifies the LSBs of the pixel values to extract the hidden picture, saves the extracted image, and displays it in a new window.

```
# Decode the secret message from the steganographic image image
def decode_image():
    filepath = filedialog.askopenfilename(filetypes=[("Image Files", "*.png")])
    if filepath:
        img = cv2.imread(filepath)
        width = img.shape[0]
        height = img.shape[1]
    # img2 is blank image
        img2 = np.zeros((width, height, 3), np.uint8)
        for i in range(width):
            for j in range(height):
                for l in range(3):
                    v1 = format(img[i][j][l], '08b')
                    v3 = v1[4:] + chr(random.randint(0, 1)+48) * 4
                    # Appending data to img2
                    img2[i][j][l]= int(v3, 2)
    # This is the image produced from the encrypted image
        cv2.imwrite('pic3_re.png', img2)
        img = Image.open('pic3_re.png')
        img.show() # Shows the image in a new window
```

3.2.7. Creating the main window:

(i) The ‘tkinter’ library is imported with the name ‘tk’ in this code. Then, a new instance of the ‘Tk’ class is created, representing the application's main window.

(ii) The ‘title’ method sets the window title to "STEGANOGRAPHY".

(iii) The ‘geometry’ method sets the window's dimensions to 500 pixels in width and 300 pixels in height.

(iv) The ‘resizable’ method is called with arguments ‘False, False’ to disable window resizing in horizontal and vertical directions.

(v) Lastly, the ‘configure’ method sets the padding (inner margins) of the window's content area. In this case, ‘padx=20, pady=20’ sets the padding to 20 pixels on both the x-axis and y-axis.

```
# Create the main window
window = tk.Tk()
window.title("STEGANOGRAPHY")
window.geometry("500x300")
window.resizable(False, False)
window.configure(padx=20, pady=20)
```

3.2.8. Function for creating a new window for Text in Image Steganography:

(i) The first three lines of code create a new window using Tkinter's ‘Toplevel’ class. The window is assigned to the variable ‘text_in_image_window’. The ‘title’ method sets the window title to "TEXT IN IMAGE - Steganography". The ‘geometry’ method sets the window's dimensions to 500 pixels in width and 300 pixels in height.

(ii) Inside the ‘text_in_image()’ function, there is another function definition named ‘select_images()’. This function is responsible for selecting a cover image and a secret text file, encoding the secret text into the cover image, and saving the steganographic image.

(iii) The ‘askopenfilename’ function from the ‘filedialog’ module is used to open file dialogs for selecting the cover image and secret text file. The chosen file paths are stored in the ‘cover_filepath’ and ‘text_filepath’ variables.

(iv) If both a cover image and a secret text file are selected, the code proceeds to read the cover image using ‘cv2.imread()’ and read the secret text from the file using a ‘with open’ block. The cover image is stored in the ‘cover_img’ variable, and the secret text is stored in the ‘secret_text’ variable.

(v) The ‘encode_text()’ function is then called to embed the secret text into the cover image, resulting in the steganographic image stored in the ‘stego_img’ variable.

(vi) Next, the ‘asksaveasfilename’ function from the ‘filedialog’ module is used to open a file dialog for selecting the save location of the steganographic image. The chosen save file path is stored in the ‘save_filepath’ variable.

(vii) If a save_filepath is selected, the steganographic image is saved using ‘cv2.imwrite()’, and a message box is shown using ‘messagebox.showinfo()’ to indicate that the image was encrypted and saved successfully.

Syntax: cv2.imwrite(filename, image)

Parameters: filename: A string representing the file name. The filename must include an image format like .jpg, .png, etc., image: It is the image to be saved.

Return Value: It returns true if the image is saved successfully.

(viii) Lastly, the code creates a button using Tkinter's ‘Button’ class. The variable ‘select_images_button’ is placed inside the ‘text_in_image_window’. The button displays the text "Select Image and Text File" and calls the ‘select_images()’ function when clicked. It is positioned at coordinates (170, 120) within the window using the ‘place()’ method.

```
# Text in Image Steganography
def text_in_image():
    # Create a new top-level window for text in image steganography
    text_in_image_window = tk.Toplevel(window)
    text_in_image_window.title("TEXT IN IMAGE - Steganography")
    text_in_image_window.geometry("500x300")
    # Function to select the cover image and secret text
    def select_images():
        # Open file dialogs to select the cover image and secret text file
        cover_filepath = filedialog.askopenfilename(filetypes=[("Image Files", "*.png")])
        text_filepath = filedialog.askopenfilename(filetypes=[("Text Files", "*.txt")])
        if cover_filepath and text_filepath:
            # Read the cover image and secret text
            cover_img = cv2.imread(cover_filepath)
            with open(text_filepath, 'r') as file:
                secret_text = file.read()
            # Embed the secret text into the cover image
            stego_img = np.copy(cover_img)
            encode_text(stego_img, secret_text)
            # Save the steganographic image
            save_filepath = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Files", "*.png")])
            if save_filepath:
                cv2.imwrite(save_filepath, stego_img)
                messagebox.showinfo("Steganography", "Image encrypted and saved successfully")
    # Create a button to select the cover image and secret text
    select_images_button = tk.Button(text_in_image_window, text="Select Image and Text File", command=select_images )
    select_images_button.place(x=170,y=120)
```

3.2.9. Function for creating a new window for Image in Image Steganography:

(i) The first three lines of code define a function named ‘image_in_image()’ that creates a new window for an image in image steganography. The code creates a new window using Tkinter's ‘Toplevel’ class. The window is assigned to the variable ‘image_in_image_window’. The window's title is "IMAGE IN IMAGE - Steganography". The geometry of the window is 500 in width and 300 in height.

