Image and Audio Steganography

**SUBMITTED BY**

Aditya Varma Dantuluri

[202017b3670@wilp.bits-pilani.ac.in](mailto:202017b3670@wilp.bits-pilani.ac.in)

#### Synopsis on

*Image Steganography*

### Statement about the Problem:

Steganography is the practice of hiding private or sensitive information within something that appears to be nothing out to the usual. Steganography is often confused with cryptology because the two are similar in the way that they both are used to protect important information. The difference between two is that steganography involves hiding information, so it appears that no information is hidden at all. If a person or persons views the object that the information is hidden inside of he or she will have no idea that there is any hidden information, therefore the person will not attempt to decrypt the information.

What steganography essentially does is exploit human perception, human senses are not trained to look for files that have information inside of them, although this software is available that can do what is called Steganography. The most common use of steganography is to hide a file inside another file.

### Why is the topic chosen?

Throughout history Steganography has been used to secretly communicate information between people.

Some examples of use of Steganography is past times are:

1. During World War 2 invisible ink was used to write information on pieces of paper so that the paper appeared to the average person as just blank pieces of paper. Liquids such as milk, vinegar and fruit juices were used, because when each one of these substances are heated, they darken and become visible to the human eye.
2. In Ancient Greece they used to select messengers and shave their head, they would then write a message on their head. Once the message had been written the hair was allowed to grow back. After the hair grew back the messenger was sent to deliver the message, the recipient would shave off the messenger’s hair to see the secrete message.

Now in today’s electronic world the message is sent over E media. So there should be a necessity for new steganography techniques which will defend against interception attacks.

### Why This Steganography?

This technique is chosen, because this system includes not only imperceptibility but also un-delectability by any steganalysis tool.

### Objective: -

* + The requirement of this steganography system is that the hider message carried by stego-media should not be sensible to human beings.
  + The other goal of steganography is to avoid drawing suspicion to the existence of a hidden message.

This approach of information hiding technique has recently become important in a number of applications areas.

### Project Scope:

* + The scope of the project is to limit unauthorized access and provide better security during message transmission. To meet the requirements, I use the simple and basic approach of steganography.
  + In this project, the proposed approach finds the suitable algorithm for embedding the data in an image / audio using steganography which provides a better security pattern for sending messages through a network.

### Methodology:

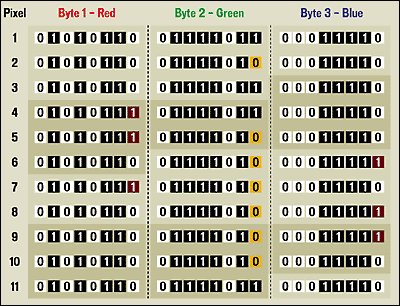
User needs to run the application. The user has two tab options – encrypt and decrypt. If the user select encrypt, application give the screen to select image file, information file and option to save the image file. If user select decrypt, application gives the screen to select only image file and ask path where user want to save the secrete file.

This project has two methods – Encrypt and Decrypt.

In encryption the secrete information is hiding in with any type of image file.Decryption is getting the secrete information from image file. We will make use of the LSB algorithm for this process.

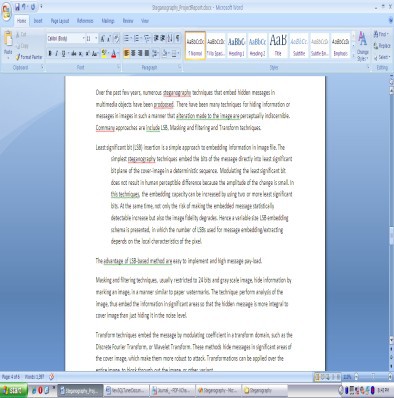
**LSB Algorithm:**

* + LSB (Least Significant Bit) substitution is the process of adjusting the least significant bit pixels of the carrier image.
  + It is a simple approach for embedding messages into the image.
  + The Least Significant Bit insertion varies according to the number of bits in an image.
  + For an 8-bit image, the least significant bit i.e., the 8th bit of each byte of the image is changed to the bit of secret message.
  + For 24-bit image, the colors of each component like RGB (red, green and blue) are changed.
  + LSB is effective in using BMP images since the compression in BMP is lossless



