Jaypee Institute of Information Technology, Noida

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY



**Project Title:** Steganography in Python

**Enroll. No. Name of Student**

21103042 Astha Raghuwanshi

21103043 Prerna

21103048 Princi Agrawal

Course Name: Information Security Lab Course Code: 15B17CI576 Program: B. Tech. CS&E

3rd Year 5th Sem

**2023 - 2024**

# ABSTRACT

The purpose of this lab project is to explore the concepts and techniques of steganography, specifically focusing on image and audio steganography. Steganography involves the art and science of hiding information within other data in a way that is not easily detectable. In this project, we will delve into both image and audio steganography, understanding the methods of concealing information within these types of media and analyzing the security implications.

# INTRODUCTION

In the dynamic landscape of information security, where safeguarding sensitive data is paramount, the clandestine art of steganography emerges as a crucial ally. Unlike encryption which focuses on securing the content of a message, steganography takes a subtle approach by concealing the very existence of the message itself. This clandestine communication technique has evolved into a vital aspect of modern cybersecurity, offering a covert means to transmit information without triggering suspicion.

In the realm of steganography, our project aims to unravel the intricacies of two of its most prevalent and intriguing forms: image and audio steganography. These techniques delve into the depths of multimedia files, harnessing the imperceptible nuances within images and audio to embed hidden information. As we embark on this exploration, we seek to not only understand the methodologies behind these covert communication techniques but also to dissect their security implications and unveil the challenges in detecting and countering such surreptitious practices.

# IMAGE STEGANOGRAPHY

Image encoding:

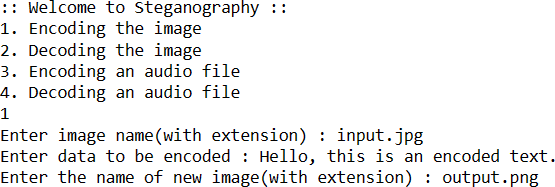
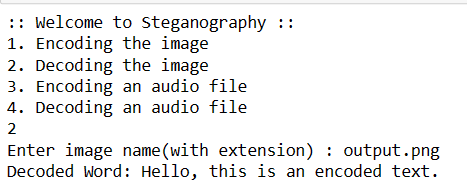


Image decoding:



The technique used for image steganography in the provided Python code is the Least Significant Bit (LSB) substitution. LSB substitution is one of the simplest and widely used methods in image steganography. In this method, the least significant bit of each pixel in the image is replaced with a bit of the secret message.

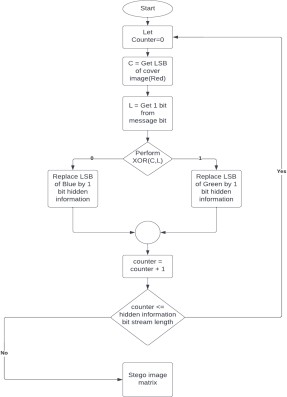
Encoding (Embedding):

The genData function converts the characters of the secret message into 8-bit binary form using the ASCII values.

The modPix function iterates through the pixels of the image and modifies the least significant bit of each color channel (RGB) to encode the binary data.

The binary '0' or '1' from the message is substituted into the least significant bit of the pixel values, ensuring minimal visual impact on the image.

Flow Chart for encoding:

Decoding:

The decode function iterates through the pixels of the image and extracts the least significant bit from each color channel.

These extracted bits are then concatenated to form the binary representation of the hidden message.

The binary message is then converted back to ASCII characters.

This basic LSB substitution technique is easy to implement, but it has some limitations. It is sensitive to image manipulations, and modifications to the least significant bit may introduce visual artifacts, especially in high-contrast or smooth regions of the image. Advanced steganographic techniques may employ more sophisticated methods to enhance security and reduce the likelihood of detection.