(ii) Inside the ‘image_in_image()’ function, there is another function definition named ‘select_images()’. This function is responsible for selecting a cover image and a secret image, encoding the secret image into the cover image, and saving the steganographic image.

(iii) The ‘askopenfilename’ function from the ‘filedialog’ module is used to open file dialogs for selecting the cover and intimate images. The chosen file paths are stored in the ‘cover_filepath’ and `secret_filepath` variables.

(iv) If both a cover image and a secret image are selected, the code proceeds to read the cover image and the secret image using ‘cv2.imread()’. The cover image is stored in the ‘cover_img’ variable, and the secret image is stored in the `secret_img` variable.

(v) Resize the cover and secret images for the exact dimensions.

(vi) The ‘encode_image()’ function is then called to embed the secret image into the cover image, resulting in the steganographic image stored in the ‘stego_img’ variable.

(vii) Next, the ‘asksaveasfilename’ function from the ‘filedialog’ module is used to open a file dialog for selecting the save location of the steganographic image. The chosen save file path is stored in the `save_filepath` variable.

(viii) If a save file path is selected, the steganographic image is saved using ‘cv2.imwrite()’, and a message box is shown using ‘messagebox.showinfo()’ to indicate that the image was encrypted and saved successfully.

(ix) Lastly, the code creates a button using Tkinter's `Button` class. The variable `select_images_button` is placed inside the `image_in_image_window`. The button's text is set to "Select Images: Cover and Secret" and calls the ‘select_images()’ function when clicked. It is positioned at coordinates (170, 120) within the window using the ‘place()’ method.

```
# Image in Image Steganography
def image_in_image():
    # Create a new top-level window for image in image steganography
    image_in_image_window = tk.Toplevel(window)
    image_in_image_window.title("IMAGE IN IMAGE - Steganography")
    image_in_image_window.geometry("500x300")
    # Function to select the cover image and secret image
    def select_images():
        # Open file dialogs to select the cover image and secret image
        cover_filepath = filedialog.askopenfilename(filetypes=[("Image Files", "*.png")])
        secret_filepath = filedialog.askopenfilename(filetypes=[("Image Files", "*.png")])
        if cover_filepath and secret_filepath:
            # Read the cover image and secret image
            cover_img = cv2.imread(cover_filepath)
            secret_img = cv2.imread(secret_filepath)
            # Resize the images to have the same dimensions
            cover_img, secret_img = resize_images(cover_img, secret_img)
            # Embed the secret image into the cover image
            stego_img = np.copy(cover_img)
            encode_image(stego_img, secret_img)
            # Save the steganographic image
            save_filepath = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Files", "*.png")])
            if save_filepath:
                cv2.imwrite(save_filepath, stego_img)
                messagebox.showinfo("Steganography", "Image encrypted and saved successfully")
    # Create a button to select the cover image and secret image
    select_images_button = tk.Button(image_in_image_window, text="Select Images: Cover and Secret", command=select_images)
    select_images_button.place(x=170,y=120)
```

3.2.10. Creating Buttons for selecting the Steganography method:

(i) The code creates a button widget using the ‘Button’ class from the Tkinter module.

(ii) ‘window’ is the parent window or frame where the button will be placed.

(iii) text=”.....” sets the label or the text displayed on the button.

(iv) command=....... specifies the function to execute when the button is clicked.

(v) The `pack` method organises and displays the button within the parent window or frame. ‘pady=10’ sets the padding (vertical space) around the button to 10 pixels.

The lines of code create two buttons labelled "Text In Image" and "Image In Image" within a Tkinter window or frame. Each button is associated with a command function (‘text_in_image’ and ‘image_in_image’, respectively) executed when the button is clicked.

```
# Create buttons for selecting steganography method
text_in_image_button = tk.Button(window, text="Text In Image", command=text_in_image)
text_in_image_button.pack(pady=10)
image_in_image_button = tk.Button(window, text="Image In Image", command=image_in_image)
image_in_image_button.pack(pady=10)
```