### Encryption Process

**IMAGE FILE INFORMATION FILE**

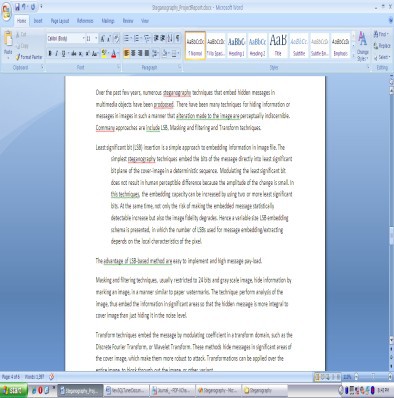


**BMP FILE**

### Decryption Process

**BMP FILE**

**INFORMATION IMAGE FILE**



**FILE**

### Limitations of the Software:

This project has an assumption that both the sender and receiver must have shared some secret information before imprisonment. Pure steganography means that there is no prior information shared by two communication parties.

### What contribution would this Project make?

* In the present world, the data transfers using the internet is rapidly growing because it is so easier as well as faster to transfer the data to destination. So, many individuals and business people use it to transfer business documents, important information using internet.
* Security is an important issue while transferring data using internet because any unauthorized individual can hack the data and make it useless or obtain information un- intended to him.

### Software Environment

* **Operating System** : Window 11
* **Development Environment** : Visual Studio Code
* **Languages** : Python

### Hardware Environment

* **Processor** : 11th Gen Intel(R) Core(TM) i7-1165G
* **RAM** : 16 GB
* **Hard Disk** : 1 TB

### Conclusion: -

Steganography is an interesting subject and outside of the mainstream cryptography and system administration that most of us deal with day after day.

Steganography can be used for hidden communication. We have explored the limits of steganography theory and practice. We printed out the enhancement of the image steganography system using LSB approach to provide a means of secure communication. A stego-key has been applied to the system during embedment of the message into the cover image.

This steganography application software provided for the purpose to how to use any type of image formats to hiding any type of files inside their. The master work of this application is in supporting any type of pictures without needing to convert to bitmap, and lower limitation on file size to hide, because of using maximum memory space in pictures to hide the file.

# Objective and Scope of the Project

Objective

The objective of steganography is covert communication. So, a fundamental requirement of this steganography system is that the hider message carried by stego-media should not be sensible to human beings.

The other objective of steganography is to avoid drawing suspicion to the existence of a hidden message. This approach of information hiding technique has recently became important in a number of application area

This project has following objectives:

* + To product security tool based on steganography techniques.
  + To explore techniques of hiding data using encryption module of this project
  + To extract techniques of getting secret data using decryption module.

Steganography is sometimes used when encryption is not permitted. Or, more commonly, steganography is used to supplement encryption. An encrypted file may still hide information using steganography, so even if the encrypted file is deciphered, the hidden message is not seen

**Scope of Steganography**

Steganography is a very interesting and advantageous science these days and has following uses:

* + - Digital Watermarking

To protect copyright on information. Photo collections, sold on CD, often have hidden messages in the photos which allow detection of unauthorized use. The same technique applied to DVDs is even more effective since the industry builds DVD recorders to detect and disallow copying of protected DVDs.

* + - The simplest and oldest are used in map making, where cartographers sometimes add a tiny fictional street to their maps, allowing them to prosecute copycats.
    - A similar trick is to add fictional names to mailing lists as a check against unauthorized resellers.
    - Steganography doesn't just apply to written forms of communication. Radio and TV messages, from World War II to today, can be used to hide coded or hidden messages. Some government sources suspect that Osama bin Laden's pre- recorded videos that are re-played on TV stations around the world contain hidden messages.
    - Even biological data, stored on DNA, may be a candidate for hidden messages, as biotech companies seek to prevent unauthorized use of their genetically engineered material. The technology is already in place for this: three New York researchers successfully hid a secret message in a DNA sequence and sent it across the country.
    - Steganography can also be used to allow communication within an underground community.
    - Steganography is used by some modern printers, including HP and Xerox brand color laser printers. Tiny yellow dots are added to each page. The dots are barely visible and contain encoded printer serial numbers, as well as date and time stamps.

# Overview

The word steganography comes from the Greek “Seganos”, which mean covered or secret and – “graphy” mean writing or drawing. Therefore, steganography means, literally, covered writing. It is the art and science of hiding information such its presence cannot be detected, and communication is happening. Secret information is encoding in a manner such that the very existence of the information is concealed. Paired with existing communication methods, steganography can be used to carry out hidden exchanges.

