

INTERDISCIPLINARY PROJECT REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By
KISHAN.T (Reg.No.40733018)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU

APRIL 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **KISHAN.T (40733018)** who carried out the project entitled "Image Steganographer" under my supervision from FEB 2023 to APRIL 2023.

Internal Guide

Ms.K.ABINAYA

Head of the Department

Dr.S.Vigneshwari

Submitted for Viva voce Examination held on 17-04-2023

Internal Examiner

External Examiner

DECLARATION

I **KISHAN.T** hereby declare that the Project Report entitled IMAGE STEGANOGRAPHER done by me under the guidance of Dr./Prof./Mr./Ms. _____ (Internal) and _____ (External) at TG CYBERLABS is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 17-04-2023

PLACE: Chennai



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr.S.Vigneshwari M.E., Ph.D** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide -----
-----for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE

CERTIFICATE OF COMPLETION

This certificate is awarded to:

Kishan T

has completed the Professional Training Program conducted by
Techgreech Cyberlabs Private Limited from 14/02/2023 - 28/04/2023



Mr. Sriram
Director, TG Cyberlabs

TG Cyberlabs.
Security our Priority...

Cert-ID: TGCYBLS-PT2-0035

ABSTRACT

The project: “Image Steganograph” is a novel hash-based approach for color image steganography. As, the available approaches for color image steganography are using chaos-based and symmetric-key based cryptographic algorithms are not efficient and good for bulky data. However, the hash algorithm based approaches are considerably better in terms of providing better speed but these approaches are vulnerable in terms of providing security due to inherent flaws caused by used checksum approach. The key reason of vulnerability is that the used algorithms in such approaches such as MD5 and SHA-2 have flaws. In our approach, we purpose the use of perfect hash-function algorithm to provide a secure and fast approach for color image steganography. We also present a prototype tool in this paper that is implementation of the presented approach and is also a proof of concept. Another contribution of the approach is that the presented approach can be used for coding data in any type of color images such as bmp, jpeg, gif, and tiff as other available approaches are file format specific. The results of the initial experiments are very encouraging and support not only the used approach but also uphold the potential of the presented approach in general.

Chapter No	TITLE		Page No.
	ABSTRACT		vi
	LIST OF FIGURES		viii
1	INTRODUCTION		1
	1.1	Application of Image Steganography	2
2	AIM AND SCOPE OF IMAGE STEGANOGRAPHER		3
	2.1	AIM	3
	2.2	OBJECTIVE	3
3	METHODOLOGY AND ALGORITHMS USED		5
	3.1	SYSTEM REQUIREMENTS	5
	3.2	PACKAGES USED	5
	3.3	LSB – LEAST SIGNIFICANT BITS	6
	3.4	AES ALGORITHM	7
	3.5	SHA ALGORITHM	8
	3.6	IMAGE STEGANOGRAPHY ALGORITHM	9
4	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS		12
	4.1	Results	12
5	SUMMARY AND CONCLUSION		13
	5.1	Summary	13
	5.2	Conclusion	13
	APPENDIX		14
	A. SOURCE CODE		14
	B. SCREENSHOTS		33
	REFERENCES		37

LIST OF FIGURES

FIGURE No	FIGURE NAME	PAGE No
1.	Stenography Framework	1
2.	Cover image	6
3.	Stegao image	6
4.	Flowchart of aes	8
5.	Flowchart of sha	9
6.	encryption	11
7.	decryption	11
8.	screenshots	33-36

LIST OF ABBREVIATIONS

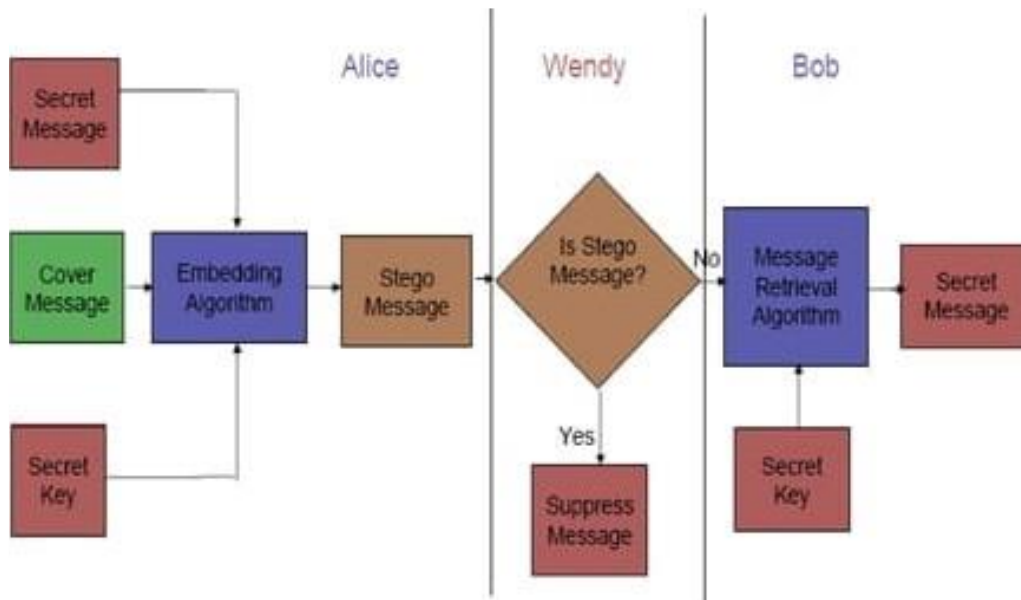
ABBREVIATION	EXPANSION
AES	Advanced Encryption Standard
SHA	Secure Hash Algorithm
OS	Operating System
IDE	Integrated Development Environment
LSB	Least Significant Bit

CHAPTER 1

INTRODUCTION

Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message.

Steganography Framework:



- The Project "Image Steganography" is based on Image Steganography.
- Image Steganography is the technique of hiding the data within the image in such a way that prevents the unintended user from the detection of the hidden messages or data.



cover image



Data/message



stegao image

1.1 Applications of Image Steganography:

- Passwords and encryption key
- Private Files and Documents.
- Hide Passwords and EncryptionKeys.
- Transport Highly Private Documents between International and government.
- Transmit message/data without revealing the existence of available message.

CHAPTER-2

AIM AND SCOPE OF IMAGE STEGANOGRAPHER

2.1 AIM:

The aim of image steganography is to provide a secure and covert method for communicating sensitive information. By hiding secret data within an image, steganography can be used for various purposes such as confidential communication, data hiding, digital watermarking, and copyright protection.

