A Project Report On

Image Steganography

For
Mini Project 2B (REV- 2019 'C' Scheme) of Third Year, (TE Sem-VI)

Course code: CSM 501

in

Computer Science and Engineering (AI&ML)

By

Hamdule Zohaib Yusuf (R-20-0056)
Kambli Sujan Shridhar (R-20-0147)
Chavan Raj Shivaram (R-20-0113)
Salvi Raj Rajendra (R-20-0189)

Under the guidance of
Prof. Sprooha Athalye

Hope Foundation's

Finolex Academy of Management & Technology, Ratnagiri

Academic Year 2022-23

CERTIFICATE

This is to certify that the project entitled **Image Steganography** is a bonafide work of

Hamdule Zohaib Yusuf	**R 20 0056**
Kambli Sujan Shridhar	**R 20 0147**
Chavan Raj Shivaram	**R 20 0113**
Salvi Raj Rajendra	**R 20 0189**

Submitted to the University of Mumbai in partial fulfillment of the requirement for the award of **Mini Project 2B (REV- 2019 'C' Scheme) of Third Year, (TE Sem-VI)** in **Computer Science and Engineering(AI&ML)** as laid down by **University of Mumbai** during academic year **2022-23**

(_____)
Examiner/Reviewer-1

(_____)
Examiner/ Reviewer -2

Prof. Sprooha Athalye
Guide

Prof. V. V. Nimbalkar
Head of Department

INDEX

List of Figures

1. INTRODUCTION

1.1 What is Steganography?

Steganography is the art and science of concealing messages or information within other seemingly harmless messages or data, such as an image, audio, or video file. The goal of steganography is to hide the presence of the message in such a way that it is not detectable to unintended recipients. The use of steganography can be traced back to ancient times, where people used hidden symbols, codes, and ciphers to communicate secretly.

An example of this can be seen in the following sentence:
Some elephants never do have enough lettuce, perhaps.

Reading the first letter of each word we get the message: *send help*

In modern times, steganography has gained immense popularity due to the rise of digital media and the internet. It offers a unique advantage over encryption in that it does not alert anyone that a message exists in the first place. By embedding a message within a larger file, a sender can communicate undetected with a recipient, bypassing censorship or surveillance.

1.2 What is Steganalysis?

Steganalysis is the process of detecting and extracting hidden information in digital media using specialized tools and algorithms. It is the countermeasure to steganography and is used by law enforcement officials, intelligence agencies, and security researchers to identify and uncover hidden messages.

1.3 Image Steganography

Image steganography is the practice of hiding information within digital images without causing any visible changes to the image itself. This is achieved by slightly modifying the least significant bits of the image pixels, which are not easily perceivable to human vision. The hidden message can be anything from a small text message to a large multimedia file, and can be hidden in any part of the image, such as in the colour channels or in the image's metadata.

1.4 Applications of Image Steganography

Listed below are several interesting applications of image steganography.

- Image steganography can be used for watermarking images [1] with copyright information, enabling creators to protect their intellectual property.

- It can be used for digital rights management, allowing content creators to embed information about authorized access to the image or its content.

- Image steganography can be used for covert communication, such as in espionage or other forms of clandestine activities.

- It can also be used for authentication and data hiding in sensitive images, such as medical images or forensic evidence.

- It can be used to share data through platforms that do not allow sharing of certain type of data. By embedding this data in images and sharing the encoded images.

2. LITERATURE SURVEY

Authors T. Morkel , J.H.P. Eloff and M.S. Olivier explain the concept of steganography and how it involves hiding information within other data to conceal communication [2]. The authors notes that digital images are commonly used as carrier files for steganography due to their prevalence on the internet. The authors also highlights the existence of various steganographic techniques, each with its own strengths and weaknesses, and how different applications may require specific steganography techniques based on their requirements. Additionally, the authors aims to provide an overview of image steganography and identify the characteristics of a good steganographic algorithm.

The authors Chandramouli and Nasir Memon discuss the issue of perceptually indiscernible message hiding in images and whether they result in statistically indistinguishable images from untampered ones [3]. They introduce specific image-based steganography techniques and demonstrate that an observer can differentiate between images that carry hidden messages and those that do not. The authors derive a formula for the probability of detection and false alarm in terms of the number of hidden bits, leading them to the concept of steganographic capacity. This concept measures the maximum number of bits that can be hidden in an image without causing statistically significant modifications. The authors' ongoing work focuses on adaptive steganographic techniques that aim to increase the number of bits that can be embedded significantly by foiling detection mechanisms.

K Thangadurai and G Sudha Devi discusses the concept of steganography, which involves hiding information in digital files like images, videos, or audio. The hidden information is concealed in such a way that it is not perceptible to an ordinary viewer. The focus is on image steganography, which involves hiding information in cover images. The Least Significant Bit (LSB) algorithm is a popular technique for embedding information in a cover file [4]. The paper aims to provide detailed knowledge about LSB-based image steganography and its applications to various file formats. The paragraph also highlights the importance of combining steganography with cryptography to enhance security.

3. PROBLEM STATEMENT

The problem that this project aims to address is the need for secure and efficient communication of sensitive information over digital channels. While encryption can be used to secure the data, it does not provide complete security as an attacker may still be able to identify that a message is being transmitted. Therefore, steganography can be used to hide the message in plain sight by embedding it in a carrier image. However, the challenge with steganography is to achieve a balance between imperceptibility and the amount of data that can be hidden in the carrier image. Additionally, the steganographic technique must be robust enough to resist various attacks that an attacker may use to detect the hidden information. Therefore, this project aims to develop an efficient and secure image steganography technique that uses encryption to protect the hidden message while maintaining imperceptibility and robustness against various attacks.

4. PROJECT WORKING

A simple image steganography program typically follows the following workflow:

- Load cover image: The program loads the image file that will be used as a cover for the hidden message.

- Load secret message: The program loads the secret message that will be hidden within the cover image. This message can be in any format, such as a text file or an image file.

- Encode secret message: The program uses a steganographic technique to encode the secret message into the cover image. This technique can be as simple as least significant bit (LSB) insertion, where the message bits are inserted into the least significant bits of the cover image pixels.

- Save *stego* image: The program saves the new image file that contains the hidden message.

- Decode secret message: The program can optionally provide a decoding function to extract the secret message from the stego image. This function should use the same steganographic technique as the encoding function to extract the message bits from the image pixels.

- Decrypt secret message: If the message was encrypted before being hidden, the program needs to decrypt it using the appropriate decryption algorithm and secret key.