The main goal of this project it to communicate securely in a completely undetectable manner and to avoid drawing suspicion to the transmission of a hider data. There has been a rapid growth of interest in steganography for two reasons:

The publishing and broadcasting industries have become interested in techniques for hiding encrypted copyright marks and serial numbers in digital films, audio recordings, books and multimedia products.

Moves by various governments to restrict the availability of encryption services have motivated people to study methods by which private messages can be embedded in seemingly innocuous cover messages.

The basic model of steganography consists of Carrier, Message and password. Carrier is also known as cover-object, in which the message is embedded and serves to hide the presence of the message.

Basically, the model for steganography is shown on following figure:

Stego-key, K

Stego Object, Z

Message, M

**F(X,M,K)**

Cover- object, C

Message is the data that the sender wishes to remain confidential. It can be plain text, ciphertext, other image, or anything that can be embedded in a bit stream such as a copyright mark, a covert communication, or a serial number. Password is known as *stego-key*, which ensures that only recipient who knows the corresponding decoding key will be able to extract the message from a *cover-object*. The *cover-object* with the secretly embedded message is then called the *Stego-object*.

Recovering message from a *stego-object* requires the *cover-object* itselt and a corresponding decoding key if a *stego-key* was used during the encoding process. The original image may or may not be required in most applications to extract the message.

There are several suitable carriers below to be the *cover-object:*

* Network protocols such as TCP, IP and UDP
* Audio that using digital audio formats such as wav, midi, avi, mpeg, mpi and voc
* File and Disk that can hides and append files by using the slack space
* Text such as null characters, just alike morse code including html and java
* Images file such as bmp, gif and jpg, where they can be both color and grayscale.

In general, the information hiding process extracts redundant bits from *cover-object.* The process consists of two steps:

* Identification of redundant bits in a *cover-object.* Redundant bits are those bits that can be modified without corrupting the quality or destroying the integrity of the *cover-object.*
* Embedding process then selects the subset of the redundant bits to be replaced with data from a secret message. The *stego-object* is created by replacing the selected redundant bits with message bits.

##### Steganography vs Cryptography:

Basically, the purpose of cryptography and steganography is to provide secret communication. However, steganography is not the same as cryptography. Cryptography hides the contents of a secrete message from a malicious people, whereas steganography even conceals the existence of the message. In cryptography, the system is broken when the attacker can read the secret message. Breaking a steganography system needs the attacker to detect that steganography has been used.

It is possible to combine the techniques by encrypting message using cryptography and then hiding the encrypted message using steganography. The resulting stego-image can be transmitted without revealing that secret information is being exchanged.

# THEORETICAL BACKGROUND

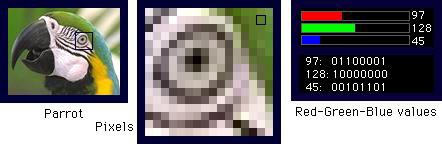
##### 

*An image file is merely a binary file containing a binary representation of the color or light intensity of each picture element (pixel) comprising the image.*

***Images typically use either 8-bit or 24-bit color****.*

When using 8-bit color, there is a definition of up to 256 colors forming a palette for this image, each color denoted by an 8-bit value.

A 24-bit color scheme, as the term suggests, uses 24 bits per pixel and provides a much better set of colors. In this case, each pixel is represented by three bytes, each byte representing the intensity of the three primary colors **red, green, and blue (RGB)**, respectively. The color orange, for example, would be displayed with red set to 100% , green set to 50% and no blue .



**The size of an image file, then, is directly related to the number of pixels and the granularity of the color definition**. A typical 640x480 pix image using a palette of 256 colors would require a file about 307 KB in size (640 • 480 bytes), whereas a 1024x768 pix high-resolution 24-bit color image would result in a 2.36 MB file (1024 • 768 • 3 bytes).

To avoid sending files of this enormous size, a number of compression schemes have been developed over time, notably **Bitmap (BMP), Graphic Interchange Format (GIF), and Joint Photographic Experts Group (JPEG)** file types. Not all are equally suited to steganography, however.

GIF and 8-bit BMP files employ what is known as **lossless compression**, a scheme that allows the software to exactly reconstruct the original image. JPEG, on the other hand, uses **lossy compression**, which means that the expanded image is very nearly the same as the original but not an exact duplicate. Lossless compression is much better suited to applications where the integrity of the original information must be maintained, such as steganography. While JPEG can be used for stego applications, it is more common to embed data in GIF or BMP files.