Flow chart for decoding:

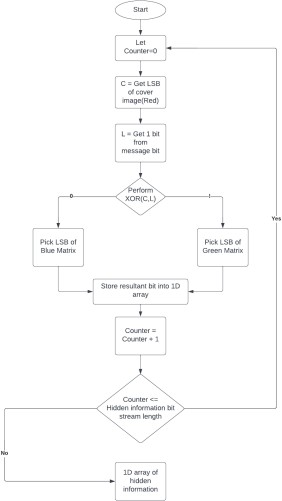


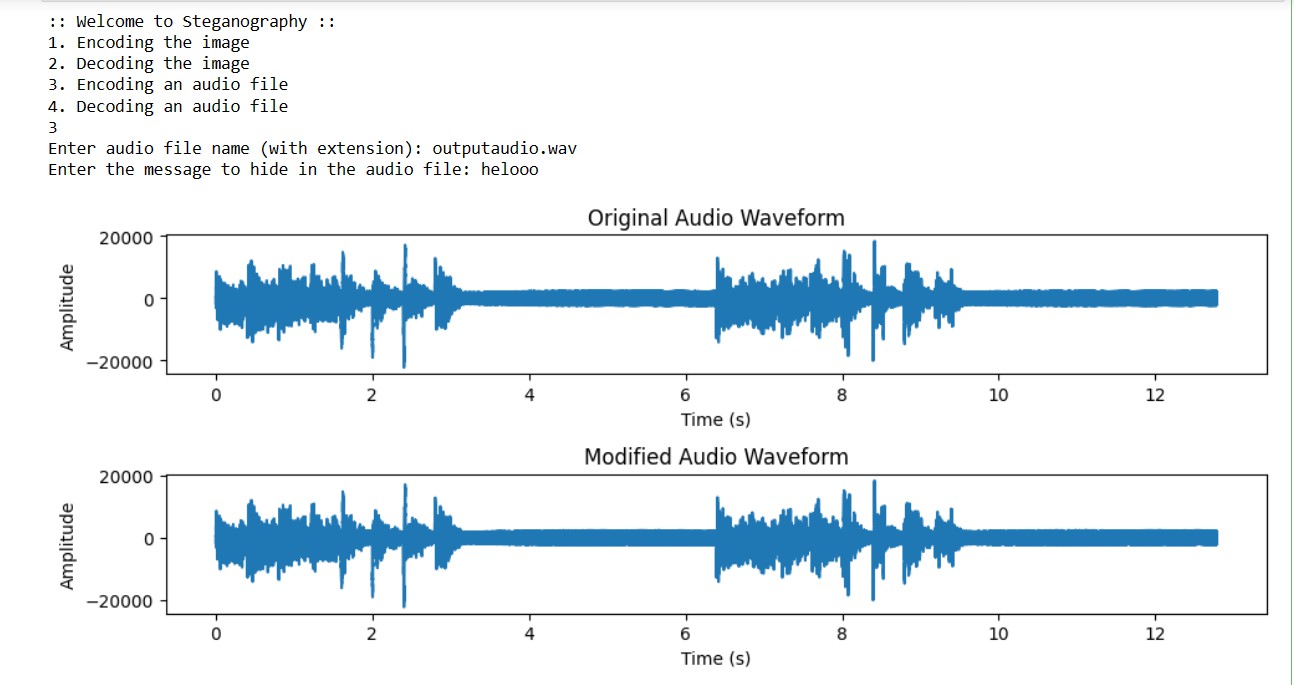


Fig1 : Input image Fig2: Output image

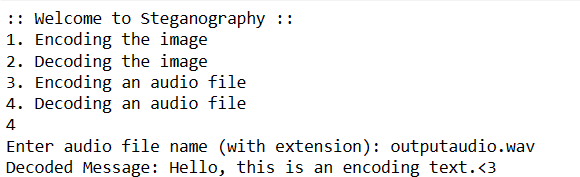
After encoding the input image, the output image looks exactly same with our message being hidden which is the beauty of image steganography!!

# AUDIO STEGANOGRAPHY

Audio encoding:



Audio decoding:



The technique used for audio steganography is Least Significant Bit (LSB) substitution. Similar to image steganography, LSB substitution is a common and straightforward method for embedding information within audio files.

Here's how the LSB substitution technique is applied to audio steganography in the code:

Encoding (Embedding):

The hide\_message function takes the audio samples (waveform) and a message as input.

The message is converted into a binary string where each character is represented by its ASCII value in 8 bits.

The least significant bit of each audio sample is replaced with a corresponding bit from the binary message.

The modified audio samples are then written to a new audio file. Decoding:

The decode\_message function reads the audio samples from the audio file.

It extracts the least significant bit from each audio sample and concatenates them to form a binary string.

The binary string is then processed to retrieve the hidden message.