3.2.11. Creating buttons for decoding the secret message and image from the steganographic image: The lines of code create two buttons labelled "Decode - Text" and "Decode - Image" within a Tkinter window or frame. Each button is associated with a command function (‘decode_text’ and ‘decode_image’, respectively) executed when the button is clicked.

```
# Create a button for decoding the secret message from the steganographic image
decode_button = tk.Button(window, text="Decode - Text", command=decode_text)
decode_button.pack(pady=10)
# Create a button for decoding the secret image from the steganographic image
decode_button = tk.Button(window, text="Decode - Image", command=decode_image)
decode_button.pack(pady=10)
```

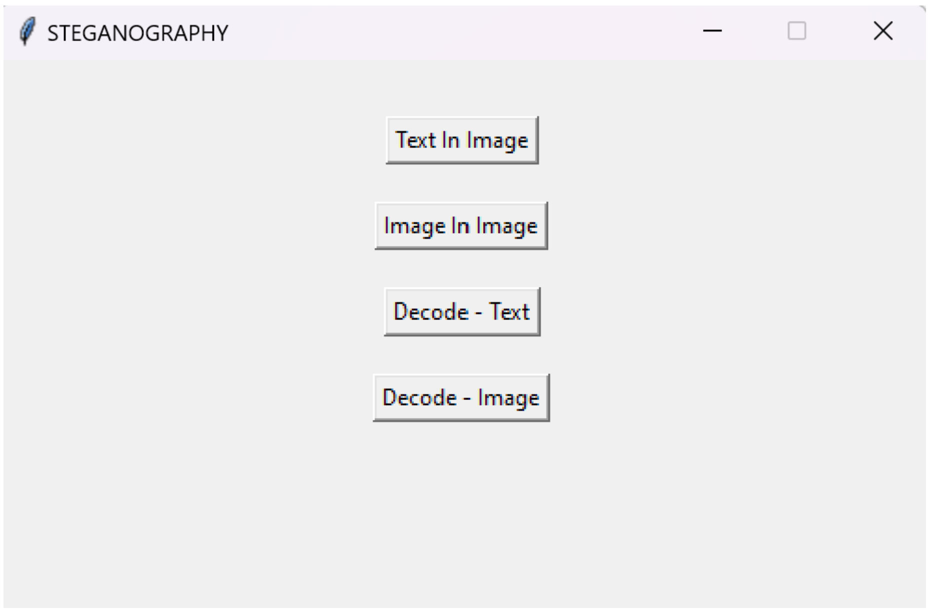
3.2.12. Run the Tkinter event loop: In Tkinter, mainloop() is a method that starts the main event loop of the Tkinter window. It continuously listens for events, such as button clicks or keyboard inputs, and updates the GUI accordingly. The mainloop() function runs indefinitely until the window is closed or an exit condition is met. It ensures that the GUI remains responsive and processes events as they occur. It's important to note that mainloop() should be called only once in your Tkinter application, typically at the end of your code after all the necessary widgets and event handlers have been set up. It keeps the GUI running and allows users to interact until they close the window.

```
# Run the Tkinter event loop
window.mainloop()
```

CHAPTER 4: RESULTS

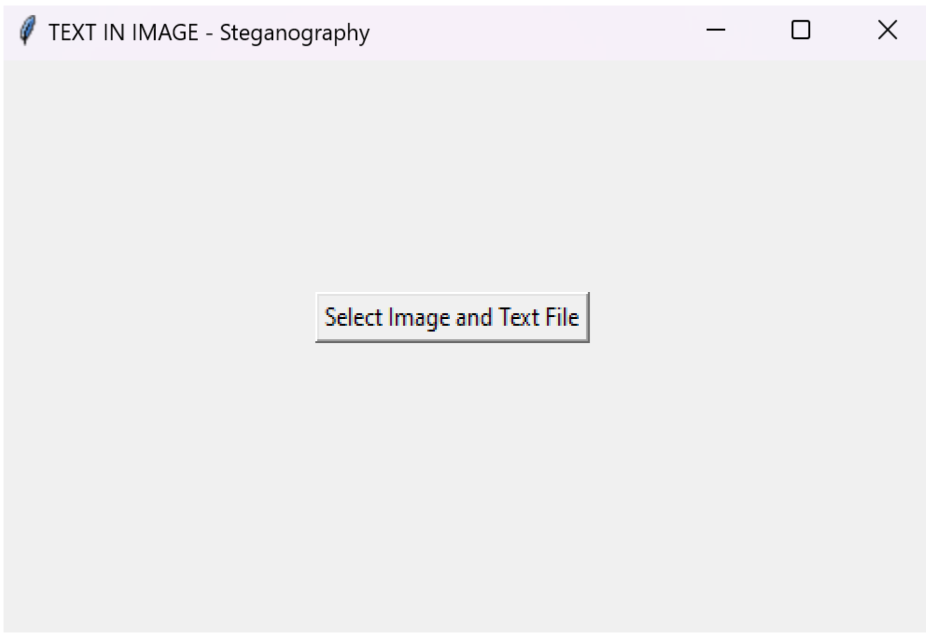
<u>**Output Screens:**</u>

After executing the code, the output GUI window looks as below.