The simplest approach to hiding data within an image file is called **Least Significant Bit (LSB) insertion**. In this method, we can take the binary representation of the hidden\_data and overwrite the LSB of each byte within the cover\_image. If we are using 24-bit color, the amount of change will be minimal and indiscernible to the human eye. As an example, suppose that we have three adjacent pixels (nine bytes) with the following RGB encoding:

|  |  |
| --- | --- |
| 10010101 | 00001101 11001001 |
| 10010110 | 00001111 11001010 |
| 10011111 | 00010000 11001011 |

Now suppose we want to "hide" the following 9 bits of data (the hidden data is usually compressed prior to being hidden): 101101101. If we overlay these 9 bits over the LSB of the 9 bytes above, we get the following (where bits in bold have been changed):

|  |  |  |
| --- | --- | --- |
| 10010101 | 0000110**0** | 11001001 |
| 1001011**1** | 0000111**0** | 1100101**1** |
| 10011111 | 00010000 | 11001011 |

Note that we have successfully hidden 9 bits but at a cost of only changing 4, or roughly 50%, of the LSBs.

A 640x480 pixel image, the size of a small computer monitor, can hold over 400,000 characters. That's a whole novel hidden in one modest photo! This poject involves following formats of images:

Bitmap Images

Joint Photographic Experts Group Portable Network Graphics Tagged Image File Format Windows Meta Files

(.bmp)

(.jpg)

(.png)

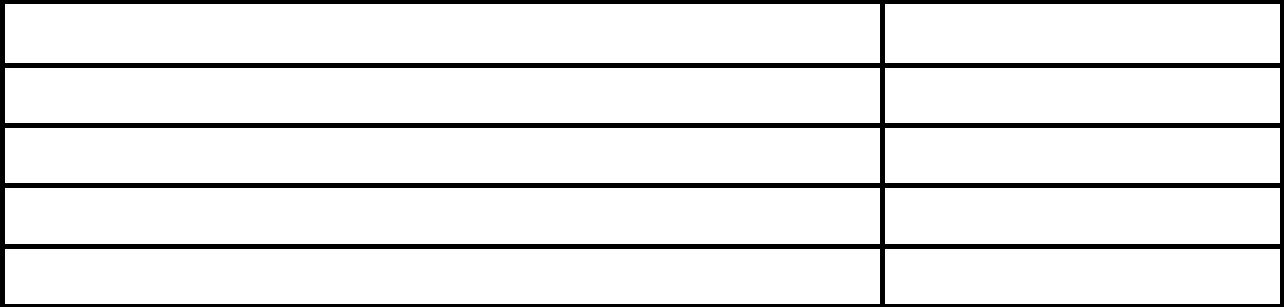
(.tif)

(.wmf)

**BMP Image File Format**

#### File Header

|  |  |
| --- | --- |
| Data | Size(Bytes) |
| File Type(BM in case of BMP files) | 2 |
| File Size | 4 |
| Reserved Byte(Always 0) | 1 |
| Reserved Byte(Always 0) | 1 |
| Bf of Bits |  |
| Size of Info Header | 4 |
| Width of Bitmap | 2 |
| Height of Bitmap | 2 |
| No of planes(1 for BMP) | 1 |
| Bit Count(Bits/Pixel, Must be 1, 4, 8, 24) | 1 |
| Type of Compression used(none) | 2 |
| Size of Image Data in Bytes | 2 |
| Horizontal Resolution in Pixels/Meter | 2 |
| Vertical Resolution in Pixels/Meter | 2 |
| No of Color’s Indexes Used | 2 |
| Important Color Indexes | 2 |

**Reading Pixel Values**

|  |  |
| --- | --- |
| Data | Size(Bytes) |
| R(Red) | 1 |
| B(Blue) | 1 |
| G(Green) | 1 |
| Reserved Word for RGB | 1 |

## ALGORITHM

1. Import the necessary modules: tkinter, messagebox from tkinter, and PIL (Python Imaging Library).

2. Create a function called `generate\_data` that takes in `pixels` (image pixels) and `data` (text to be encoded). This function converts the data into 8-bit binary format using ASCII values and returns them.

3. In the `generate\_data` function, iterate over each character in the data and convert it to binary using the `ord` function and `format` method. Append the binary representation to a list called `data\_in\_binary`.

