ABSTRACT

Video steganography is an intriguing and innovative field of research within the domain of information security and digital forensics. Steganography involves concealing sensitive data or secret information within a seemingly normal carrier medium to prevent unauthorized access or detection. Unlike cryptography, which focuses on encrypting data to secure it, steganography aims to hide data in plain sight, making it indistinguishable from the original content.

The tool uses a spatial domain method for video steganography, specifically focusing on the Least Significant Bit (LSB) technique. In this method, data concealment occurs by substituting the least significant bits of pixel values in video frames with secret information. The concept capitalizes on the undetectable impact of altering these minor bits on the overall visual quality of the video.

The video steganography tool uses Python as the main programming language. To manipulate and process videos, the project leveraged the capabilities of OpenCV, a powerful library for computer vision and image processing. Other relevant libraries were also utilized to enhance the tool functionality and ensure efficient video data handling. The tools collectively provided the foundation for implementing the steganography techniques and seamlessly embedding secret information within video files while maintaining the normal appearance of the video

In conclusion, this project successfully achieves its objectives by developing a simple yet effective video steganography tool. The implementation demonstrates the chosen techniques' proficiency in concealing information within video files while maintaining the visual integrity of the video. The evaluation confirms the tool's effectiveness and highlights the importance of considering key performance metrics during steganography implementation.

Keywords: video steganography, Least Significant Bit (LSB) technique, spatial domain method, steganography, digital forensics.

TABLE OF CONTENTS

List of Figures

CHAPTER - 1

1. INTRODUCTION

1.1 INTRODUCTION TO THE SYSTEM

In the ever-evolving landscape of information security and digital forensics, our project introduces a cutting-edge Video Steganography Tool designed to fortify data protection measures. This sophisticated system leverages the capabilities of Python and integrates a curated selection of modules to provide users with a robust platform for concealing and retrieving sensitive information within multimedia files. Recognizing the need for covert communication and secure data manipulation, the Video Steganography Tool operates within the spatial domain of images, audio, and videos, employing techniques such as the Least Significant Bit (LSB) methodology. By seamlessly embedding secret data into seemingly innocuous media, the tool ensures a covert layer of information security, contributing to the broader efforts in safeguarding digital assets.

1.2 PROBLEM STATEMNET

Within the domain of information security and digital forensics, the current landscape of video steganography tools presents challenges related to user accessibility, functionality, and the seamless integration of spatial domain methods. Existing solutions, relying on techniques like the Least Significant Bit (LSB) methodology, may lack a comprehensive approach, limiting their effectiveness in concealing and retrieving sensitive information within multimedia files.

The problem at hand involves the development of a Video Steganography Tool that transcends the constraints of current solutions. This tool should integrate advanced spatial domain methods, particularly focusing on the LSB technique, to ensure secure data concealment within video frames. Emphasis is placed on creating a user-friendly graphical interface that accommodates users with varying technical expertise. The challenge extends to providing enhanced functionalities, including the seamless splitting and combining of videos, as well as the hiding and retrieval of data within frames and audio files. The aim is to deliver a versatile and accessible solution that addresses the current limitations in video steganography tools, offering a reliable means of manipulating multimedia files securely.

1.3 OBJECTIVE

The primary objective of our Video Steganography Tool is to provide users with a versatile and user-friendly solution for concealing and retrieving information within video files. Leveraging Python and a curated selection of modules, the tool encompasses a range of functionalities, including splitting videos into frames, combining audio and frames into videos, and performing steganalysis to identify frames with hidden data. The tool's capabilities extend to hiding and retrieving data within both audio and image frames, offering a comprehensive suite for secure data manipulation.

1.4 AIM OF THE PROJECT

The overarching aim of our project is to develop a versatile and user-friendly Video Steganography Tool. By empowering users to split videos into audio and frames, combine audio and frames into videos, and hide/retrieve data from frames and audio, the tool seeks to be an invaluable asset in the realm of digital forensics and information security. Furthermore, the incorporation of steganalysis capabilities adds an additional layer of sophistication, enabling users to discern the presence of concealed data within frames. This project aspires to contribute to the broader field of secure data manipulation and communication, providing an effective solution for both experts and novices alike.