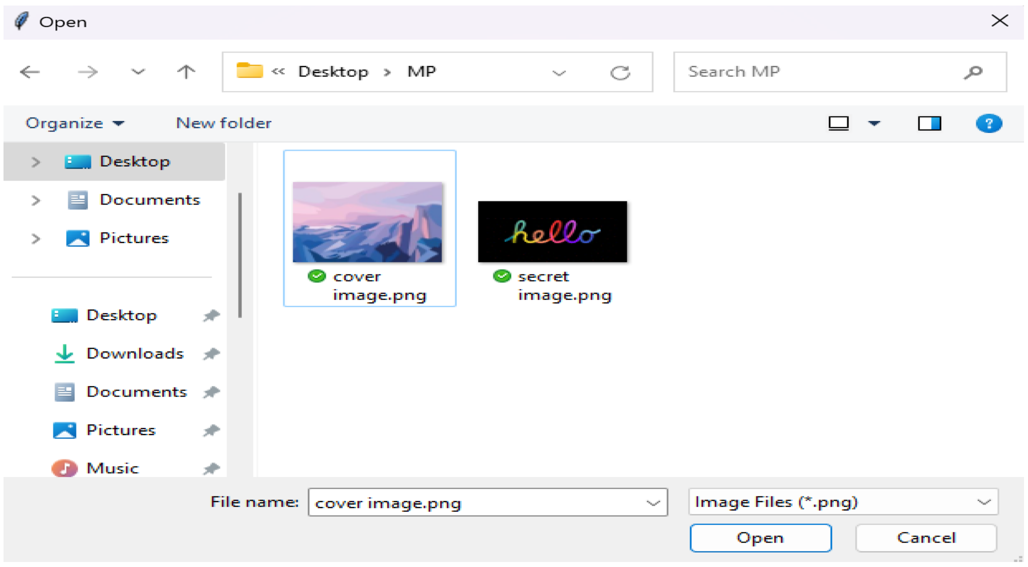
Text Steganography:

Click on ‘Text In Image’ on the main GUI screen.

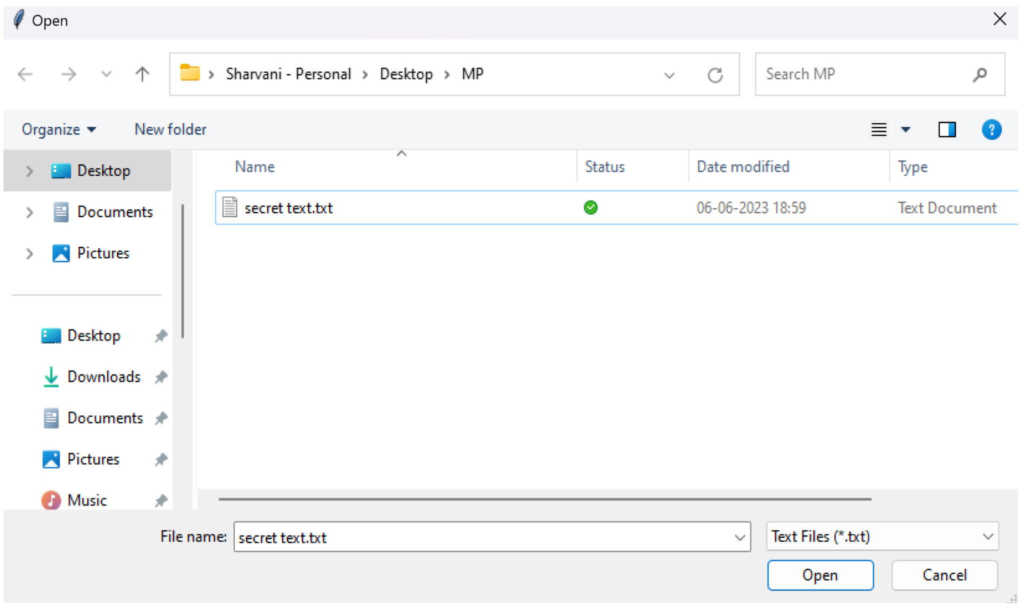


After clicking the button ‘Select Image and Text File’, the file dialog window opens to select the cover image and secret text that must be encrypted.

First, we will select the cover image in which the secret text has to be encrypted and click Open.