4. Calculate the length of the data in binary (`length\_of\_data`) and create an iterator for the image pixels (`image\_data`).

5. Iterate `a` from 0 to `length\_of\_data` and perform the following steps:

- Extract the next three pixels from `image\_data` and store them in a list called `pixels`.

- Iterate `b` from 0 to 7 (8 bits) and perform the following steps:

- If the `b`th bit of the `a`th binary data is '1' and the `b`th pixel value is odd, decrement the pixel value by 1.

- If the `b`th bit of the `a`th binary data is '0' and the `b`th pixel value is even, increment the pixel value by 1 if it's not already 0, and then decrement it by 1.

- If it's the last iteration (`length\_of\_data-1 == a`), check the last pixel value. If it's even, increment it by 1 if it's not already 0, and then decrement it by 1.

- Yield the modified pixel values in groups of three (RGB format).

6. Create a function called `encryption` that takes an `img` (image object) and `data` (text to be encoded) as inputs. This function encodes the data into the image by calling the `generate\_data` function and modifying the pixels of the image.

7. In the `encryption` function, get the width of the image (`size`) and initialize `(x, y)` coordinates as `(0, 0)`.

8. Iterate over the pixels generated by the `generate\_data` function and perform the following steps:

- Set the current pixel at coordinates `(x, y)` to the generated pixel value.

- If `x` is equal to `size-1`, set `x` to 0 and increment `y` by 1.

- Otherwise, increment `x` by 1.

9. Create a function called `main\_encryption` that takes `img` (image path), `text` (text to be encoded), and `new\_image\_name` (output image filename) as inputs. This function opens the image, creates a copy of it, calls the `encryption` function, and saves the new image with the specified filename.

10. In the `main\_encryption` function, open the image using the `Image.open` method and assign it to the variable `image`.

11. Check if any of the input fields are empty. If any field is empty, show an error message using `messagebox.showerror`.

12. Create a copy of the image using the `copy` method and assign it to the variable `new\_image`.

13. Call the `encryption` function with `new\_image` and the text to be encoded as arguments.

14. Append the extension `.png` to the `new\_image\_name` and save the new image using the `save` method.

15. Create a function called `main\_decryption` that takes `img` (image path) and `strvar` (a StringVar object) as inputs. This function opens the image and extracts the hidden message from it by decoding the pixels.

16. In the `main\_decryption` function, open the image using the `Image.open` method and assign it to the variable `image`.

17. Initialize an empty string variable called `data` to store the decoded message.

18. Create an iterator for the image pixels (`image\_data`).

19. Set `decoding` to `True` to start the decoding process.

20. While `decoding` is `True`, perform the following steps:

- Extract the next three pixels from `image\_data` and store them in a list called `pixels`.

- Initialize a binary string variable called `binary\_string` to store the binary data extracted from the pixels.

- Iterate over the first 8 pixels in `pixels` and convert them to binary by checking if they are even or odd. If even, append '0' to `binary\_string`, otherwise append '1'.

- Convert the binary string to an ASCII character using `int` and `chr` functions, and append it to `data`.

- If the last pixel value is odd, set the value of `data` to `strvar`.

21. Create a function called `encode\_image` that opens a new window for encoding an image.

22. In the `encode\_image` function, create a new window (`encode\_wn`) using the `Toplevel` class and configure its properties.

23. Add labels and entry fields for entering the image path, data to be encoded, and output image filename.

24. Add a button to encode the image, which calls the `main\_encryption` function with the provided inputs.

25. Create a function called `decode\_image` that opens a new window for decoding an image.

26. In the `decode\_image` function, create a new window (`decode\_wn`) using the `Toplevel` class and configure its properties.

27. Add a label and an entry field for entering the image path.

28. Create a `StringVar` variable called `text\_strvar` to store the decoded message.

29. Add a button to decode the image, which calls the `main\_decryption` function with the provided image path and `text\_strvar`.

30. Add a label and an entry field (disabled state) to display the decoded text.

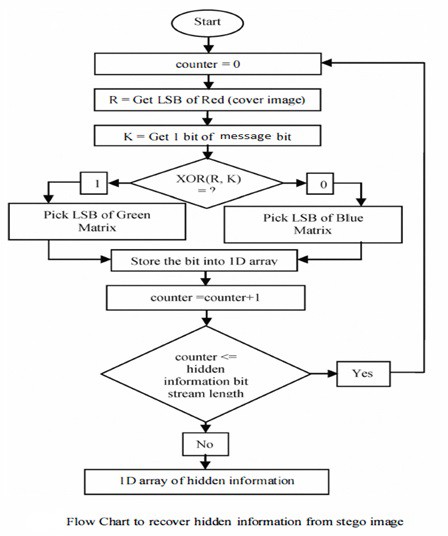
31. Create the main window (`root`) using the `Tk` class and configure its properties.