2.2 OBJECTIVE:

- The main objective of image steganography is to provide a secure and covert method for communicating sensitive information. This involves hiding a secret message within an image in such a way that the image appears unchanged to the human eye. The objective of image steganography can be further broken down into the following specific objectives:
- Concealment: The primary objective of image steganography is to conceal the secret message within the image in a way that it is not detectable by unauthorized parties. The message should be hidden in such a way that it cannot be easily accessed, decoded or extracted.
- Security: The secret message should be secure from unauthorized access, modification, or destruction during transmission or storage.

- It should be protected using secure encryption algorithms to prevent unauthorized access or interception.
- Robustness: The steganographic algorithm should be able to withstand various image processing operations such as compression, filtering, and noise addition without affecting the quality of the hidden message.
- Efficiency: The steganographic algorithm should be computationally efficient and fast to encode and decode the hidden message in the image.
- Integration: The steganographic algorithm should be easily integrated with other image processing techniques and algorithms.
- Evaluation: The steganographic algorithm should be evaluated for its performance and effectiveness using various metrics such as PSNR, MSE, and BER.
- Overall, the objective of image steganography is to provide a secure and covert method of communication, while also ensuring that the quality of the image is not compromised. The development of new and more sophisticated algorithms and techniques in steganography has expanded its applications and made it a powerful tool for securing sensitive information.

CHAPTER – 3

METHODOLOGY AND ALGORITHMS USED

3.1 SYSTEM REQUIREMENTS:

****ANY SYSTEM WHICH CONTAINS MINIMUM OF 4GB RAM AND WITH ANY OS CAN BE ACCESSIBLE****

LANGUAGE USED: PYTHON

ENVIRONMENT: VISUAL STUDIO CODE.

3.2 PACKAGES USED:

- **hashlib:** to perform hash functions using SHA.
- **os:** it includes various functions for interacting with files, directories, processes, and environment variables.
- **numpy:** to support multi-dimensional arrays like the images.
- **cryptodome:** it provides a comprehensive set of cryptographic algorithms like AES, DES, MD5, SHA, RSA, DSA, etc.
- **PIL (a.k.a) pillow:** It allows you to hide a secret message within an image using a simple LSB (Least Significant Bit) technique.
- **Tkinter:** used to create a Graphical User Interfaces(GUI's) in python.
- **ctypes:** It provides C compatible data types, and allows calling functions in DLLs (Dynamic link library) or shared libraries. Used to run this gui of this project in windows os.

3.3 LSB – LEAST SIGNIFICANT BIT:

Suppose we have the following binary representation for the Cover Img

```
10D10101 00001101  
10010110 00001111
```

Suppose we want to "hide" the following q bits of data: 1011, we get the following,

```
10D10101 00001101  
10010110 00001111
```

Where each data bits are accommodated in the least significant bits of each byte of the image.

Least Significant Bit Substitution results in a very minor distortion of the image which is very much negligible for the human eyes.

Cover Image



Stego image

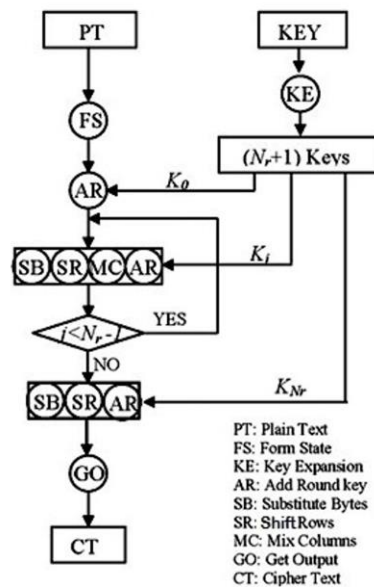


3.4 AES Encryption:

- AES encryption uses a symmetric key, which means that the same key is used for both encryption and decryption of the data. The key length can vary from 128 bits, 192 bits, to 256 bits. The longer the key length, the more secure the encryption is.
- AES encryption operates on a block of data, typically 128 bits at a time, and uses a series of mathematical operations such as substitution, permutation, and transformation to transform the plaintext into ciphertext. The encryption process involves several rounds of these operations, and the number of rounds depends on the key length. For example, AES-128 uses 10 rounds, AES-192 uses 12 rounds, and AES-256 uses 14 rounds.
- algorithm of AES encryption:
 1. Key expansion: generate a set of round keys from the initial key using the Rijndael key schedule.
 2. Initial round: add the first round key to the plaintext block.
 3. Main rounds: perform a series of substitution, permutation, and mixing operations on the block using the round keys.
 4. Final round: perform a final set of operations on the block without the mixing step.
 5. Output: the resulting block after the final round is the encrypted ciphertext.

- Flowchart:

Flowchart of AES:

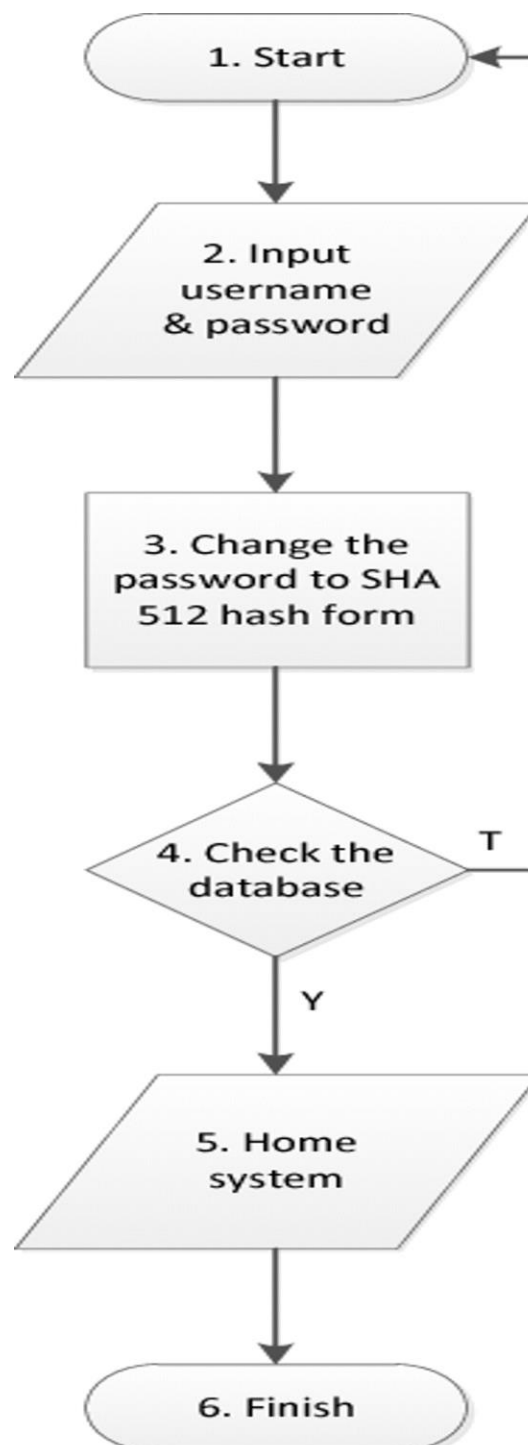


3.5 SHA Algorithm:

- SHA stands for secure hashing algorithm. SHA is a modified version of MD5 and used for hashing data and certificates. A hashing algorithm shortens the input data into a smaller form that cannot be understood by using bitwise operations, modular additions, and compression functions.
- **Algorithm:**
 1. Initialize hash values: initialize a set of hash values (constants) that will be used in the hashing process.
 2. Pad message: append padding to the message to bring it to a length that is a multiple of the block size. The padding includes a length value indicating the original length of the message.
 3. Process message in blocks: divide the padded message into fixed-size blocks and process each block.