- Display secret message: The program displays the extracted and decrypted secret message to the user.

The following figures 4.1 and 4.2 illustrate this workflow

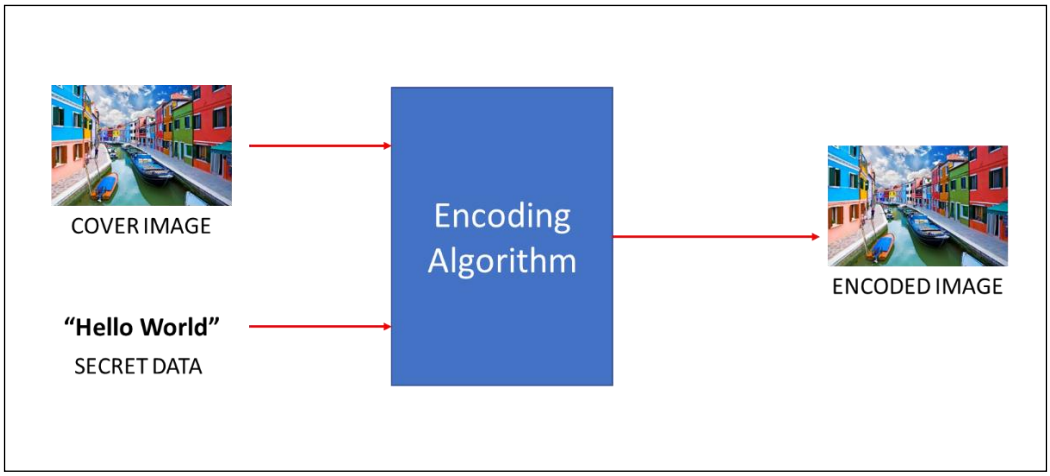

Fig 4.1 Encoding Process

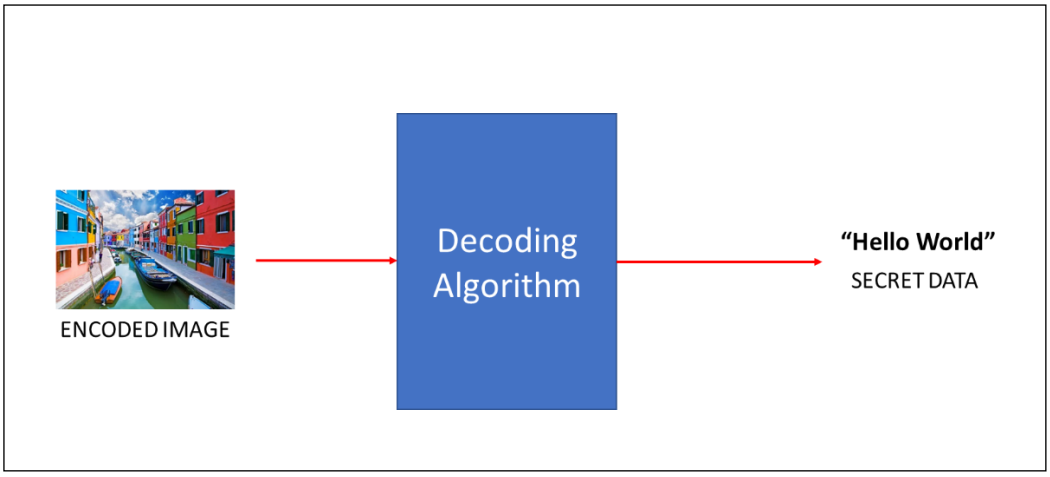

Fig 4.2 Decoding Process

5. LEAST SIGNIFICANT BIT METHOD

5.1 Representation of an Image in the memory

In a computer an image is represented as a matrix of pixels, where each pixel has a red, green and blue component. These components are represented using 8-bit numbers that range from 0 to 255. These values represent the intensity of each component. The following example illustrates an image stored as the matrix of intensity values of RGB components of each pixel.

```
[ 22, 156,  40] [198,  16,  34] [224,  14, 110] …
[118,  62,  74] [146, 231, 242] [ 75,  94, 247] …
[145, 208, 114] [206,  79,  80] [ 39, 251, 250] …
```

This matrix represented in binary would be

[00010110, 10011100, 00101000] [11000110, 00010000, 00100010] [11100000, 00001110, 01101110] …
[01110110, 00111110, 01001010] [10010010, 11100111, 11110010] [01001011, 01011110, 11110111] …
[10010001, 11010000, 01110010] [11001110, 01001111, 01010000] [00100111, 11111011, 11111010] …

5.2 Encoding data

The data to be encoded is also represented as binary bits. If the data to be encoded is a text message, the individual characters in the message can be represented using their ASCII values and then converted to their binary counterpart. For example: The message "*hello*" can be represented as:

```
01101000 01100101 01101100 01101100 01101111
```

The least significant bit of each component of each pixel is modified to encode this data. This modification has negligible effect on the appearance of the image as it only changes the intensity values by a single digit.

[0001011**0**, 1001110**1**, 0010100**1**] [1100011**0**, 0001000**1**, 0010000**0**] [1110000**0**, 0000111**0**, 0110111**0**] …
[01110110, 00111110, 01001010] [10010010, 11100111, 11110010] [01001011, 01011110, 11110111] …
[10010001, 11010000, 01110010] [11001110, 01001111, 01010000] [00100111, 11111011, 11111010] …

6. ENCODING FILES INTO IMAGES

6.1 Encoded Data Header

Files can be encoded into images same way a text message is encoded. Files are read byte by byte and the LSB of each component of each pixel the image is modified to accommodate the bits from the file. However, the program faces a problem when decoding the image as it is kept guessing whether the data encoded is a file or a text message, if it is a file then what type of file is it. It also has to know in advanced whether the encoded data is encrypted or compressed, etc. All this information is stored within a header which is also encoded within the image. See Figure 6.1.