32. Add labels, buttons, and commands for the encode and decode functionalities.

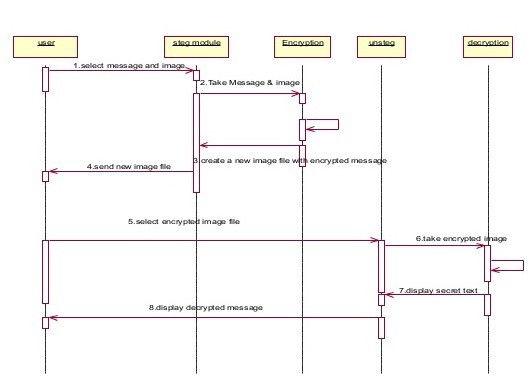
33. Run the main event loop using the `mainloop` method.

# Project Life Cycle

**Flow chart for Image Steganography**



**Sequence Diagram for Image Steganography**



### Source Code

###### Imagesteganography.py

from tkinter import \*

from tkinter import messagebox as mb

from PIL import Image

def generate\_data(pixels, data):

    data\_in\_binary = []

    for c in data:

        binary\_data = format(ord(c), '08b')

        data\_in\_binary.append(binary\_data)

    length\_of\_data = len(data\_in\_binary)

    image\_data = iter(pixels)

    for a in range(length\_of\_data):

        pixels = [val for val in image\_data.\_\_next\_\_()[:3] + image\_data.\_\_next\_\_()[:3] + image\_data.\_\_next\_\_()[:3]]

        for b in range(8):

            if (data\_in\_binary[a][b] == '1') and (pixels[b] % 2 != 0):

                pixels[b] -= 1

            elif (data\_in\_binary[a][b] == '0') and (pixels[b] % 2 == 0):

                if pixels[b] == 0:

                    pixels[b] += 1

                pixels[b] -= 1

        if (length\_of\_data-1) == a:

            if pixels[-1] % 2 == 0:

                if pixels[-1] == 0:

                    pixels[-1] += 1

                else:

                    pixels[-1] -= 1

        pixels = tuple(pixels)

        yield pixels[:3]

        yield pixels[3:6]

        yield pixels[6:9]

def encryption(img, data):

    # This method will encode data to the new image that will be created

    size = img.size[0]

    (a, b) = (0, 0)

    for pixel in generate\_data(img.getdata(), data):

        img.putpixel((a, b), pixel)

        if size-1 == a:

            a = 0; b += 1

        else:

            a += 1

def main\_encryption(img, text, new\_image\_name):

    # This function will take the arguments, create a new image, encode it and save it to the same directory

    image = Image.open(img, 'r')

    if (len(text) == 0) or (len(img) == 0) or (len(new\_image\_name) == 0):

        mb.showerror("Error", 'You have not put a value! Please put all values before pressing the button')

    new\_image = image.copy()

    encryption(new\_image, text)

    new\_image\_name += '.png'

    new\_image.save(new\_image\_name, 'png')

def main\_decryption(img, strvar):

    # This function will decode the image given to it and extract the hidden message from it

    image = Image.open(img, 'r')

    data = ''

    image\_data = iter(image.getdata())

    decoding = True

    while decoding:

        pixels = [value for value in image\_data.\_\_next\_\_()[:3] + image\_data.\_\_next\_\_()[:3] + image\_data.\_\_next\_\_()[:3]]

        # string of binary data

        binary\_string = ''

        for c in pixels[:8]:

            if c % 2 == 0:

                binary\_string += '0'

            else:

                binary\_string += '1'

        data += chr(int(binary\_string, 2))

        if pixels[-1] % 2 != 0:

            strvar.set(data)

# Creating the button functions

def encode\_image():

    encode\_wn = Toplevel(root)

    encode\_wn.title("Encode an Image")

    encode\_wn.geometry('600x220')

    encode\_wn.resizable(0, 0)

    encode\_wn.config(bg='AntiqueWhite')