4. Update hash values: update the hash values for each block based on the current state of the hash values and the contents of the block.
5. Output: the final hash value is the concatenation of the hash values computed for each block.

6. Flowchart:



3.6 IMAGE STEGANOGRAPHY ALGORITHM:

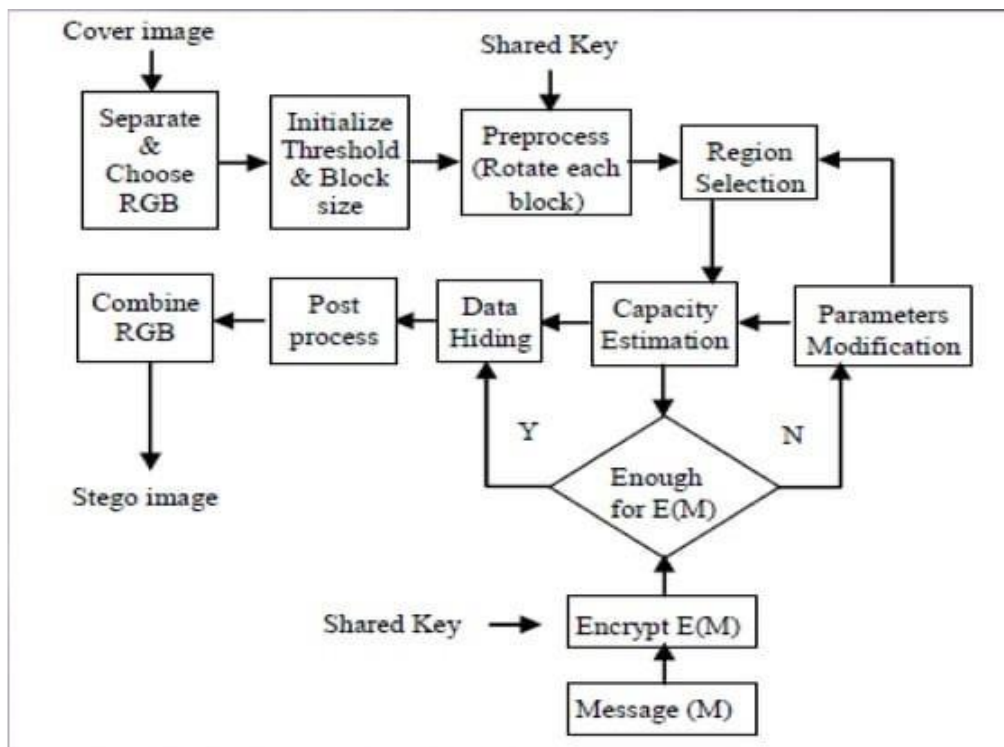
1) encode the secret message into the image, the following steps can be used

1. Convert the cover image to binary format.
2. Convert the secret message to binary format.
3. Calculate the SHA-256 hash value of the secret message.
4. Use the hash value to generate a secret key for AES encryption.
5. Use AES encryption to encrypt the secret message with the secret key.
6. Embed the encrypted message into the cover image using a steganography technique such as LSB (Least Significant Bit) substitution.
7. Save the stego-image.

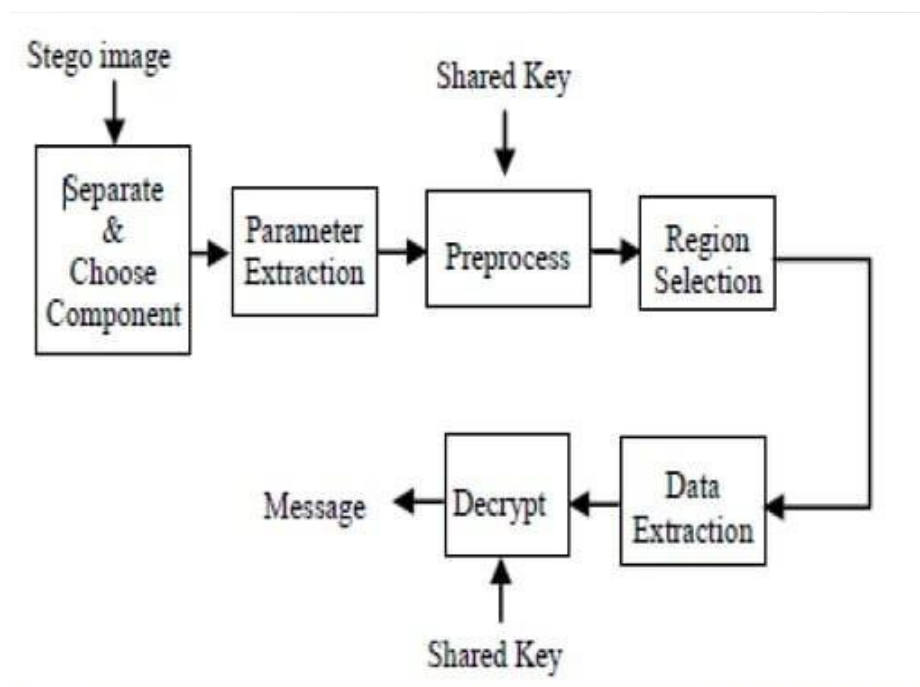
2) extract the secret message from the stego-image, the following steps can be used:

1. Retrieve the stego-image.
2. Convert the stego-image to binary format.
3. Extract the encrypted message from the stego-image using the same steganography technique used for embedding.
4. Use the hash value to regenerate the secret key for AES decryption.
5. Use AES decryption to decrypt the secret message.
6. Convert the decrypted message from binary format to its original form.

ENCRYPTION:



DECRYPTION:



CHAPTER-4

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

4.1 RESULTS:

- We can encrypt and decrypt secret messages easily and securely.
- Performance of edge adaptive steganography for color image with increasing values of parameters is analyzed.
- In addition to that, color image is separated into RGB layers and then data hiding is performed. The image quality after data embedding is very important for better performance of steganography methods.
- When threshold is high only minimum set of characters can be embedded. As the threshold value decreases more characters can be implanted.
- Separation of image into RGB component results increases the embedding capacity.
- This method encrypts the message, which improves the security, This scheme can be applied to other covers such as audio and video which is taken as the future work.