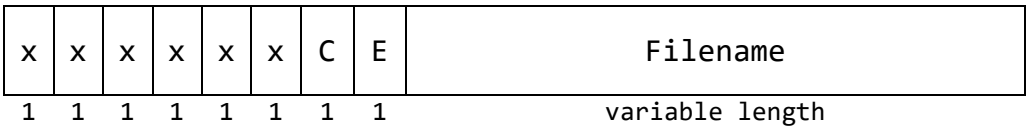

Fig 6.1 Encoded Data Header

6.2 Bit Stuffing and Delimiter

The program needs to know when to stop decoding the pixels of the image. That is when the end of the message / data is reached. This is done by adding a delimiter to the end of the data before encoding it. But to ensure the delimiter does not already exist in the data Bit Stuffing is carried out. Bit stuffing is the process of adding a 0 after very certain number of 1s in the binary. For example, let us consider the word "*combo*" represented in binary

`1100011 1101111 1101101 1100010 1101111`

After bit stuffing (adding a 0 after every 3 consecutive 1s)
`11000111010111011100110111000010110111001`

Adding the delimiter
`1100011101011101110011011100001011011101011111`

Note that the header will also undergo bit stuffing and a delimiter will be added to the header. This will then be concatenated with the bit stuffed and delimited data. And this will simply be encoded in the image using LSB method.

Stuffed Header	Delimiter	Stuffed Data	Delimiter

Fig 6.2 Data to be encoded

6.3 Decoding the Data

The decoder will simple keep reading the LSB of each component of each pixel until it finds the delimiter. It will then unstuff the data to acquire the header. It will inspect this header to resolve the encoding mode and options. After this the decoder keeps reading the LSB of each component of each pixel until it finds the delimiter again. It then unstuffs the bits collected to get the encoded data. Now this data can be encrypted or compressed which is dealt with in the next chapter.

Unstuffing the data (removing a 0 after every 3 consecutive 1s)
`110001110101110111001101110000101101110 1→`
`1000111101111110110111000101101111`

Note that the delimiter is chosen arbitrarily here. In the actual program the delimiter is of 8 bits: `01111111`. Therefore, during bit stuffing a 0 is added after every 6 consecutive 1s in the binary data to remove the possibility of the delimiter appearing in the data.

7. DYNAMICALLY SCALING IMAGES

7.1 Need to Scale Images

Since we only modify the LSB the amount of data that can be encoded in a image is less than $1/8^{th}$ the size of the image. Thus, the program has to enlarge the image in order to fit the entire data. In order to do this, we need to enlarge the image by the least amount possible such that it just fits the data. To find the minimum scaling factor required we take help of some basic algebra.

7.2 Data Length and Total Encoding Limit

Let *dataLength* be the number of bits to be encoded in the image.
Let total number of bits that can be encoded in an image be the *totalEncodingLimit*. It is given by:
*totalEncodingLimit = 3 * w * h*
where $w \rightarrow$ width of the image in pixels, and
$h \rightarrow$ height of the image in pixels
From this we can calculate the number of extra bits required to accommodate the data as
newBitsRequired = dataLength – totalEncodingLimit

7.1 Scaling Factor and New Bits Added

Let *scalingFactor* be the factor by which the height and width of the image will be multiplied to get the new dimensions of the image.
The new size of the image (no. of pixels) will be
$= (scalingFactor * w)\ (scalingFactor * h)$
$= scalingFacotr^2 * w * h$

The number of pixels added will be: new size - original size
newPixelsAdded $= (scalingFacotr^2 * w * h) – (w * h)$
$= w * h * (scalingFactor^2 - 1)$

The new bits added to the encoding limit therefore will be,
$newBitsAdded = 3 * w * h * (scalingFactor^2 - 1)$

We know that *newBitsRequired* should be equal to *newBitsAdded*, therefore,
$3 * w * h * (scalingFactor^2 - 1) = dataLength – totalEncodingLimit$

Simplifying this we get the equation for scaling factor,
scalingFactor $= \sqrt{}$ ***(dataLength / totalEncodingLimit)***

We use this equation to find the scaling factor and resize the image such that it just fits all the data to be encoded in it.

8. ENCRYPTION

8.1 What is Encryption?

Encryption is the process of converting plain, readable text or data into an encoded or encrypted format, using algorithms or mathematical formulas, to protect it from unauthorized access or use. The encrypted data can only be decrypted and understood by authorized parties who possess the correct decryption key or password. Encryption is widely used to secure sensitive information in communication channels, online transactions, and data storage, to ensure privacy, confidentiality, and integrity. Strong encryption methods, such as Advanced Encryption Standard (AES), are essential to safeguard sensitive information against hacking, identity theft, and cyberattacks.
Here we encrypt the data before encoding it into the image. Therefore, even if an intended recipient is able to decode the data they will only receive garbled / encrypted data.

8.2 Advanced Encryption Standard (AES)

AES (Advanced Encryption Standard) is a widely-used encryption algorithm that provides high-level security for sensitive data. AES is a symmetric key encryption algorithm, meaning that the same key is used for both encryption and decryption of data. It uses block cipher cryptography, where the plaintext is divided into blocks and each block is encrypted separately. AES comes in different key sizes, including 128, 192, and 256 bits, with 256-bit being the strongest and most secure.
In this project we are using AES 256 encryption. If a password (of any length) is provided, its SHA 256 digest of 32 bytes is used as the key for the AES encryption. Encryption odes not take place is password is not provided and the data is encoded as it is.

8.3 Validation of Password

Since the hash of the password is not stored in the header, there is no way of telling if the password provided during the decoding is correct. Providing the wrong password, the decryption may still take place but the data output will be corrupted. To tackle this, a small number of fixed characters is added to the beginning of the data before it is encrypted. When the decoding and decrypting is complete the program checks integrity of these characters. If these characters are intact the password entered is correct. Otherwise, the user is prompted to enter the password again.

9. IMPLEMENTATION IN PYTHON

9.1 Python 3

Python 3 is a popular high-level programming language used for web development, data analysis, scientific computing, and artificial intelligence. It is known for its readability and easy-to-learn syntax, making it an ideal language for beginners and experienced developers alike.

9.2 OpenCV

OpenCV (Open-Source Computer Vision) is a powerful open-source library that provides tools and algorithms for image and video processing. It includes a vast collection of functions for image processing, such as image filtering, feature detection, object recognition, and motion analysis. OpenCV is widely used in computer vision research, robotics, and video surveillance applications.

9.3 PyCryptodome

PyCryptodome is a collection of cryptographic libraries for Python that provides secure encryption and decryption methods for data protection. It supports various encryption algorithms, including AES, DES, Blowfish, and RSA, and provides cryptographic hash functions for data integrity checks.