CHAPTER - 2

2. LITERATURE SURVEY

The project delves into the intricate field of video steganography, an indispensable facet of information security. Extensive literature elucidates a myriad of techniques within this realm, with a particular emphasis on spatial domain methods like the revered Least Significant Bit (LSB) technique. Studies, such as those conducted by Gupta et al. (2018) and Khan et al. (2017), underscore the efficiency of LSB-based steganography in videos, revealing its prowess in concealing data imperceptibly.

Spatial domain methods, which manipulate pixel values, are indispensable components of video steganography. Drawing inspiration from successful adaptations in image steganography, our project integrates these methods seamlessly into video frames. The custom tool's development, a relatively nascent area in the field, responds to the emerging need for innovative solutions that effortlessly combine established steganographic techniques with cutting-edge technology.

Python, with its multifaceted modules, plays a central role in our project's execution. Modules such as Pillow for image processing, Tkinter for the graphical user interface, and Moviepy for video manipulation provide a robust foundation for multimedia processing. Literature by Clark (2018) and Rosebrock (2019) enriches our understanding of these modules, offering insights into maximizing their capabilities for effective data manipulation.

Moreover, OpenCV, a powerful library for computer vision and image processing, stands as a cornerstone in the project's video processing capabilities. The influential work of Bradski and Kaehler (2008) substantiates the significance of OpenCV, guiding the integration of Python and OpenCV to ensure the tool's efficiency and reliability.

1. R. Chandramouli and N. Memon's articles gives the analysis of various methods of LSB techniques for image steganography

2. Johnson, N. and Jajodia ST his article explores the different methods of steganography such as LSB, Masking and Filtering and also explains about different software tools present in the market for Steganography, such as Stego Dos, White Noise Storm, S-tool etc.

3 .Bender et al describes the techniques of data hiding like low bit rate data hiding in detail

2.1 EXISTING SYSTEM

Current systems in the field of video steganography exhibit varying degrees of sophistication and effectiveness. The prevalent methods often rely on spatial domain techniques, including the widely employed Least Significant Bit (LSB) methodology. Gupta et al. (2018) demonstrate the efficiency of LSB-based steganography in videos, emphasizing its ability to conceal data within multimedia files without perceptible degradation. However, existing tools might lack comprehensive functionalities or user-friendly interfaces, limiting accessibility for users with varying technical expertise.

2.2 PROPOSED SYSTEM

In response to the nuances and limitations of existing systems, our project introduces a novel Video Steganography Tool designed to offer an advanced and user-friendly solution. The proposed system integrates state-of-the-art spatial domain methods, including the LSB technique, ensuring effective data concealment within video frames. Leveraging the capabilities of Python and incorporating modules such as Pillow, Tkinter, and Moviepy, the tool enhances multimedia processing and user accessibility. The custom tool not only facilitates the splitting and combining of videos but also enables users to hide and retrieve data within frames and audio.

2.3 SCOPE OF THE PROJECT

The Video Steganography Tool aims to carve a niche in the ever-evolving landscape of information security and digital forensics. Its scope encompasses a comprehensive solution for concealing and retrieving sensitive information within multimedia files, with a primary focus on video data. The tool seeks to enhance the user experience by integrating cutting-edge spatial domain methods, specifically the Least Significant Bit (LSB) technique, ensuring the covert embedding of data within video frames. The scope extends to facilitating user-friendly operations through a well-designed graphical interface, making the tool accessible to users with varying technical backgrounds. Beyond conventional functionalities, the project incorporates features such as the seamless splitting and combining of videos, providing users with a versatile platform for secure data manipulation. The Video Steganography Tool aspires to contribute to the broader field of information security by addressing current limitations in existing solutions and providing a robust, user-centric, and efficient means of multimedia data concealment.