CHAPTER-5

SUMMARY AND CONCLUSION

5.1 SUMMARIZE:

We are able to perform image steganography by encrypting the message inside the cover images and decrypt the stego file to find the real secret message.

We were enable to perform the encryption using just AES encryption technique and SHA hashing algorithm.

5.2 CONCLUSION:

Therefore we can image steganography has still working as on of the best way to encrypt the for image steganography is by using LSB and encrypt the message with AES encryprion and hash the secure key in Chennai.

APPENDIX:

A. Source code:

encode.py

```
import hashlib

import os.path

import numpy as np

from Crypto.Cipher import AES

from PIL import Image


class Encode:

    def __init__(self, image_path, password, text_to_encode):

        self.image_path = image_path

        self.password = password.strip()

        self.text_to_encode = text_to_encode.strip()


    def is_password_valid(self):

        if len(self.password) == 0:

            return False

        return True


    def is_text_valid(self):
```

```
if len(self.text_to_encode) == 0:
```

```
    return False
```

```
return True
```

```
def is_image_path_valid(self):
```

```
    if os.path.exists(self.image_path):
```

```
        return True
```

```
return False
```

```
def get_text_binary(self):
```

```
    secret_key = hashlib.sha1(str(self.password).encode()).hexdigest()[:32]
```

```
    encryption_key = AES.new(secret_key.encode('utf-8'), AES.MODE_EAX,
```

```
secret_key.encode())
```

```
    encrypted_text = encryption_key.encrypt(self.text_to_encode.encode('utf-8'))
```

```
    encrypted_text = str(encrypted_text)
```

```
    encrypted_text += "$@&#"
```

```
    binary_value = ".join([format(ord(character), "08b") for character in encrypted_text])
```

```
    return binary_value
```

```
def encode_into_image(self):
```

```
    try:
```

```
        raw_image = Image.open(self.image_path, 'r')
```

```
        width, height = raw_image.size
```

```
        channels = 3
```

```
        if raw_image.mode == 'RGBA':
```

```

        channels = 4

        image_array = np.array(list(raw_image.getdata()))

        image_size = image_array.size // channels

        binary_value = self.get_text_binary()

        secret_hash = str(int(hashlib.md5(self.password.encode('utf-8')).hexdigest(),
16))[:5]

        if int(secret_hash) > image_size:

            secret_hash = secret_hash[:4]

            if int(secret_hash) > image_size:

                secret_hash = secret_hash[:3]

                if int(secret_hash) > image_size:

                    secret_hash = secret_hash[:2]

                    if int(secret_hash) > image_size:

                        secret_hash = secret_hash[:1]

                        if int(secret_hash) > image_size:

                            return ['Image size is not sufficient to encode the given text.', False]

        text_size = len(binary_value)

        encode_space = image_size - int(secret_hash)

        retro_encode = False

        if text_size > encode_space:

            retro_encode = True

        if text_size > image_array.size:

            return ['Image size is not sufficient to encode the given text.', False]

        else:

            bin_index = 0

```

```

for pixel in range(int(secret_hash), image_size):
    for channel in range(0, channels):
        if bin_index < text_size:
            image_array[pixel, channel] = int(bin(image_array[pixel][channel])[2:9] +
                                                binary_value[bin_index], 2)

            bin_index += 1

```

```

if retro_encode:
    for pixel in range(int(secret_hash)):
        for channel in range(0, channels):
            if bin_index < text_size:
                image_array[pixel, channel] = int(bin(image_array[pixel][channel])[2:9]
+
                                                    binary_value[bin_index], 2)

                bin_index += 1

```

```

image_array = image_array.reshape(height, width, channels)

stego_image = Image.fromarray(image_array.astype('uint8'), raw_image.mode)

return [stego_image, True]

```

except Exception:

```

return ['Unidentified Error. The possible reasons might be\n'
        '\n1. Unsupported Image File'
        '\n2. Invalid text characters', False]

```

```

def are_values_valid(self):

```

```
if not self.is_password_valid():  
    return ["Password can't be empty.", False]  
  
elif not self.is_text_valid():  
    return ["Text to encode can't be empty.", False]  
  
elif not self.is_image_path_valid():  
    return ["Selected image doesn't exist anymore.", False]  
  
else:  
    return ["Validated", True]
```

decode.py

```
import hashlib

import os.path

import numpy as np

from Crypto.Cipher import AES

from PIL import Image


class Decode:

    def __init__(self, image_path, password):

        self.image_path = image_path

        self.password = password.strip()


    def is_password_valid(self):

        if len(self.password) == 0:

            return False

        return True


    def is_image_path_valid(self):

        if os.path.exists(self.image_path):

            return True

        return False


    def get_decoded_text(self, bytes_string):
```

```

secret_key = hashlib.sha1(str(self.password).encode()).hexdigest()[:32]

decryption_key = AES.new(secret_key.encode('utf-8'), AES.MODE_EAX,
secret_key.encode())

bytes_value = eval(bytes_string)

return decryption_key.decrypt(bytes_value).decode('utf-8')

def decode_from_image(self):

    try:

        raw_image = Image.open(self.image_path, 'r')

        channels = 3

        if raw_image.mode == 'RGBA':

            channels = 4

        image_array = np.array(list(raw_image.getdata()))

        image_size = image_array.size // channels

        secret_hash = str(int(hashlib.md5(self.password.encode('utf-8')).hexdigest(),
16))[:5]

        if int(secret_hash) > image_size:

            secret_hash = secret_hash[:4]

        if int(secret_hash) > image_size:

            secret_hash = secret_hash[:3]

        if int(secret_hash) > image_size:

            secret_hash = secret_hash[:2]

        if int(secret_hash) > image_size:

            secret_hash = secret_hash[:1]

        if int(secret_hash) > image_size:

```

```

        return ["Encoded text not found. The possible reasons might
be:\n"

               "\n1. Incorrect Password"

               "\n2. Wrong Image", False]

binary_value = ""

for pixel in range(int(secret_hash), image_size):

    for channel in range(0, channels):

        binary_value += bin(image_array[pixel][channel])[-1]


for pixel in range(int(secret_hash)):

    for channel in range(0, channels):

        binary_value += bin(image_array[pixel][channel])[-1]


binary_list = [binary_value[value:value + 8] for value in range(0,
len(binary_value), 8)]

decoded_text = ""

for i in range(len(binary_list)):

    if decoded_text[-4:] == "$@&#":

        break

    else:

        decoded_text += chr(int(binary_list[i], 2))


if "$@&#" in decoded_text:

    decoded_text = decoded_text[:-4]

    try:

```

```

        decoded_text = self.get_decoded_text(decoded_text)

        return [decoded_text, True]

    except UnicodeDecodeError:

        return ["Encoded text not found. The possible reasons might be:\n"

                "\n1. Incorrect Password"

                "\n2. Wrong Image", False]

    else:

        return ["Encoded text not found. The possible reasons might be:\n"

                "\n1. Incorrect Password"

                "\n2. Wrong Image", False]

    except Exception:

        return ["Encoded text not found. The possible reasons might be:\n"

                "\n1. Incorrect Password"

                "\n2. Wrong Image", False]

def are_values_valid(self):

    if not self.is_password_valid():

        return ["Password can't be empty.", False]

    elif not self.is_image_path_valid():

        return ["Selected image doesn't exist anymore.", False]

    else:

        return ["Valid", True]

```

imgstegano.py

```
from tkinter import *

from ctypes import windll

from tkinter import filedialog, messagebox

from PIL import ImageTk, Image

import encode

import decode


encode_opened = False

decode_opened = False

about_opened = False

windows = True

file_path = ""


def open_encode_window():

    global encode_opened

    if not encode_opened:

        encode_window = Toplevel(window)

        encode_window.title("Image stenographer - Encode")

        encode_window.geometry('800x500')

        encode_window.resizable(True, True)

    if windows:

        windll.shcore.SetProcessDpiAwareness(1)
```

```
raw_image_label = Label(encode_window, text="Select Raw Image", height=20,  
width=50, relief="solid",
```

```
bg="#FFFFFF")
```

```
raw_image_label.place(x=20, y=20)
```

```
text_to_encode_label = Label(encode_window, text="Text to Encode")
```

```
text_to_encode_label.config(font=("Open Sans", 12))
```

```
text_to_encode_label.place(x=400, y=20)
```

```
text_to_encode = Text(encode_window, height=7, width=34)
```

```
text_to_encode.config(relief="solid", font=("Open Sans", 15))
```

```
text_to_encode.place(x=400, y=51)
```

```
pass_to_encode_label = Label(encode_window, text="Password")
```

```
pass_to_encode_label.config(font=("Open Sans", 12))
```

```
pass_to_encode_label.place(x=400, y=262)
```

```
pass_to_encode = Entry(encode_window, width=34)
```

```
pass_to_encode.config(relief="solid", font=("Open Sans", 15), show="*")
```

```
pass_to_encode.place(x=400, y=293)
```

```
browse_image_btn = Button(encode_window, text="Browse Raw Image", width=29,  
cursor="hand2",
```

```
command=lambda: browse_image(raw_image_label))
```

```

        browse_image_btn.config(font=("Open Sans", 15), bg="#36923B", fg="white",
borderwidth=0)

        browse_image_btn.place(x=20, y=350)


        encode_image_btn = Button(encode_window, text="Encode", width=15,
cursor="hand2",

                                command=lambda: encode_image(file_path, pass_to_encode.get(),
                                                                text_to_encode.get("1.0", END)))

        encode_image_btn.config(font=("Open Sans", 15), bg="#503066", fg="white",
borderwidth=0)

        encode_image_btn.place(x=592, y=420)


        encode_opened = True

        encode_window.protocol("WM_DELETE_WINDOW", lambda:
close_encode_window(encode_window))


def save_image(stego_image):

    save_path = filedialog.asksaveasfile(initialfile="encryptstego.png", mode="wb",
defaulttextextension=".png",

                                filetypes=(("Image File", "*.png"), ("All Files", "*.*")))

    stego_image.save(save_path)


def encode_image(image_path, password, text_to_encode):

```

```

encode_action = encode.Encode(image_path, password, text_to_encode)

msg = encode_action.are_values_valid()

if not msg[1]:

    messagebox.showerror("Error Encoding", msg[0])

else:

    stego_image = encode_action.encode_into_image()

    if stego_image[1]:

        if save_image(stego_image[0]) is None:

            messagebox.showinfo("Image Saved", "Encode operation was successful.")

        else:

            messagebox.showerror("Error Encoding", stego_image[0])

```

```

def open_decode_window():

    global decode_opened

    if not decode_opened:

        decode_window = Toplevel(window)

        decode_window.title("Image Stenographer - Decode")

        decode_window.geometry('800x500')

        decode_window.resizable(True, True)

        if windows:

            windll.shcore.SetProcessDpiAwareness(1)

        stego_image_label = Label(decode_window, text="Select Stego Image", height=20,
width=50, relief="solid",