9.4 NumPy

NumPy is a popular Python library used for scientific computing and data analysis. It provides powerful tools for performing numerical calculations, such as array manipulation, mathematical functions, and linear algebra operations. NumPy arrays are used extensively in data analysis, machine learning, and scientific computing applications for their efficient memory management and fast computation capabilities**.**

10. SOURCE CODE

10.1 imgstegano.py

```
import tkinter as tk
from tkinter import ttk, filedialog
import tkinter.simpledialog as sd
import tkinter.messagebox as mb
from PIL import Image, ImageTk
import cv2
import numpy as np
import os
import math
from aes_encryption import AESCipher

integrityCheckString = 'abcde'
# This is used to check the integrity of the file after it is decrypted
# Used to identify if the password entered was correct / incorrect


def dataToBinary(data):
    'Converts string, bytes, NumPy array of Integers to Binary or list of bytes
represented using strings'
    if type(data) == str:
        p = ''.join([format(ord(i), '08b')for i in data])
    elif type(data) == bytes or type(data) == np.ndarray:
        p = [format(i, '08b')for i in data]
    return p


def stuffBits(binaryData):
    'Adds a 0 after every 6 consecutive 1s (Bit Stuffing). And adds delimiter at the
end of binary string data'
    bitCount = 0
    bitLength = len(binaryData)
    bitIndex = 0

    while (bitIndex < bitLength - 1):
        if binaryData[bitIndex] == '1':
            bitCount += 1

        if binaryData[bitIndex] == '0':
            bitCount = 0

        if bitCount == 6:
            bitCount = 0
            bitIndex += 1
            binaryData = binaryData[:bitIndex] + '0' + binaryData[bitIndex:]
            bitLength += 1

        bitIndex += 1
```

```
    binaryData += '01111111'
    return binaryData


def unstuffBits(binaryData):
    'Unstuffs bits from the biinary string and removes the delimiter'
    binaryData = binaryData[:-8]

    bitCount = 0
    bitLength = len(binaryData)
    bitIndex = 0

    while (bitIndex < bitLength - 1):
        if binaryData[bitIndex] == '1':
            bitCount += 1

        if binaryData[bitIndex] == '0':
            bitCount = 0

        if bitCount == 6:
            bitCount = 0
            binaryData = binaryData[:bitIndex+1] + binaryData[bitIndex+2:]
            bitLength -= 1

        bitIndex += 1

    return binaryData


def encode(fileMode, encodeImageFilePath, encodeData, encodedOutputPath,
encodeCompress, dynamicRescaling, encodePassword):
    'Function implements LSB method to encode data into images'
    global integrityCheckString

    # Generate the Header
    binaryHeader = '000000'

    if encodeCompress:
        binaryHeader += '1'
    else:
        binaryHeader += '0'

    if encodePassword:
        binaryHeader += '1'
    else:
        binaryHeader += '0'


    binaryData = ''
```

```
    if fileMode:
        # Add filename to the header
        binaryHeader += dataToBinary(os.path.basename(encodeData))

        with open(encodeData, 'rb') as file:
            content = file.read()

            # Encryption on bytes read
            if encodePassword:

                # If password is provided (ecryption is to be applied)
                encodedByteList = []
                for byte in content:
                    # append is faster in python list than in numpy arrays (or += in
strings)
                    encodedByteList.append(chr(int(byte)))

                encodedbyteString = integrityCheckString + ''.join(encodedByteList)

                cipher = AESCipher(encodePassword)
                encryptedbyteString = cipher.encrypt(encodedbyteString)

                binaryData = dataToBinary(encryptedbyteString)

            else:
                # No encryption
                byteStringList = dataToBinary(content)
                binaryData = ''.join(byteStringList)



    else:

        if encodePassword:
            # Encryption on text
            cipher = AESCipher(encodePassword)
            encodeData = integrityCheckString + encodeData
            encodeData = cipher.encrypt(encodeData)

        binaryData = dataToBinary(encodeData)



    # Stuffing hedaer and data seperately. Each will have a seperate delimiter
    binaryHeader = stuffBits(binaryHeader)
    binaryData = stuffBits(binaryData)
    binaryData = binaryHeader + binaryData


    '''
    We did not go with numpy array of boolean values for binaryData since append is
```

```
very slow in numpy
    and also since we are not doing any operations on the list.
    We did not go with list of True / False values as it will not have a significant
increase in performance
    and will make the program much harder to read and interpret.
    '''


    image = cv2.imread(encodeImageFilePath)

    # Dyanmic Rescaling to store data
    dataLength = len(binaryData)
    imageBitLength = image.shape[0] * image.shape[1] * 3


    if dataLength >= imageBitLength:
        if dynamicRescaling:
            scalingFactor = math.sqrt(dataLength/imageBitLength)

            height = math.ceil(image.shape[0] * scalingFactor)
            width = math.ceil(image.shape[1] * scalingFactor)

            dim = (width, height)

            image = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
        else:
            mb.showinfo('Dynamic Rescaling', 'The data to be encoded is too large.
Enable Dynamic Rescaling to scale the image to fit the data.')
            return -1


    dataIndex = 0

    # Modifying the Least Significant Bit of each component of each pixel to store
bits of Binary Data
    for row in image:
        for pixel in row:

            if dataIndex >= dataLength:
                break
            if dataIndex < dataLength:
                pixel[0] = int(pixel[0] & 0b11111110) + int(binaryData[dataIndex])
                dataIndex += 1
            if dataIndex < dataLength:
                pixel[1] = int(pixel[1] & 0b11111110) + int(binaryData[dataIndex])
                dataIndex += 1
            if dataIndex < dataLength:
                pixel[2] = int(pixel[2] & 0b11111110) + int(binaryData[dataIndex])
                dataIndex += 1
```

```
    cv2.imwrite(encodedOutputPath, image)

    mb.showinfo('Encoding complete', f'Output generated: {encodedOutputPath}')



def decode(decodeImageFilePath, decodeDestination=''):