    Label(encode\_wn, text='Encode an Image', font=("Comic Sans MS", 15), bg='AntiqueWhite').place(a=220, rely=0)

    Label(encode\_wn, text='Enter the path to the image(with extension):', font=("Times New Roman", 13),

          bg='AntiqueWhite').place(a=10, b=50)

    Label(encode\_wn, text='Enter the data to be encoded:', font=("Times New Roman", 13), bg='AntiqueWhite').place(

        a=10, b=90)

    Label(encode\_wn, text='Enter the output file name (without extension):', font=("Times New Roman", 13),

          bg='AntiqueWhite').place(a=10, b=130)

    img\_path = Entry(encode\_wn, width=35)

    img\_path.place(a=350, b=50)

    text\_to\_be\_encoded = Entry(encode\_wn, width=35)

    text\_to\_be\_encoded.place(a=350, b=90)

    after\_save\_path = Entry(encode\_wn, width=35)

    after\_save\_path.place(a=350, b=130)

    Button(encode\_wn, text='Encode the Image', font=('Helvetica', 12), bg='PaleTurquoise', command=lambda:

    main\_encryption(img\_path.get(), text\_to\_be\_encoded.get(), after\_save\_path.get())).place(a=220, b=175)

def decode\_image():

    decode\_wn = Toplevel(root)

    decode\_wn.title("Decode an Image")

    decode\_wn.geometry('600x300')

    decode\_wn.resizable(0, 0)

    decode\_wn.config(bg='Bisque')

    Label(decode\_wn, text='Decode an Image', font=("Comic Sans MS", 15), bg='Bisque').place(a=220, rely=0)

    Label(decode\_wn, text='Enter the path to the image (with extension):', font=("Times New Roman", 12),

          bg='Bisque').place(a=10, b=50)

    img\_entry = Entry(decode\_wn, width=35)

    img\_entry.place(a=350, b=50)

    text\_strvar = StringVar()

    Button(decode\_wn, text='Decode the Image', font=('Helvetica', 12), bg='PaleTurquoise', command=lambda:

    main\_decryption(img\_entry.get(), text\_strvar)).place(a=220, b=90)

    Label(decode\_wn, text='Text that has been encoded in the image:', font=("Times New Roman", 12), bg='Bisque').place(

        a=180, b=130)

    text\_entry = Entry(decode\_wn, width=94, text=text\_strvar, state='disabled')

    text\_entry.place(a=15, b=160, height=100)

# Initializing the window

root = Tk()

root.title('Bits Pilani Project by\nAditya Varma')

root.geometry('300x200')

root.resizable(0, 0)

root.config(bg='Moccasin')

Label(root, text='Bits Pilani Project by\nAditya Varma', font=('Aldhabi', 15), bg='Moccasin',

      wraplength=500).place(x=65, y=10)

Button(root, text='Encode', width=25, font=('Times New Roman', 13), bg='SteelBlue', command=encode\_image).place(

    x=30, y=80)

Button(root, text='Decode', width=25, font=('Times New Roman', 13), bg='SteelBlue', command=decode\_image).place(

    x=30, y=130)

# Finalizing the window

root.update()

root.mainloop()

v

# Annexure

### Bibliography

Websites

Following websites are referring to create this project reports.

* [http://www.wikipedia.org](http://www.wikipedia.org/)
* <https://www.geeksforgeeks.org/>
* <https://www.python.org/>
* <https://docs.python.org/3/library/tkinter.html>
* <https://pillow.readthedocs.io/en/stable/>

**Books**

Following books and ebook are used to complete reports.

* "Python GUI Programming with Tkinter" by Alan D. Moore
* "Cryptography and Network Security: Principles and Practice" by William Stallings
* “Python Imaging Library Handbook" by Fredrik Lundh
* Professional ASP.NET (Paperback)
* MCAD/MCSD Self-Paced Training Kit: Developing Web Applications with Microsoft® Visual Basic® .NET and Microsoft Visual C#® .NET, Second Edition

# Appendix

## KEYWORDS AND DEFINITIONS

**Steganography:** The art and science of hidden writing. **Cryptography :** The science of writing in secret codes. **Cover Medium:** file in which we will hide the hidden\_data

**Plain Text :** Data to be hidden.

**Cipher Text :** The encrypted data to be hidden.

**Stego Key :** Data is hidden by using this string **Stego Medium :** The final resultant file after hiding data. **Bit Stream :** The binary code generated from the string.