CHAPTER – 3

3. HARDWARE AND SOFTWARE REQUIREMETS

3.1. Hardware Requirements

- **Operating system:** Windows
- **RAM**: 256 MB (minimum requirement)
- **Hard Disk**: 10 GB

3.2. Software Requirements

- **IDE**: Python IDLE, Visual Studio
- Windows terminal or power shell
- **Language Used**: Python

CHAPTER-4

SYSTEM DESIGN

Video steganography is the technique of embedding secret data within video files in a way that is imperceptible to the human eye. This technique has various applications, such as secure communication, covert data transmission, and copyright protection.

Requirement Analysis: Clearly define the requirements of the steganography tool, such as the type of video files it supports, the embedding and extraction methods, and the level of security desired.

System Architecture: Design the overall architecture of the system, including the different modules, their interactions, and the data flow between them. Consider factors like modularity, coupling, and cohesion.

Module Design: Break down the system into smaller, more manageable modules and design each module in detail. Define the functionalities, interfaces, and internal data structures of each module.

Data Design: Design the data structures and databases required to store and manage the video data and embedded messages. Consider data integrity, security, and efficiency.

Implementation Plan: Create a detailed implementation plan that outlines the tasks, timelines, and resources required to develop the steganography tool.

SOFTWARE DESIGN

The software design for the video steganography tool should adhere to the following principles:

Modularity and Partitioning: Divide the software into separate modules with well-defined interfaces to promote code re-usability and maintainability.

Coupling: Minimize the coupling between modules to enhance independence and simplify testing.

Cohesion: Ensure each module performs a single, well-defined task to improve code clarity and maintainability.

Shared Use: Avoid code duplication by creating reusable modules that can be shared across different parts of the application.

INPUT DESIGN

The input design should focus on efficient data collection and user interaction:

Data Format: Define the input data format, including video file types, file structure, and supported metadata.

Error Handling: Implement robust error handling mechanisms to gracefully handle invalid or corrupted input data.

User Interface: Design a user-friendly interface that allows users to easily select video files, specify embedding options, and monitor the embedding process.

OUTPUT DESIGN

The output design should prioritize user-friendly presentation and efficient message extraction:

Output Format: Define the output data format, including video file structure, embedded message format, and associated metadata.

Data Security: Implement encryption and decryption mechanisms to protect the embedded messages from unauthorized access.

Message Extraction: Design efficient algorithms for extracting hidden messages from the stego video files without compromising video quality.

UML DIAGRAMS

Unified Modelling Language

The Unified Modelling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven to be successful in the modelling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. UML mostly uses graphical notations to express the design of software projects. Using the UML helps the project teams to communicate, explore potential designs and validate the architectural design of the software.

The Unified Modelling Language (UML) is a standard language for writing software blue prints. The UML is a language for

- Visualizing
- Specifying
- Constructing
- Documenting the artifacts of a software system.

UML is a language which provides vocabulary and the rules for combining words in that vocabulary for the purpose of communication. A modelling language is a language whose vocabulary and the rules focus on the conceptual and physical representation of a system. Modelling yields an understanding of a system.

A use case diagram is a graphical representation in the Unified Modeling Language (UML) that provides a high-level view of a system's functionalities from an external user's perspective. It serves to outline the various ways that users interact with a system and showcases the specific tasks or goals they can achieve. The diagram consists of actors (representing users or external systems) and use cases (depicting specific functionalities or features). Actors are entities that initiate interactions with the system, while use cases represent specific functionalities the system offers. Connecting lines between actors and use cases illustrate the interactions, demonstrating which actors are involved in each use case. Use case diagrams serve as a valuable tool for system analysts, designers, and stakeholders to gain a quick, visual understanding of a system's scope and user interactions, aiding in the early stages of system requirements gathering and design.

USE CASE DIAGRAM

- Use case Diagram consists of use case and actors.
- The main purpose is to show the interaction between the use cases and the actor.
- It intends to represent the system requirements from the user's perspective.
- The use cases are the functions that are to be performed in the module.
- Actors represent entities that interact with the system. They can be users, external systems, or even other software components.
- Use case diagrams serve as a foundation for more detailed system design, helping to identify components and interfaces that need to be developed.
- Use cases are depicted as ovals and represent specific functionalities or tasks that the system can perform. They describe the system's behavior from an external perspective.