```

```
        bg="#FFFFFF")

stego_image_label.place(x=20, y=20)


text_to_decode_label = Label(decode_window, text="Decoded Text")
text_to_decode_label.config(font=("Open Sans", 12))
text_to_decode_label.place(x=400, y=20)


text_to_decode = Text(decode_window, height=7, width=34)
text_to_decode.config(relief="solid", font=("Open Sans", 15), state=DISABLED)
text_to_decode.place(x=400, y=51)


pass_to_decode_label = Label(decode_window, text="Password")
pass_to_decode_label.config(font=("Open Sans", 12))
pass_to_decode_label.place(x=400, y=262)


pass_to_decode = Entry(decode_window, width=34)
pass_to_decode.config(relief="solid", font=("Open Sans", 15), show="*")
pass_to_decode.place(x=400, y=293)


browse_stego_btn = Button(decode_window, text="Browse Stego Image", width=29,
cursor="hand2",

                        command=lambda: browse_image(stego_image_label))

browse_stego_btn.config(font=("Open Sans", 15), bg="#503066", fg="white",
borderwidth=0)

browse_stego_btn.place(x=20, y=350)
```

```
decode_stego_btn = Button(decode_window, text="Decode", width=15,
cursor="hand2",
command=lambda: decode_image(file_path, pass_to_decode.get(),
text_to_decode))
decode_stego_btn.config(font=("Open Sans", 15), bg="#36923B", fg="white",
borderwidth=0)
decode_stego_btn.place(x=592, y=420)
decode_opened = True
decode_window.protocol("WM_DELETE_WINDOW", lambda:
close_decode_window(decode_window))
```

```
def decode_image(image_path, password, text_field):
    decode_action = decode.Decode(image_path, password)
    msg = decode_action.are_values_valid()
    if not msg[1]:
        messagebox.showerror("Error Decoding", msg[0])
    else:
        decoded_text = decode_action.decode_from_image()
        if decoded_text[1]:
            text_field.config(state=NORMAL)
            text_field.delete(1.0, END)
            text_field.insert(1.0, decoded_text[0])
```

```
text_field.config(state=DISABLED)
```

```
messagebox.showinfo("Text Decoded", "Decode operation was successful.")
```

```
else:
```

```
messagebox.showerror("Error Decoding", decoded_text[0])
```

```
def browse_image(image_frame):
```

```
    global file_path
```

```
    file_path = filedialog.askopenfilename(title="Choose an Image",
```

```
                                           filetypes=(("Image Files", "*.png"), ("All Files", "*.*")))
```

```
    selected_image = Image.open(file_path)
```

```
    max_width = 350
```

```
    aspect_ratio = max_width / float(selected_image.size[0])
```

```
    max_height = int((float(selected_image.size[1]) * float(aspect_ratio)))
```

```
    selected_image = selected_image.resize((max_width, max_height), Image.LANCZOS)
```

```
    selected_image = ImageTk.PhotoImage(selected_image)
```

```
    image_frame.config(image=selected_image, height=304, width=354)
```

```
    image_frame.image = selected_image
```

```
def close_encode_window(encode_window):
```

```
    global encode_opened
```

```
    encode_window.destroy()
```

```
    encode_opened = False
```

```