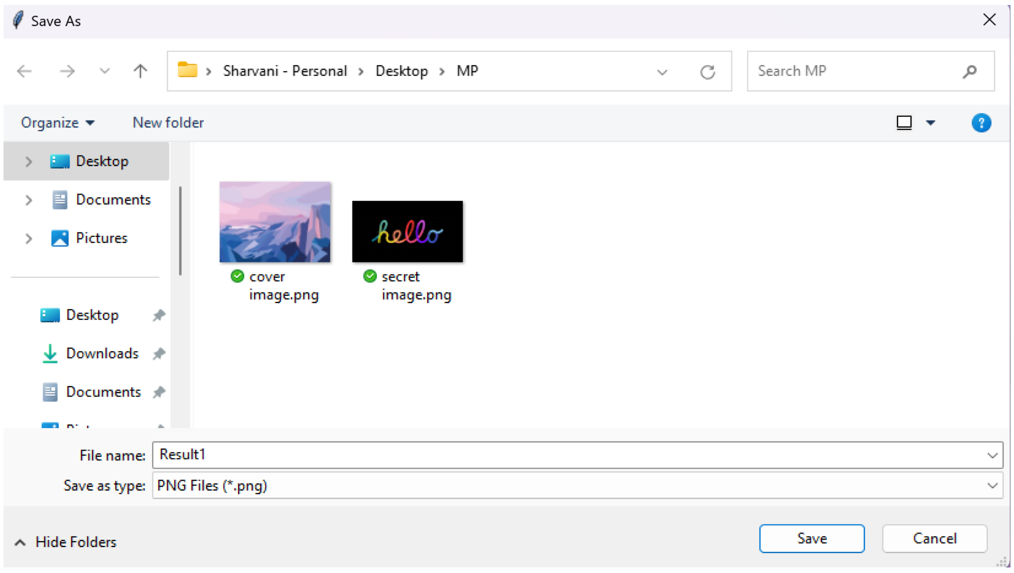


Then select the secret text file to be embedded within the cover image and click Open.



Name	Status	Date modified	Type
secret text.txt	✓	06-06-2023 18:59	Text Document

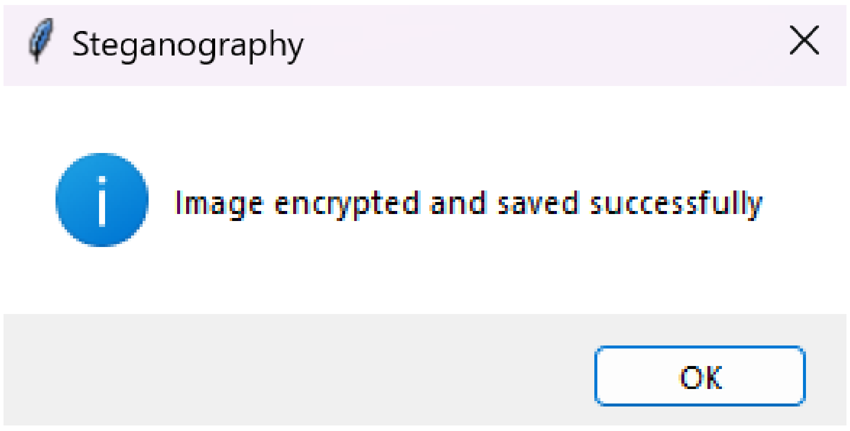
Now, it opens the window for saving the file. Here, we have saved our encrypted image with the filename ‘Result1’ and click Save.



File name: Result1

Save as type: PNG Files (*.png)

A message dialogue box is displayed below.



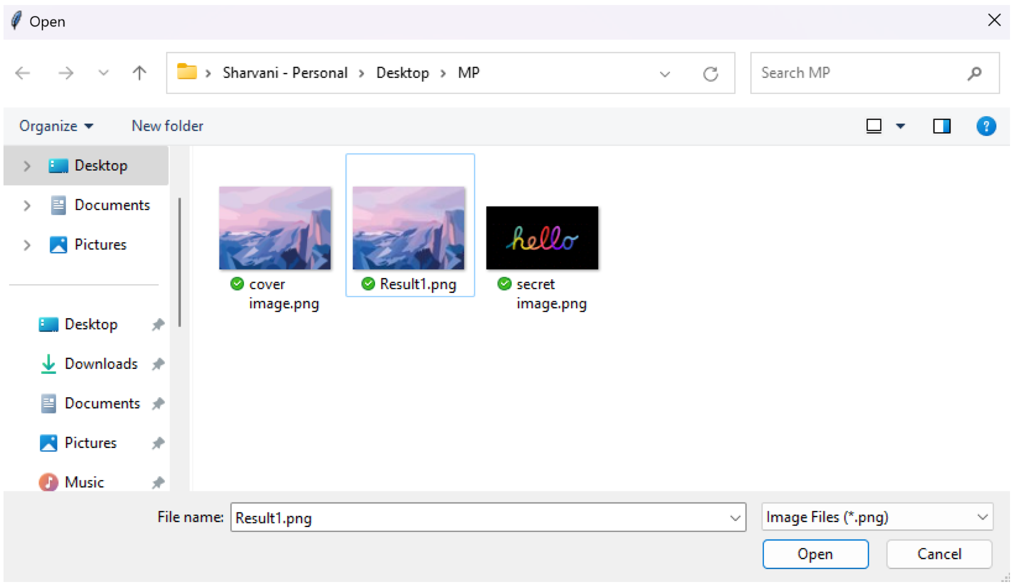
Steganography

Image encrypted and saved successfully

Till here, the process of encoding text within an image is done.