    global integrityCheckString

    compressed = False
    protected = False

    delimsFound = 0
    bitCount = 0

    binaryHeader = ''
    binaryData = ''

    readBitsList = []

    image = cv2.imread(decodeImageFilePath)


    # Reading the Least Significant Bit of each component of each pixel
    for row in image:

        # The reading stops when both the delimiters (header and data) are found
        if delimsFound == 2:
            break

        for pixel in row:
            b = pixel[0] & 1
            g = pixel[1] & 1
            r = pixel[2] & 1

            readBitsList.append(b)

            if b:
                bitCount += 1

                if bitCount == 7:
                    delimsFound += 1

                    # When the first delimiter is found, the data read is stored in
binaryHeader
                    if delimsFound == 1:
                        binaryHeader = ''.join(map(str, readBitsList))
                        readBitsList = []

                    if delimsFound == 2:
```

```
                    break

            else:
                bitCount = 0


            readBitsList.append(g)

            if g:
                bitCount += 1

                if bitCount == 7:
                    delimsFound += 1

                    if delimsFound == 1:
                        binaryHeader = ''.join(map(str, readBitsList))
                        readBitsList = []

                    if delimsFound == 2:
                        break

            else:
                bitCount = 0


            readBitsList.append(r)

            if r:
                bitCount += 1

                if bitCount == 7:
                    delimsFound += 1

                    if delimsFound == 1:
                        binaryHeader = ''.join(map(str, readBitsList))
                        readBitsList = []

                    if delimsFound == 2:
                        break

            else:
                bitCount = 0



binaryData = ''.join(map(str, readBitsList))
binaryHeader = unstuffBits(binaryHeader)

# Header is read to get the options
if binaryHeader[6] == '1':
```

```
    compressed = True
if binaryHeader[7] == '1':
    protected = True

binaryHeader = binaryHeader[8:]

byteHeader = [binaryHeader[i: i + 8] for i in range(0, len(binaryHeader), 8)]

# Filename is extracted from the header
fileName = ""
for x in byteHeader:
    fileName += chr(int(x, 2))

binaryData = unstuffBits(binaryData)

byteStringList = [binaryData[i: i + 8] for i in range(0, len(binaryData), 8)]

if fileName:
    # File Mode (encoded data is a file)
    byteData = []

    if protected:
        # If data is password protected (to be decrypted)
        encryptedDataList = []
        for byte in byteStringList:
            encryptedDataList.append(chr(int(byte, 2)))

        encryptedDataString = ''.join(encryptedDataList)

        password = sd.askstring("Password", "Enter Password")
        cipher = AESCipher(password)

        try:
            decryptedDataString = cipher.decrypt(encryptedDataString)
        except UnicodeDecodeError:
            mb.showerror("Error", "Incorrect Password!")
            return -1

        # try except blocks do not introduce a new scope

        if decryptedDataString[0:5] != integrityCheckString:
            mb.showerror("Error", "Incorrect Password!")
            return -1

        else:
            decryptedDataString = decryptedDataString[5:]
            for char in decryptedDataString:
                byteData.append(ord(char))
```

```
        else:
            # No decryption
            for byte in byteStringList:
                byteData.append(int(byte, 2))

        byteData = bytearray(byteData)

        fullPath = os.path.join(decodeDestination, fileName)
        with open( fullPath , 'wb') as file:
            file.write(byteData)

        mb.showinfo('Decoding Complete', 'Output generated at: ' + fullPath)


    else:
        # Text Mode (encoded data is a message)
        if protected:
            # Password protected (decryption)
            encryptedDataList = []
            for x in byteStringList:
                encryptedDataList.append(chr(int(x, 2)))
            encryptedData = ''.join(encryptedDataList)

            password = sd.askstring("Password", "Enter Password")
            cipher = AESCipher(password)

            try:
                decryptedData = cipher.decrypt(encryptedData)
            except UnicodeDecodeError:
                mb.showerror("Error", "Incorrect Password!")
                return -1

            if decryptedData[0:5] != integrityCheckString:
                mb.showerror("Error", "Incorrect Password!")
                return -1

            else:
                decryptedData = decryptedData[5:]
                decodedTextBox.insert('1.0', decryptedData)


        else:
            # No password protection
            dataList = []
            for x in byteStringList:
                dataList.append(chr(int(x, 2)))
            data = ''.join(dataList)
            decodedTextBox.insert('1.0', data)


        decodedTextBox.pack(pady=5, padx=10)
```

```
'''
//////////////////
GUI Code //////////
//////////////////
'''



window = tk.Tk()
window.geometry("600x800")
window_height = 800

window.title("Image Encoder/Decoder")

# Global variables to store File Paths
encodeImageFilePath = ''
decodeImageFilePath = ''
encodeData = ''
decodeDestination = ''



def chooseImageEncode():
    'Handles chosing image to be encoded'

    global encodeImageFilePath

    filetypes = [("PNG files", "*.png"), ("All files", "*.*")]
    encodeImageFilePath = filedialog.askopenfilename(filetypes=filetypes)

    if encodeImageFilePath:
        # Scales the image to have 40% of window height
        image = Image.open(encodeImageFilePath)

        width, height = image.size
        maxHeight = int(window_height * 0.4)
        if height > maxHeight:
            width = int(width * (maxHeight / height))
            height = maxHeight

        image = image.resize((width, height))

        imageEncode = ImageTk.PhotoImage(image)
        imageBoxEncode.configure(image=imageEncode)
        imageBoxEncode.image = imageEncode # Setting a reference
        chooseImageButtonEncode.configure(text="Choose another image")
```

```
def chooseEncodeFile():
    'Handles chossing a file to be encoded in the selected image'