It's important to note that while LSB substitution is a straightforward method, it may not provide high security against more advanced steganalysis techniques. There are other advanced audio steganography techniques that manipulate different characteristics of audio signals, such as frequency domain modifications, phase coding, or spread spectrum methods, to embed information more robustly and reduce the likelihood of detection. These advanced techniques often aim to minimize the impact on the perceptual quality of the audio while maximizing the hidden data capacity.

Noise and interference can disrupt the delicate balance maintained by steganographic algorithms when hiding information in an audio file. Even subtle changes introduced by noise or interference can lead to errors during the decoding process, causing the extracted text to appear as gibberish or unintended characters.

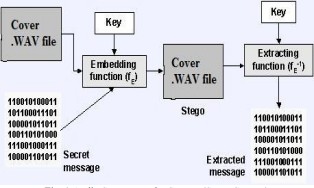


Fig3: Audio Steganography

# IMPLEMENTATION

import matplotlib.pyplot as plt

# PIL module is used to extract # pixels of image and modify it from PIL import Image

import numpy as np import wave

# Convert encoding data into 8-bit binary # form using ASCII value of characters def genData(data):

# list of binary codes # of given data

newd = []

for i in data: newd.append(format(ord(i), '08b'))

return newd

# Pixels are modified according to the

# 8-bit binary data and finally returned def modPix(pix, data):

datalist = genData(data) lendata = len(datalist) imdata = iter(pix)

for i in range(lendata):

# Extracting 3 pixels at a time

pix = [value for value in imdata. next ()[:3] +

imdata. next ()[:3] + imdata. next ()[:3]]

# Pixel value should be made # odd for 1 and even for 0 for j in range(0, 8):

if (datalist[i][j] == '0' and pix[j]% 2 != 0):

pix[j] -= 1

elif (datalist[i][j] == '1' and pix[j] % 2 == 0): if(pix[j] != 0):

pix[j] -= 1 else:

pix[j] += 1 # pix[j] -= 1

# Eighth pixel of every set tells # whether to stop ot read further.

# 0 means keep reading; 1 means thec # message is over.

if (i == lendata - 1):

if (pix[-1] % 2 == 0):

if(pix[-1] != 0):

pix[-1] -= 1 else:

pix[-1] += 1

else:

if (pix[-1] % 2 != 0):

pix[-1] -= 1

pix = tuple(pix) yield pix[0:3] yield pix[3:6] yield pix[6:9]

def encode\_enc(newimg, data): w = newimg.size[0]

(x, y) = (0, 0)

for pixel in modPix(newimg.getdata(), data):

# Putting modified pixels in the new image newimg.putpixel((x, y), pixel)

if (x == w - 1): x = 0

y += 1

else:

x += 1

# Encode data into image def encode():

img = input("Enter image name(with extension) : ") image = Image.open(img, 'r')

data = input("Enter data to be encoded : ") if (len(data) == 0):

raise ValueError('Data is empty')

newimg = image.copy() encode\_enc(newimg, data)

new\_img\_name = input("Enter the name of new image(with extension) : ") newimg.save(new\_img\_name, str(new\_img\_name.split(".")[1].upper()))

# Decode the data in the image

def decode():

img = input("Enter image name(with extension) : ") image = Image.open(img, 'r')

data = ''

imgdata = iter(image.getdata())

while (True):

pixels = [value for value in imgdata. next ()[:3] +

imgdata. next ()[:3] + imgdata. next ()[:3]]

# string of binary data binstr = ''

for i in pixels[:8]: if (i % 2 == 0):

binstr += '0' else:

binstr += '1'

data += chr(int(binstr, 2)) if (pixels[-1] % 2 != 0):

return data # ... (previous code)

def hide\_message(audio\_path, output\_path, message): # Read the audio file

with wave.open(audio\_path, 'rb') as audio\_file: params = audio\_file.getparams()

frames = audio\_file.readframes(params.nframes) samples = np.frombuffer(frames, dtype=np.int16)

# Display the original waveform plt.figure(figsize=(10, 4))

plt.subplot(2, 1, 1)

plt.plot(np.linspace(0, len(samples) / params.framerate, num=len(samples)), samples)

plt.title('Original Audio Waveform') plt.xlabel('Time (s)') plt.ylabel('Amplitude')

# Convert the message to a binary string

binary\_message = ''.join(format(ord(char), '08b') for char in message)