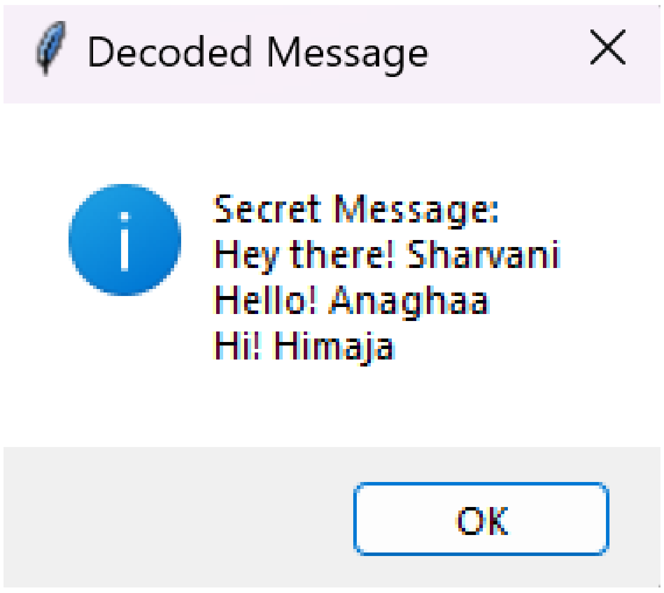
To decode the hidden text, Click the ‘Decode - Text’ button and here, one has to select the encrypted image to get the secret text in it.

Select the encrypted image ‘Result1’ and click Open as the file dialogue window opens.



File name: Result1.png
Image Files (*.png)

It displays the hidden text in the encrypted image.



Secret Message:
Hey there! Sharvani
Hello! Anaghaa
Hi! Himaja

Click OK.

The process of Text Steganography ends here.

Image Steganography:

Click on 'Image In Image' appears on the GUI window after executing the code.

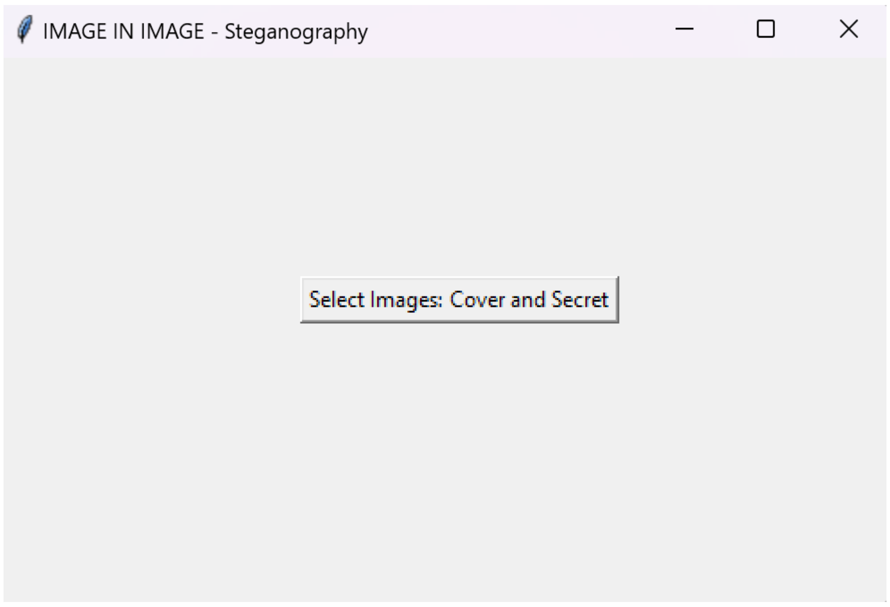
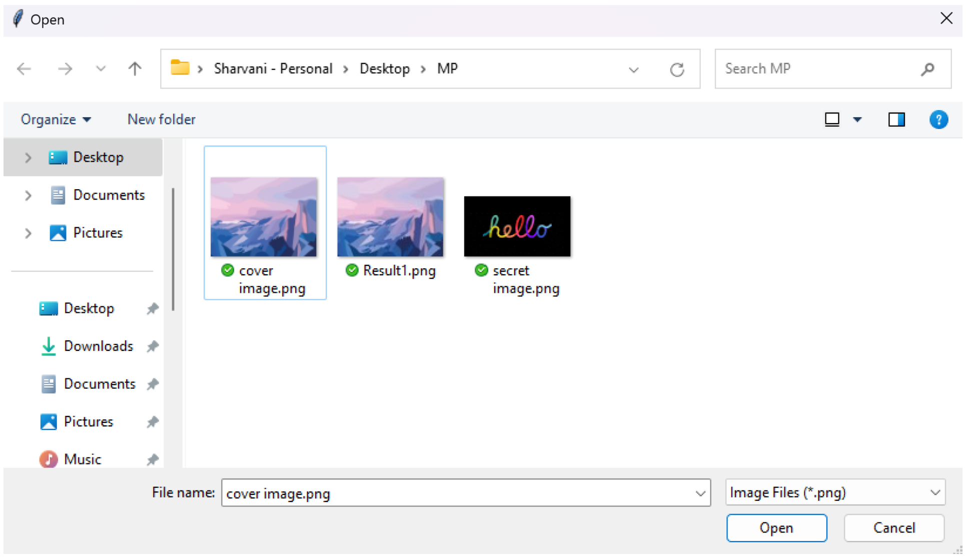


IMAGE IN IMAGE - Steganography
Select Images: Cover and Secret

After clicking the 'Select Images: Cover and Secret' button, the file dialog window opens to select the cover and secret images that must be encrypted.