    global encodeData

    filetypes = [("All files", "*.*")]
    encodeData = filedialog.askopenfilename(filetypes=filetypes)

    if encodeData:
        fileLabel.configure(text=encodeData)


def outputDestinationChange():
    'Handles change in destination of output files generated after decoding'

    global decodeDestination

    decodeDestination = filedialog.askdirectory()

    if decodeDestination:
        destinationLabel.configure(text=f'Output File Destination:
{decodeDestination}')


def chooseImageDecode():
    'Handles choosing the image to be decoded'
    global decodeImageFilePath

    filetypes = [("PNG files", "*.png"), ("All files", "*.*")]
    decodeImageFilePath = filedialog.askopenfilename(filetypes=filetypes)

    if decodeImageFilePath:
        # Scales the image to fit in 40% of window height
        image = Image.open(decodeImageFilePath)

        width, height = image.size
        maxHeight = int(window_height * 0.4)
        if height > maxHeight:
            width = int(width * (maxHeight / height))
            height = maxHeight

        image = image.resize((width, height))

        imageDecode = ImageTk.PhotoImage(image)
        imageBoxDecode.configure(image=imageDecode)
        imageBoxEncode.image = imageDecode # Setting a reference
        chooseImageButtonDecode.configure(text="Choose another image")
```

```
def onEncode():
    'Validates input and calls the encode method when the user clicks on the Encode
button'

    global encodeImageFilePath, encodeData

    # Encode Parameters
    fileMode = False

    if not encodeImageFilePath:
        mb.showinfo('Select an Image', 'Select an image to encode data into')

    if encodeSubtabs.tab(encodeSubtabs.select(), 'text') == 'File':
        fileMode = True
        if not encodeData:
            mb.showinfo('No File Selected', 'Please select a file to encode in the
image')
            return -1
    else:
        encodeData = textBox.get('1.0', 'end-1c')
        if not encodeData:
            mb.showinfo('Empty Message', 'Please enter a message to encode')

    encodedOutputPath = outputEntry.get()
    encodeCompress = compressionVar.get()
    dynamicRescaling = rescalingVar.get()

    encodePassword = passwordEntry.get()

    # Hide the Encode button and display "Encoding..." label
    encodeButton.pack_forget()
    encodingLabel.pack(pady=10)

    encode(fileMode, encodeImageFilePath, encodeData, encodedOutputPath,
encodeCompress, dynamicRescaling, encodePassword)

    # Unhide the Encode button
    encodeButton.pack(pady=10)
    encodingLabel.pack_forget()


def onDecode():
    'Calls the decode method when the user clicks on Decode button'

    global decodeImageFilePath, decodeDestination

    if not decodeImageFilePath:
        mb.showinfo('Select an Image', 'Select an image to decode')
        return -1
```

```
    decodedTextBox.pack_forget()

    # Hide the Decode button and display "Decoding..." label
    decodeButton.pack_forget()
    decodingLabel.pack(pady=10)

    decode(decodeImageFilePath, decodeDestination)

    # Make the Decode button visible again
    decodingLabel.pack_forget()
    decodeButton.pack(pady=5)


# Create tabbed group with 2 tabs
tabs = ttk.Notebook(window)
tabs.pack(fill='both', expand=True)
tabs.enable_traversal()

# Create encode tab
encodeTab = ttk.Frame(tabs)
tabs.add(encodeTab, text="Encode")

imageBoxEncode = tk.Label(encodeTab)
imageBoxEncode.pack()

# Create choose image button and image box
chooseImageButtonEncode = tk.Button(encodeTab, text="Choose an Image",
command=chooseImageEncode)
chooseImageButtonEncode.pack(pady=10)

# Create tabbed group with 2 tabs (File and Text)
encodeSubtabs = ttk.Notebook(encodeTab, height=100)
encodeSubtabs.pack()

# Create Text tab
textTab = ttk.Frame(encodeSubtabs)
encodeSubtabs.add(textTab, text="Text")

textBox = tk.Text(textTab)
textBox.pack(pady=10, padx=20)

# Create File tab
fileTab = ttk.Frame(encodeSubtabs)
encodeSubtabs.add(fileTab, text="File")

fileLabel = tk.Label(fileTab, text="No file chosen")
fileLabel.pack(pady=10)

browseButton = tk.Button(fileTab, text="Browse", command=chooseEncodeFile)
browseButton.pack(pady=10)
```

```
# Create password entry box
passwordFrame = tk.Frame(encodeTab)
passwordFrame.pack(side=tk.TOP, padx=5, pady=5)

passwordLabel = tk.Label(passwordFrame, text="Password:")
passwordLabel.pack(side=tk.LEFT)

passwordEntry = tk.Entry(passwordFrame, width=30)
passwordEntry.pack(side=tk.LEFT)


# Create output path entry
outputFrame = tk.Frame(encodeTab)
outputFrame.pack(side=tk.TOP, padx=5, pady=5)

outputLabel = tk.Label(outputFrame, text="Output:")
outputLabel.pack(side=tk.LEFT)

outputEntry = tk.Entry(outputFrame, width=30)
outputEntry.insert(0, 'out.png')
outputEntry.pack(side=tk.LEFT)

# Create compression and dynamic rescaling checkboxes

compressionVar = tk.BooleanVar()
compressionCheck = tk.Checkbutton(encodeTab, text="Compression",
variable=compressionVar)
compressionCheck.pack(pady=10)

rescalingVar = tk.BooleanVar()
dynamicCheck = tk.Checkbutton(encodeTab, text="Dynamic Rescaling",
variable=rescalingVar)
dynamicCheck.pack()

# Create encode button
encodeButton = tk.Button(encodeTab, text="Encode", command=onEncode)
encodeButton.pack(pady=10)

# Create Encoding Label
encodingLabel = tk.Label(encodeTab, text="Encoding...")

# Create decode tab
decodeTab = ttk.Frame(tabs)
tabs.add(decodeTab, text="Decode")

imageBoxDecode = tk.Label(decodeTab)
imageBoxDecode.pack()

# Create choose image button for decoding
```

```
chooseImageButtonDecode = tk.Button(decodeTab, text="Choose an Image",
command=chooseImageDecode)
chooseImageButtonDecode.pack(pady=10)

decodedTextBox = tk.Text(decodeTab, height=5)

# Create decode destination path widget
destinationFrame = tk.Frame(decodeTab)
destinationFrame.pack(side=tk.TOP, padx=5, pady=5)

destinationLabel = tk.Label(destinationFrame, text="Output File Destination: Current
Directory")
destinationLabel.pack(side=tk.LEFT)

changeButton = tk.Button(destinationFrame, text="Browse",
command=outputDestinationChange)
changeButton.pack(side=tk.LEFT)

# Create decode button
decodeButton = tk.Button(decodeTab, text="Decode", command=onDecode)
decodeButton.pack(pady=10)

# Create Decoding Label
decodingLabel = tk.Label(decodeTab, text="Decoding...")



# Start main loop
window.mainloop()
```