def close_about_window(about_window):
```

```
    global about_opened
```

```
    about_window.destroy()
```

```
    about_opened = False
```

```
def close_decode_window(decode_window):
```

```
    global decode_opened
```

```
    decode_window.destroy()
```

```
    decode_opened = False
```

```
def about_menu():
```

```
    global about_opened
```

```
    if not about_opened:
```

```
        about_window = Toplevel(window)
```

```
        about_window.title("About")
```

```
        about_window.geometry('500x420')
```

```
        about_window.resizable(False, False)
```

```
        about_window.transient(window)
```

```
    if windows:
```

```
        windll.shcore.SetProcessDpiAwareness(1)
```

```
    text_to_decode_label = Label(about_window,
```

```
        text="Image Stenographer is an Image Steganography tool "
```

```

        "used to embed text messages into an Image. The "
        "embedded text is encrypted using the password."
        "\n\nTo encode the message into an image, click on Encode"
        " and select an Image. Choose a password and the text to
embed."

        " Then click Encode to embed and save your image with
message."

        "\n\nTo decode the message from an encoded image, click
Decode and select"
        " the encoded image, provide the password used to encode and
click Decode."

        " Your decoded message will be displayed on the window. If the
provided "

        "password is incorrect, message can never be extracted.")
    text_to_decode_label.config(font=("Open Sans", 12), justify="left", wraplength=450)
    text_to_decode_label.pack(padx=20, pady=20)
    about_opened = True
    about_window.protocol("WM_DELETE_WINDOW", lambda:
close_about_window(about_window))

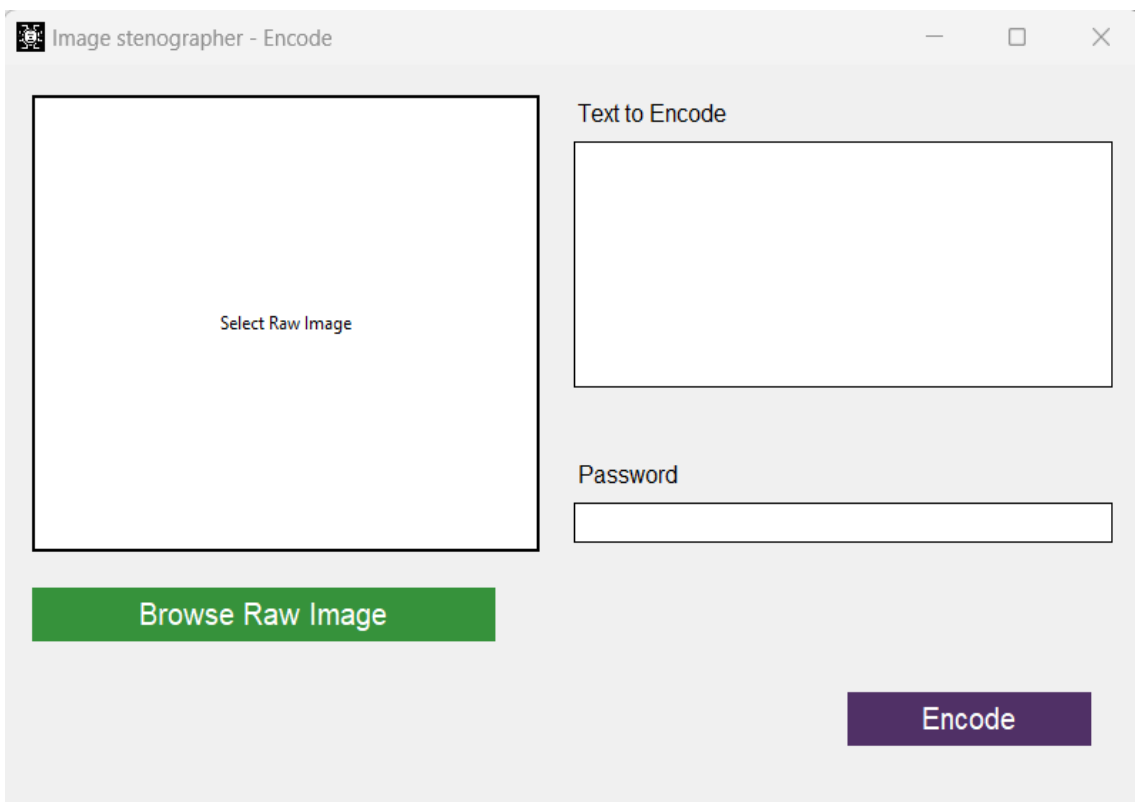
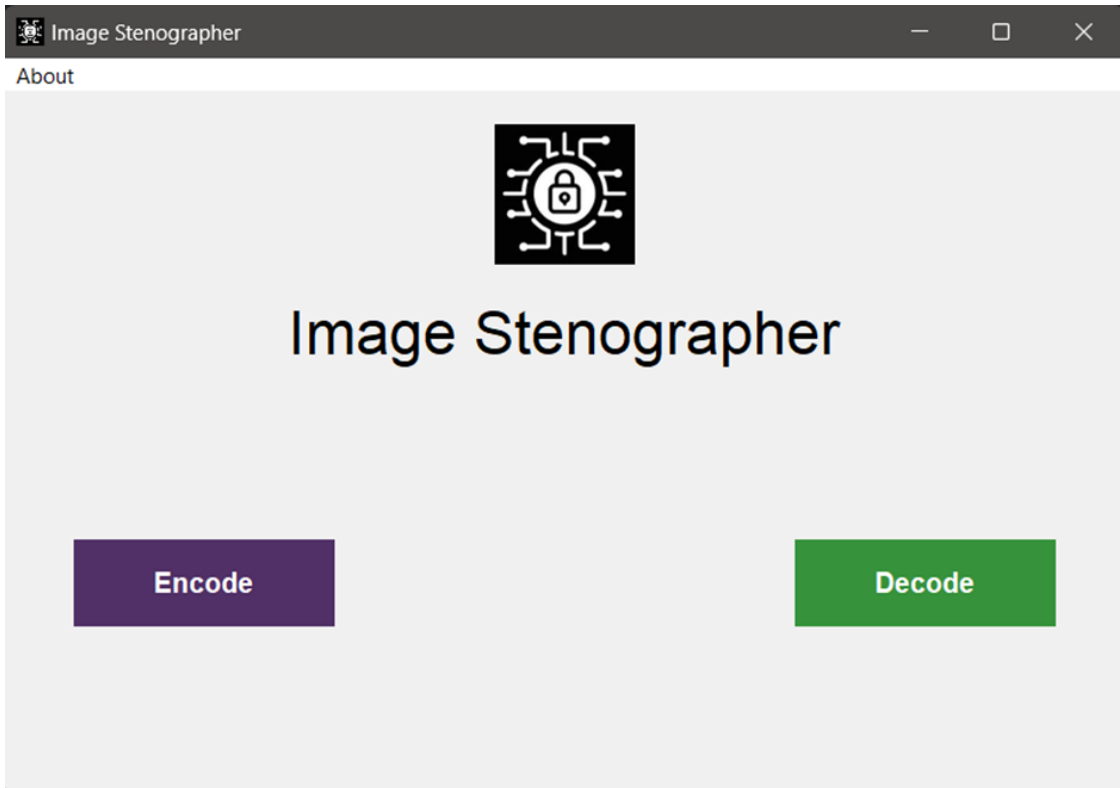
window = Tk()
window.title("Image Stenographer")
window.geometry('800x500')
window.resizable(True, True)
if windows:

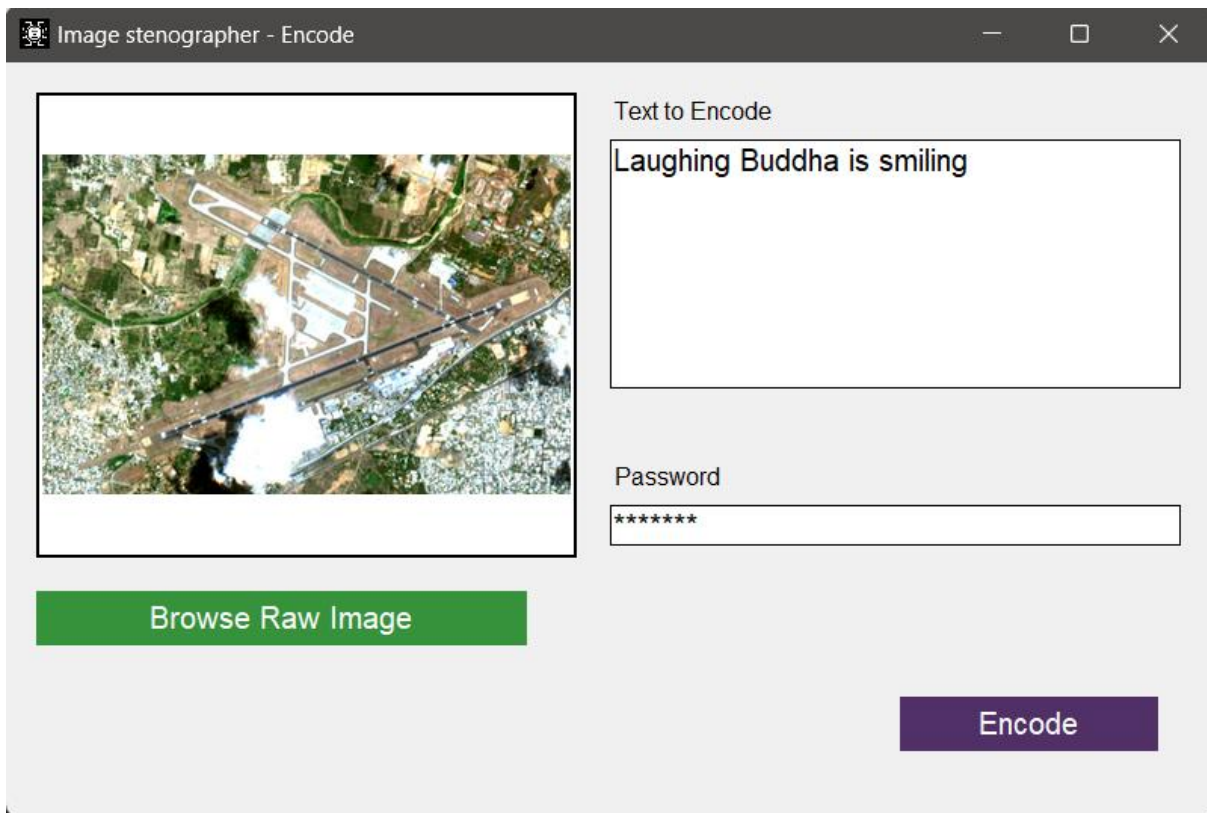
```

```
windll.shcore.SetProcessDpiAwareness(1)

logo = Image.open("src\images\clascryptlock.png")
logo = logo.resize((100, 100), Image.LANCZOS)
logo = ImageTk.PhotoImage(logo)
window.iconphoto(True, logo)
image_label = Label(window, image=logo, height=100, width=100)
image_label.pack(pady=20)
title_label = Label(window, text="Image Stenographer")
title_label.pack()
title_label.config(font=("Open Sans", 32))
encode_btn = Button(window, text="Encode", height=2, width=15, bg="#503066",
fg="white", cursor="hand2", borderwidth=0,
                    command=open_encode_window)
encode_btn.config(font=("Open Sans", 15, "bold"))
encode_btn.pack(side=LEFT, padx=50)
decode_btn = Button(window, text="Decode", height=2, width=15, bg="#36923B",
fg="white", cursor="hand2", borderwidth=0,
                    command=open_decode_window)
decode_btn.config(font=("Open Sans", 15, "bold"))
decode_btn.pack(side=RIGHT, padx=50)
menu = Menu(window)
menu.add_command(label="About", command=about_menu)
window.config(menu=menu)
window.mainloop()
```