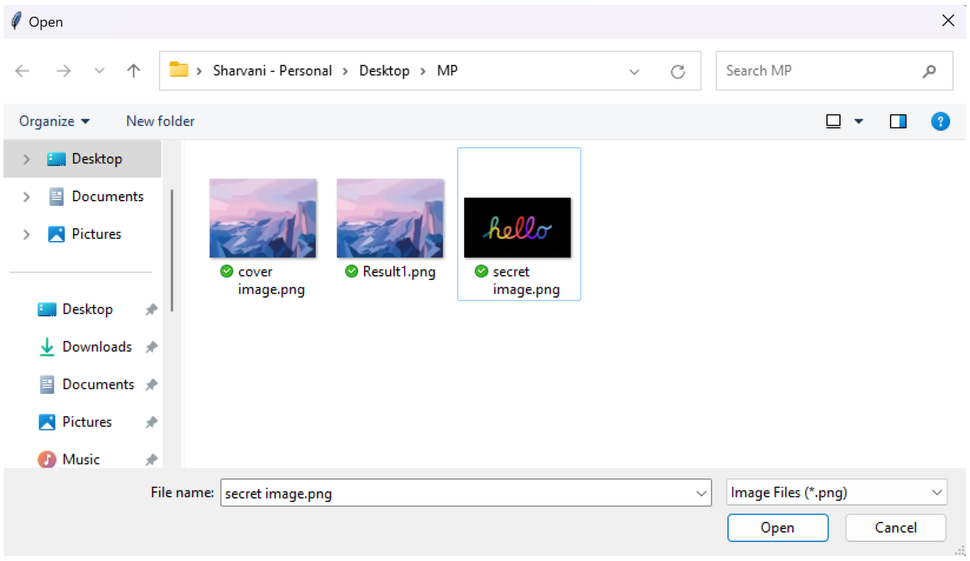
First, we will select the cover image in which the secret image has to be encrypted and click Open.



File name: cover image.png

Image Files (*.png)

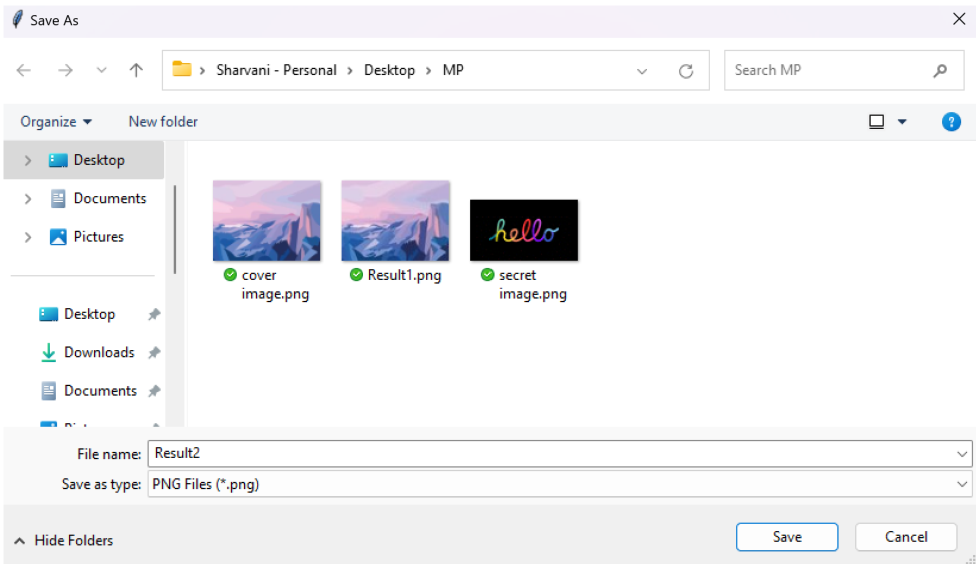
Then select the secret image file to be embedded within the cover image and click Open.



File name: secret image.png

Image Files (*.png)

Now, it opens the window for saving the file. Here, we have saved our encrypted image with the filename 'Result2' and click Save.



File name: Result2

Save as type: PNG Files (*.png)

A message dialogue box is displayed below.