# Embed the message in the least significant bits of the audio samples modified\_samples = samples.copy()

for i in range(len(binary\_message)):

modified\_samples[i] = (modified\_samples[i] & 0xFFFE) | int(binary\_message[i])

# Display the modified waveform plt.subplot(2, 1, 2)

plt.plot(np.linspace(0, len(modified\_samples) / params.framerate, num=len(modified\_samples)), modified\_samples)

plt.title('Modified Audio Waveform') plt.xlabel('Time (s)') plt.ylabel('Amplitude')

plt.tight\_layout() plt.show()

# Ask the user for the path of the output audio file

output\_path = input("Enter the path of the output audio file (with extension): ")

# Write the modified samples to the new audio file with wave.open(output\_path, 'wb') as output\_file:

output\_file.setparams(params) output\_file.writeframes(modified\_samples.tobytes())

print(f"Message hidden in {output\_path}")

def decode\_message(audio\_path): # Read the audio file

with wave.open(audio\_path, 'rb') as audio\_file: params = audio\_file.getparams()

frames = audio\_file.readframes(params.nframes) samples = np.frombuffer(frames, dtype=np.int16)

# Extract the least significant bits from the audio samples to get the binary message

binary\_message = ''.join(str(sample & 1) for sample in samples)

1s)

# Find the index of the first occurrence of '11111111' (eight consecutive end\_index = binary\_message.find('11111111')

# If '11111111' is found, extract the binary message before it, otherwise, use the entire binary message

binary\_message = binary\_message[:end\_index] if end\_index != -1 else binary\_message

# Ensure that the length of the binary message is a multiple of 8 binary\_message = binary\_message[:-(len(binary\_message) % 8)] if

len(binary\_message) % 8 != 0 else binary\_message

# Convert the binary message to characters

message = ''.join(chr(int(binary\_message[i:i+8], 2)) for i in range(0, len(binary\_message), 8))

return message

def main():

a = int(input(":: Welcome to Steganography ::\n"

"1. Encoding the image\n2. Decoding the image\n3. Encoding an audio file\n4. Decoding an audio file\n"))

if a == 1:

encode() elif a == 2:

print("Decoded Word: " + decode()) elif a == 3:

audio\_path = input("Enter audio file name (with extension): ") message = input("Enter the message to hide in the audio file: ") hide\_message(audio\_path, None, message)

elif a == 4:

audio\_path = input("Enter audio file name (with extension): ") decoded\_message = decode\_message(audio\_path)

print("Decoded Message:", decoded\_message) else:

raise Exception("Invalid option. Please enter a valid option (1, 2, 3, or 4).")

# ... (remaining code) # Driver Code

if name == ' main ': # Calling main function main()

# CONCLUSION

The scope of this project is to limit unauthorized access and provide better security during message transmission. In this project, we mainly concentrated on embedding the data into an image and an audio. “You never know if a message is hidden”, this is the dilemma that empowers steganography. As more emphasis is placed on the areas of copyright protection, privacy protection, and surveillance, we believe that steganography will continue to grow in importance as a protection mechanism. This

project deals with Steganography in Image and Audio files using **Least Significant Bit (LSB) coding**. By focusing on the concealment of data in both image and audio formats through LSB coding, the project not only contributes to the current discourse on protection mechanisms but also anticipates the continued growth of steganography's importance in safeguarding sensitive information during transmission. The discreet and adaptable nature of this approach positions it as a valuable asset in the broader context of secure communication.

# REFERENCES

* <https://core.ac.uk/download/pdf/234671204.pdf>
* <https://www.ijsr.net/archive/v11i5/SR22511125940.pdf>
* [https://sumit-arora.medium.com/audio-steganography-the-art-of-hiding-secret](https://sumit-arora.medium.com/audio-steganography-the-art-of-hiding-secrets-within-earshot-part-2-of-2-c76b1be719b3) [s-within-earshot-part-2-of-2-c76b1be719b3](https://sumit-arora.medium.com/audio-steganography-the-art-of-hiding-secrets-within-earshot-part-2-of-2-c76b1be719b3)
* [https://medium.com/swlh/lsb-image-steganography-using-python-2bbbee2c69](https://medium.com/swlh/lsb-image-steganography-using-python-2bbbee2c69a2) [a2](https://medium.com/swlh/lsb-image-steganography-using-python-2bbbee2c69a2)