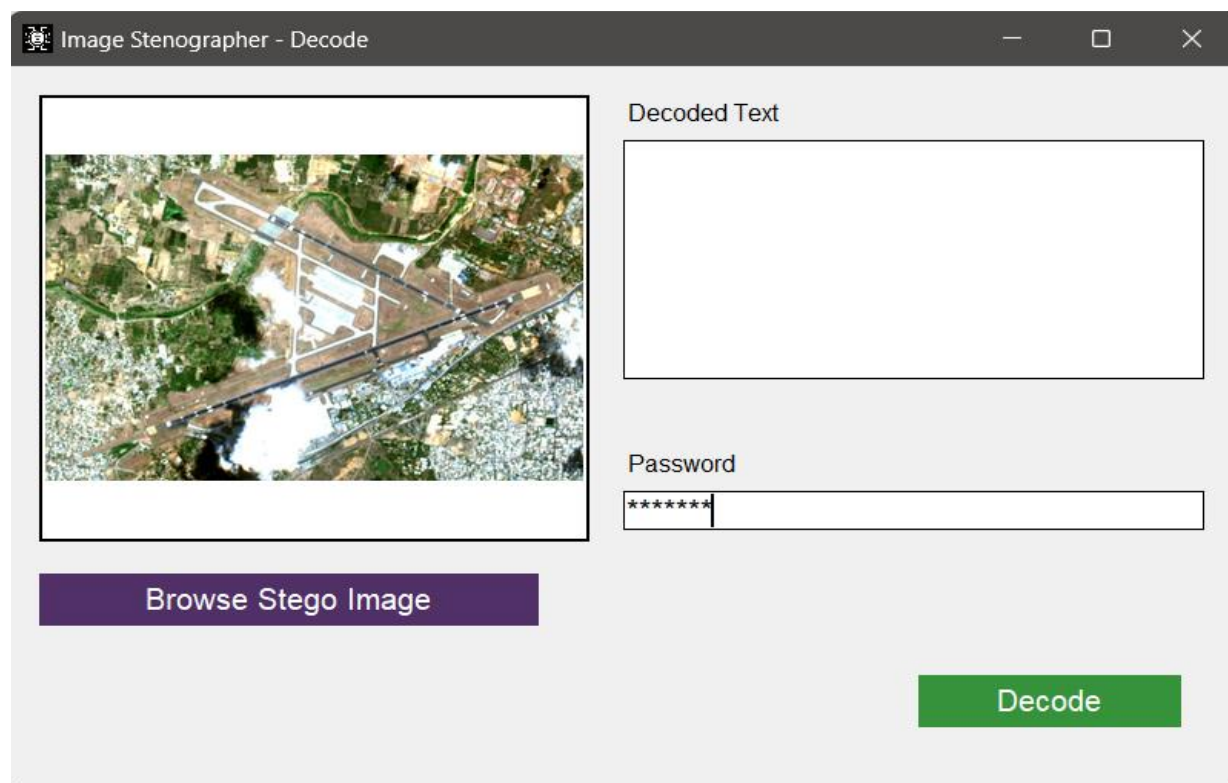
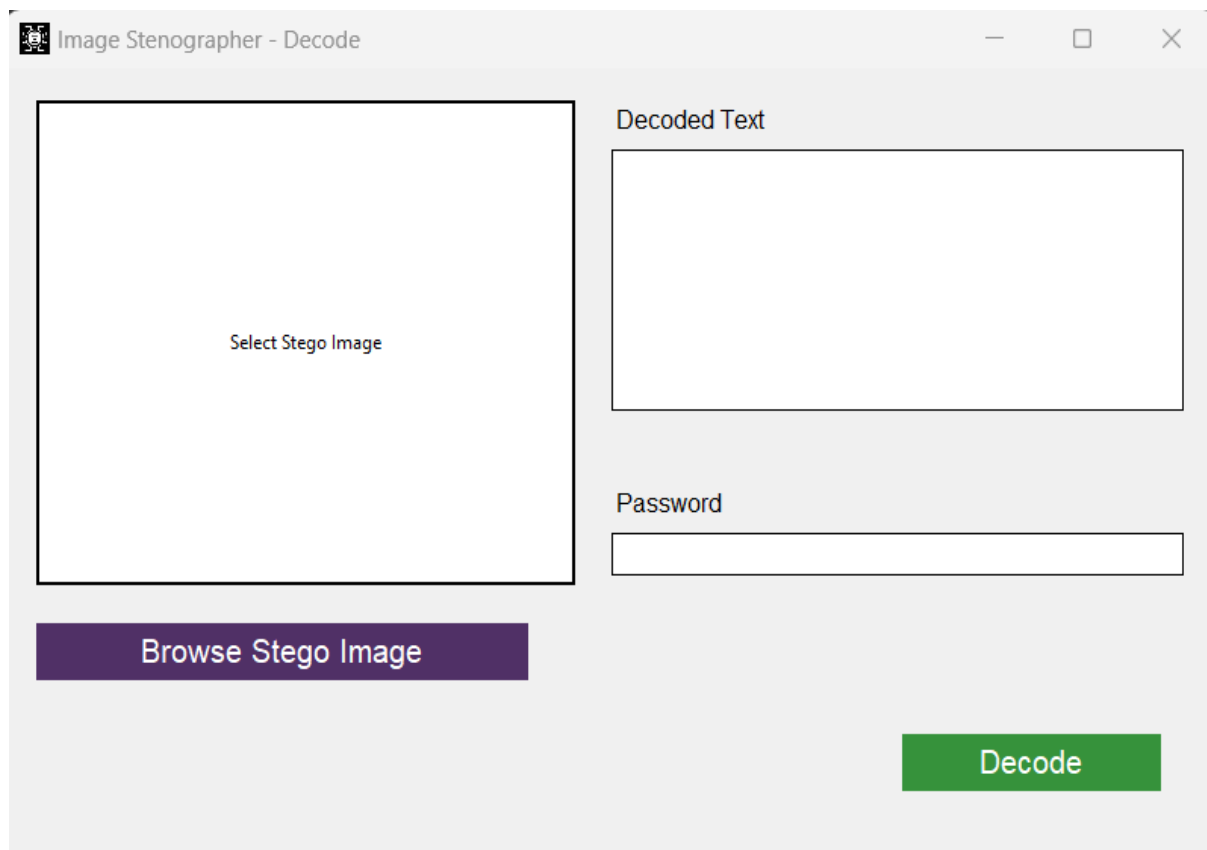
B. Screenshots:





Encode operation was successful.

OK





Text Decoded



Decode operation was successful.

OK



Image Stenographer - Decode



Decoded Text

Laughing Buddha is smiling

Password

Browse Stego Image

Decode

REFERENCES:

- <https://www.ijitee.org/wp-content/uploads/papers/v9i8/H6442069820.pdf>
- <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9748815/>
- <https://icons-forfree.com/gallery+image+landscape+mobile+museum+open+line+icon-1320183049020185924/>
- <https://stackoverflow.com/questions/273946/how-do-i-resize-an-image-using-pil-and-maintain-its-aspect-ratio>
- <https://stackoverflow.com/questions/57869364/how-to-browse-an-image-and-display-it-in-python>
- <https://www.programmingsought.com/article/85295249791/>
- <https://www.geeksforgeeks.org/python-convert-string-to-binary/>
- <https://stackoverflow.com/questions/40343414/how-to-convert-binary-string-to-byte-like-object-in-python-3>
- <https://medium.com/swlh/lsb-image-steganography-using-python-2bbb2c69a2>
- <https://stackoverflow.com/questions/58732272/how-to-convert-bytes-in-a-string-back-to-bytes>