10.2 aes_encryption.py

```
import base64
import hashlib

from Crypto import Random
from Crypto.Cipher import AES


class AESCipher(object):
    def __init__(self: "AESCipher", key: str) -> None:
        self.bs = AES.block_size
        # Convert the password of any length into a 32 byte key
        self.key = hashlib.sha256(key.encode()).digest()

    def encrypt(self: "AESCipher", raw: str) -> str:
        'Encrypts string data and returns the encrypted string'
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CFB, iv)
        return base64.b64encode(iv + cipher.encrypt(raw.encode())).decode("utf-8")

    def decrypt(self: "AESCipher", enc: bytes) -> str:
        'Decrypts string data and returns the decrypted string'
        enc = base64.b64decode(enc)
        iv = enc[: AES.block_size]
        cipher = AES.new(self.key, AES.MODE_CFB, iv)
        return (cipher.decrypt(enc[AES.block_size :])).decode("utf-8")
```

11. OUTPUT

11.1 Text Encoding

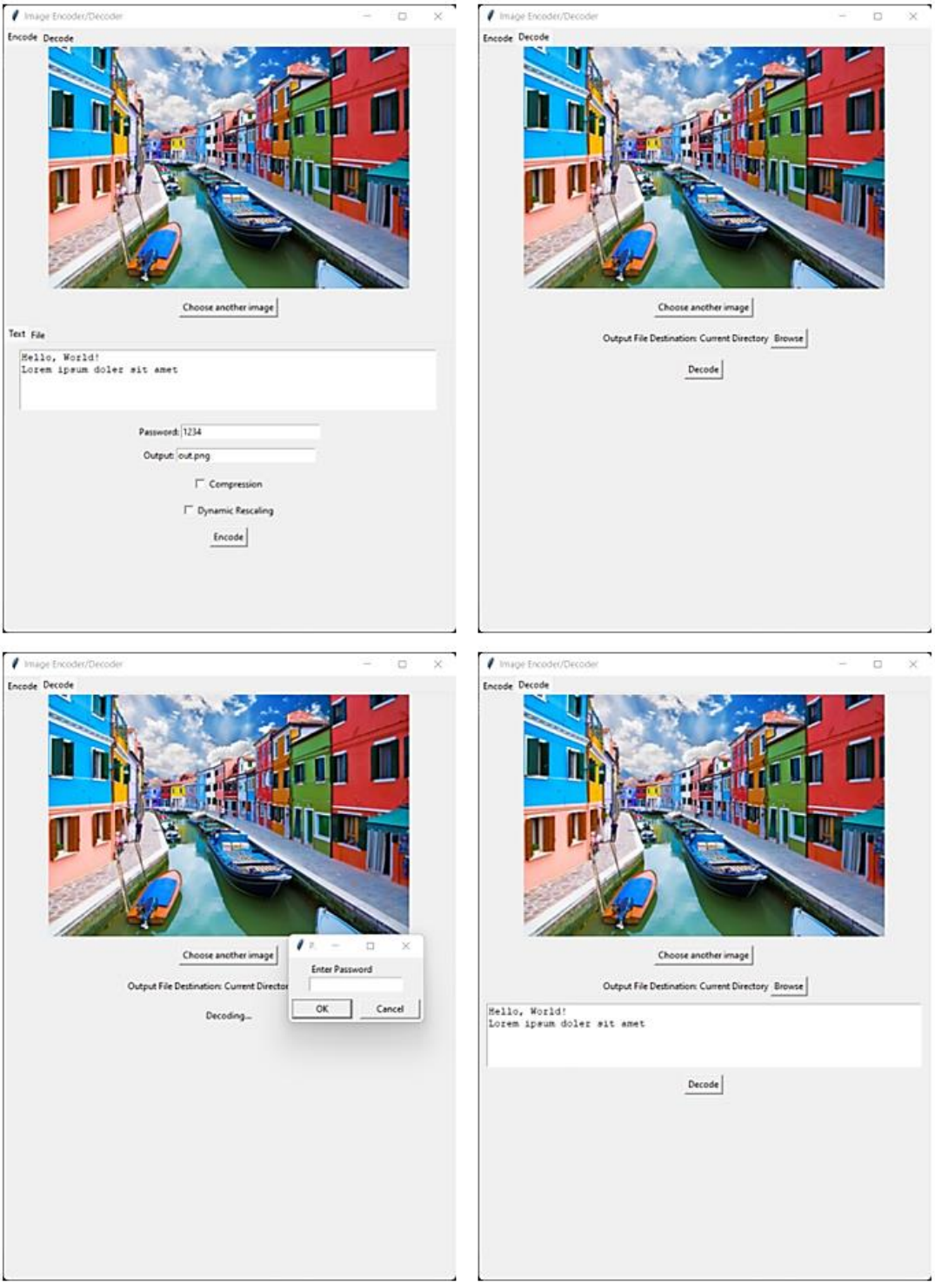

Fig 11.1 Text encoding output

11.2 File Encoding

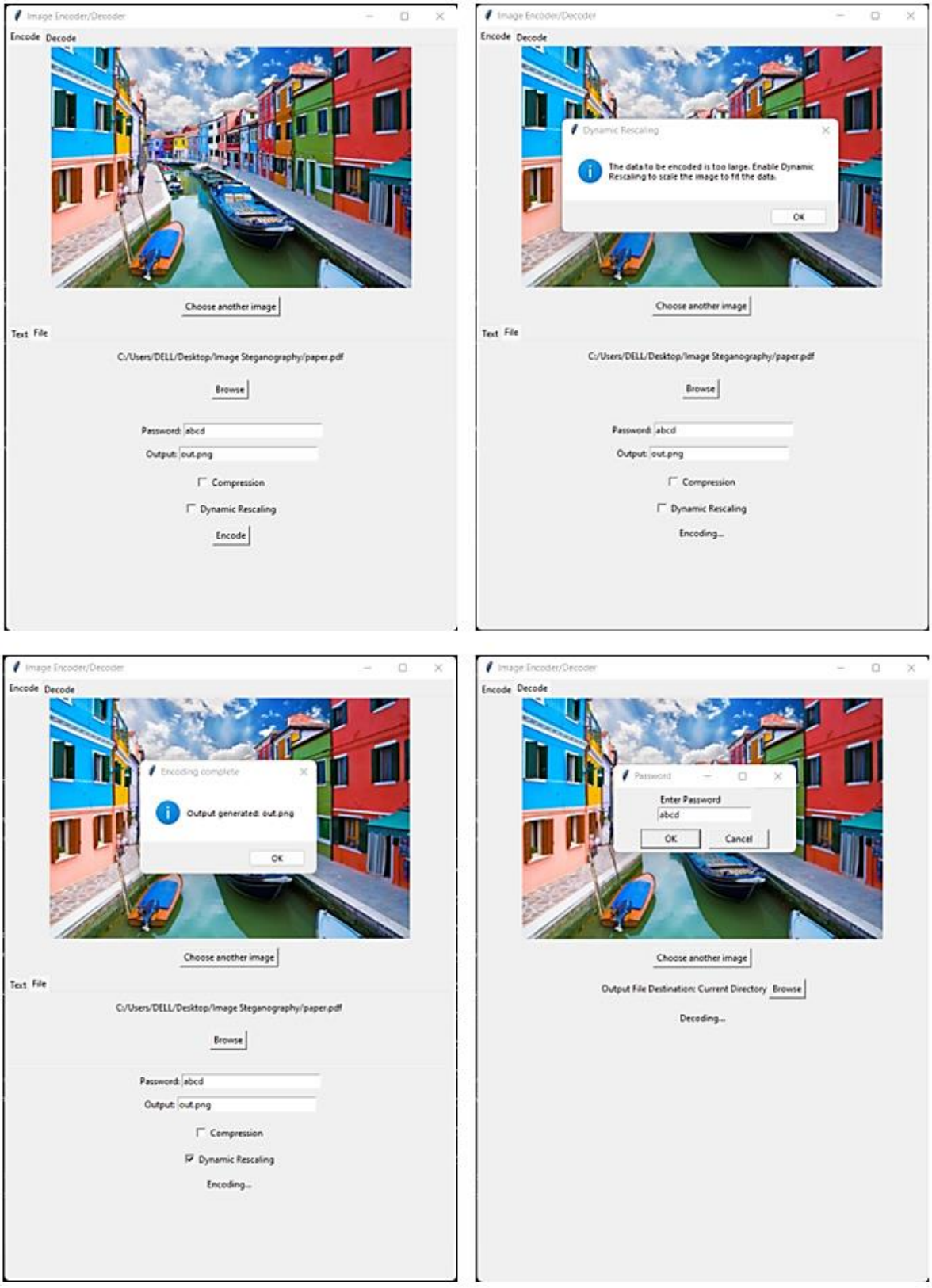

Fig 11.2 File encoding output

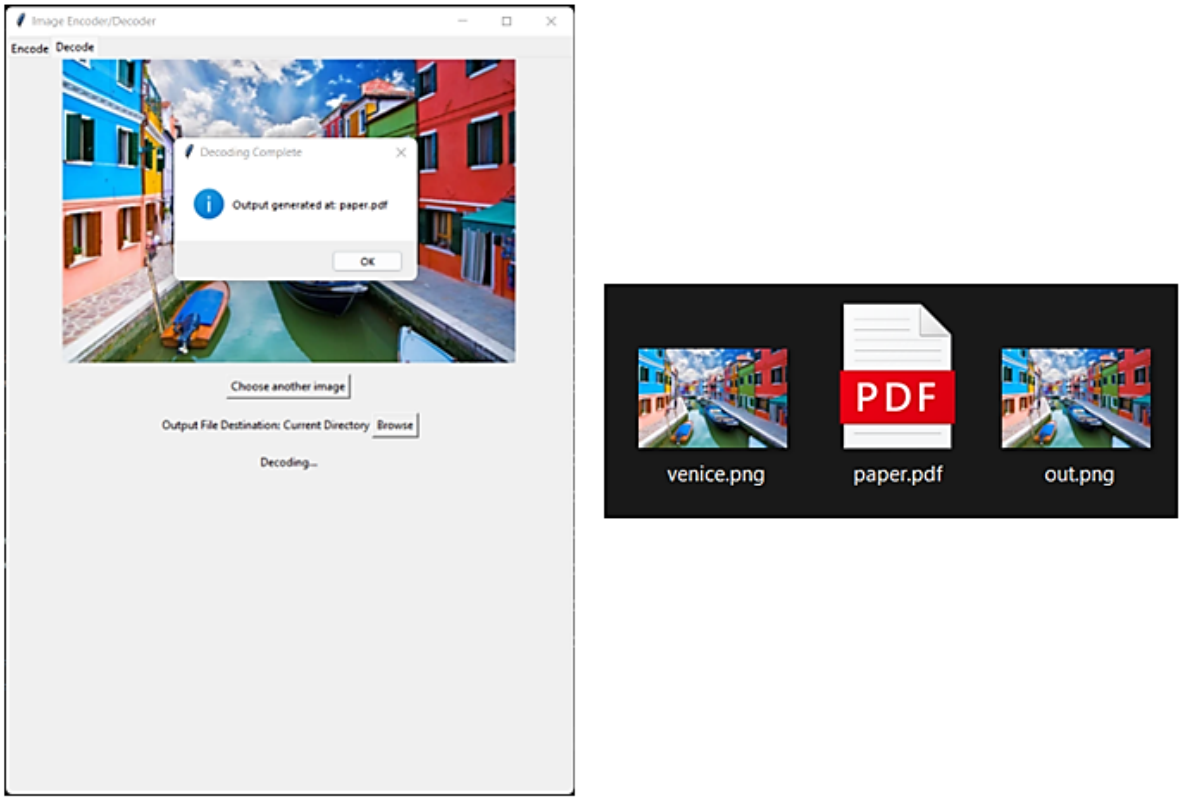

Fig 11.3 Generated output files

12. FEATURES AND LIMITATIONS

12.1 Features of the Program

Listed below are some unique features of the programs

- Can encode text and files into images.
- Capability to encrypt data before encoding.
- Dynamically scales the image to fit the data if Dynamic Scaling is enabled.
- Decoder automatically identifies encoding mode (file / text)
- Decoder identifies if the data is password protected, and validates the password entered.
- Decoder identifies the file type and generates the file as output.

12.2 Limitations

- Program is only tested to support PNG, BMP files as images.
- Performance is relatively poor as the program is written in Python