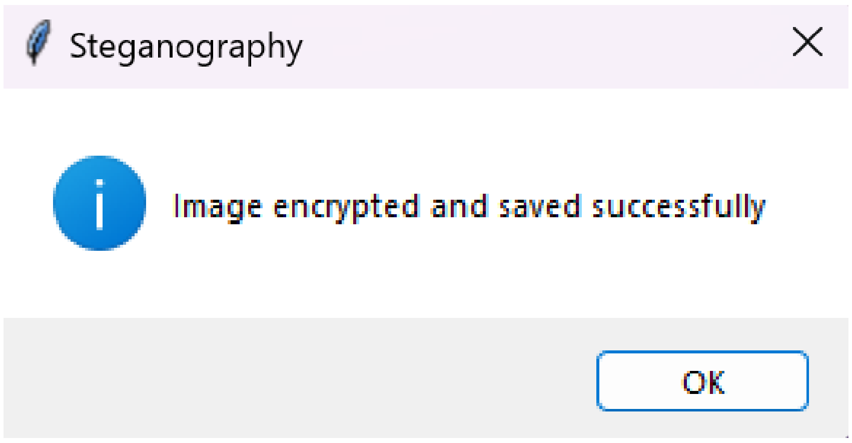
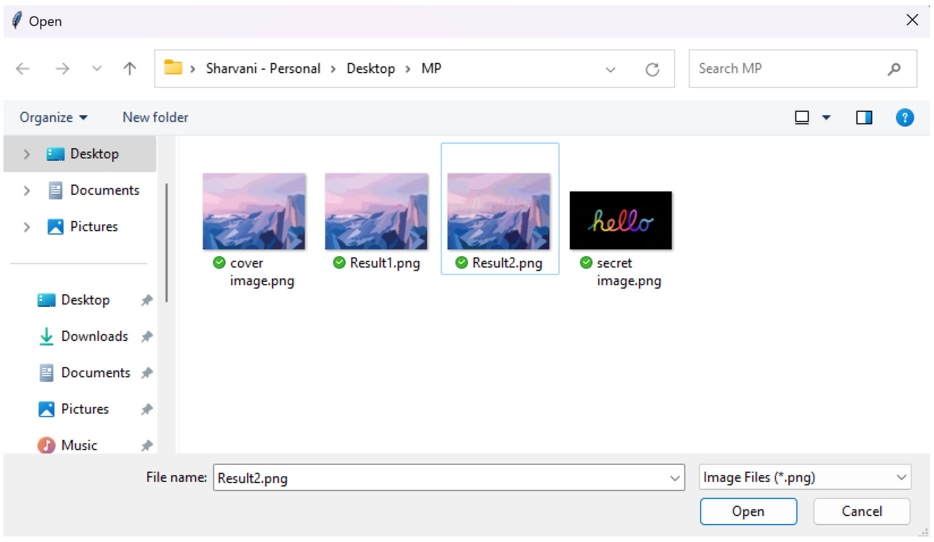

Steganography

Image encrypted and saved successfully

Till here, the process of encoding an image within an image is done.

To decode the hidden image, Click the ‘Decode - Image’ button and here, one has to select the encrypted image to get the secret image in it.

Select the encrypted image ‘Result2’ and click Open as the file dialogue window opens.



File name: Result2.png

Image Files (*.png)

It displays the hidden image in the encrypted image.

The process of Image Steganography ends here.

CHAPTER 5: CONCLUSION

In this mini-project, we created a user-friendly steganography tool using Python and Tkinter. The tool allowed users to hide secret messages or images within cover images discreetly. We implemented two methods: text-in-image steganography and image-in-image steganography. The text was encoded by modifying the least significant bits of cover image pixels, while for images, we combined the 4 most significant bits of the cover and secret images. The tool demonstrated efficiency, versatility, and support for various image formats. It found applications in secure communication, data hiding, digital watermarking, and covert messaging. However, it had limitations like limited data capacity and vulnerability to steganalysis. Overall, the project offered valuable insights into steganography concepts and provided a foundation for exploring more advanced techniques in the future.

CHAPTER 6: FUTURE SCOPE

The project aims to limit unauthorized access and provide better security during message transmission. We use a simple and basic approach to image steganography to meet the requirements. In the future, the most crucial use of steganographic techniques will probably lie in digital watermarking. The combination of both Cryptography and steganography provides a higher level of security to the information being transmitted.
The most challenging thing in steganography is to keep intruders away from identifying the difference between the original cover media file and the stego-cover file. Apart from traditional algorithms like LSB, many more advanced algorithms are in use today, and each has its pros and cons even more techniques are being introduced. Recently, research on image steganography has demonstrated great potential by introducing GAN and other neural network techniques.

CHAPTER 7: REFERENCES

[1] https://data-flair.training/blogs/python-image-steganography-project/

[2] https://www.geeksforgeeks.org/lsb-based-image-steganography-using-matlab/

[3] https://www.geeksforgeeks.org/image-steganography-using-opencv-in-python/?ref=rp

[4] https://www.geeksforgeeks.org/image-based-steganography-using-python/

[5]
https://betterprogramming.pub/a-guide-to-video-steganography-using-python-4f010b32a5b7/

[6] https://www.ripublication.com/irph/ijisaspl2019/ijisav11n1spl_19.pdf

[7] https://www.ijltet.org/wp-content/uploads/2015/02/60.pdf