- In schemes like RGBA, the LSB of alpha value is not modified due to uncertainty in the range of its values.
- Does not support monochrome Bitmap images.

13. CONCLUSION

13.1 Conclusion

In conclusion, this project has explored the field of image steganography and has successfully implemented several techniques for hiding data within images. Through our research, we have gained a deeper understanding of the importance of data security and the various methods that can be employed to protect sensitive information. As we move into the 21st century, digital media is becoming abundant. Therefore, it is important for us to know these techniques to protect our privacy.
In summary, our project has demonstrated the potential of image steganography as a valuable tool for secure communication.

13.2 Future Prospects

- Data can be compressed before being encoded in the image to tackle the problem of low encoding limit.
- The project can be expanded to encoding the payload in various other files like audio and video files using similar steganographic techniques.
- The project can be ported to a lower-level language like C/C++ to improve performance.
- Better data hiding techniques can be used to avoid detection under steganalysis.

14. REFERENCES

1. N.F. Johnson, Z. Duric, and S. Jajodia, “Information hiding: Steganography and watermarking - attacks and countermeasures,” Kluwer Academic Publishers, 2000

2. R. Chandramouli and N.D. Memon, “A distributed detection framework for watermark analysis,” Proc. ACM Multimedia and Security Workshop, 2000.

3. Nandhini Subramanian; Omar Elharrouss, “Image Steganography: A Review of Recent Advances”, IEEE Explore

4. Corinne Bernstein, Michael Cobb, “Advanced Encryption Standard (AES)”: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard

5. K Thangadurai and G. Sudha Devi, “An analysis of LSB Based Image Steganography Techniques”, 2014 International Conference on Computer Communication and Informatics