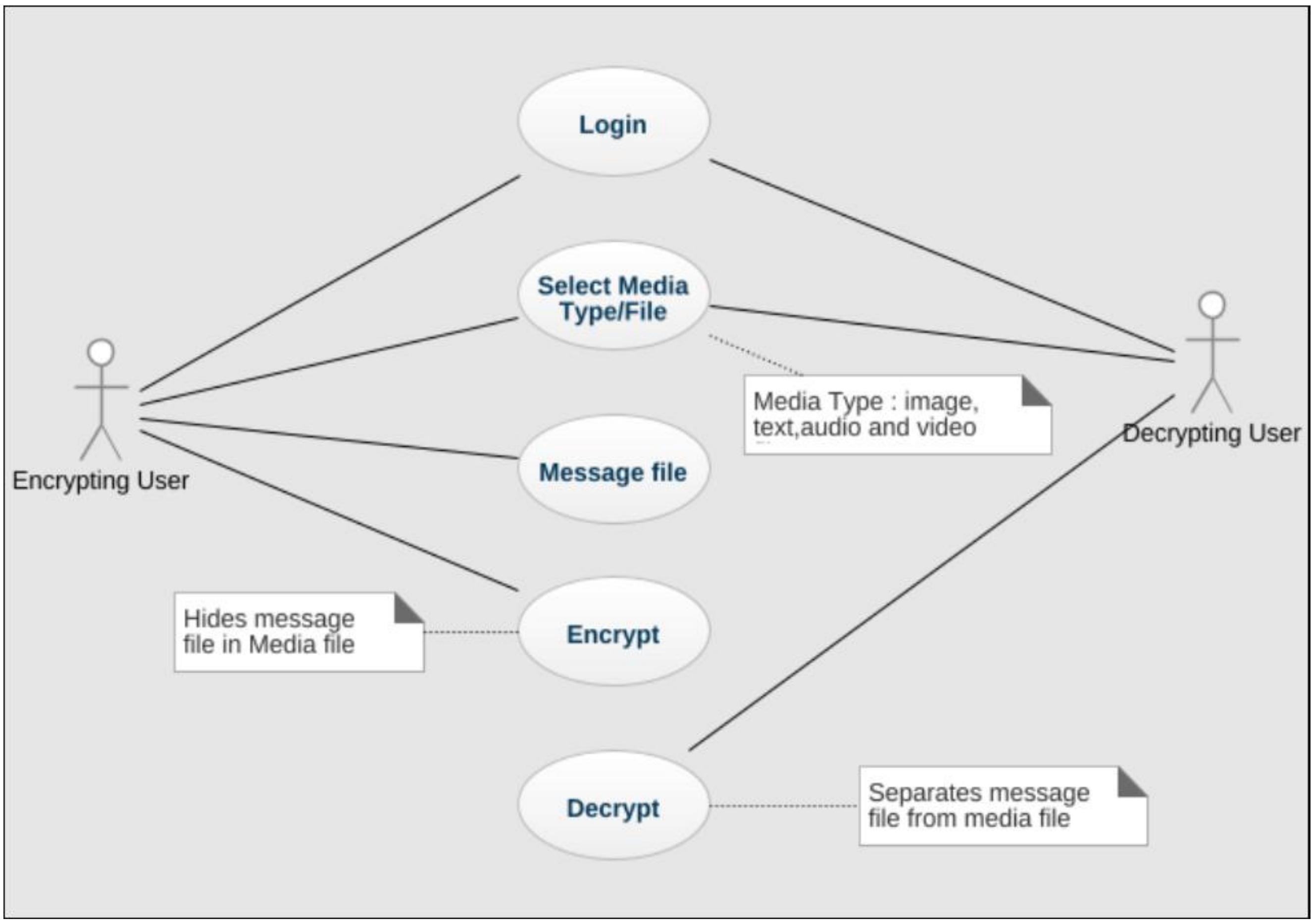
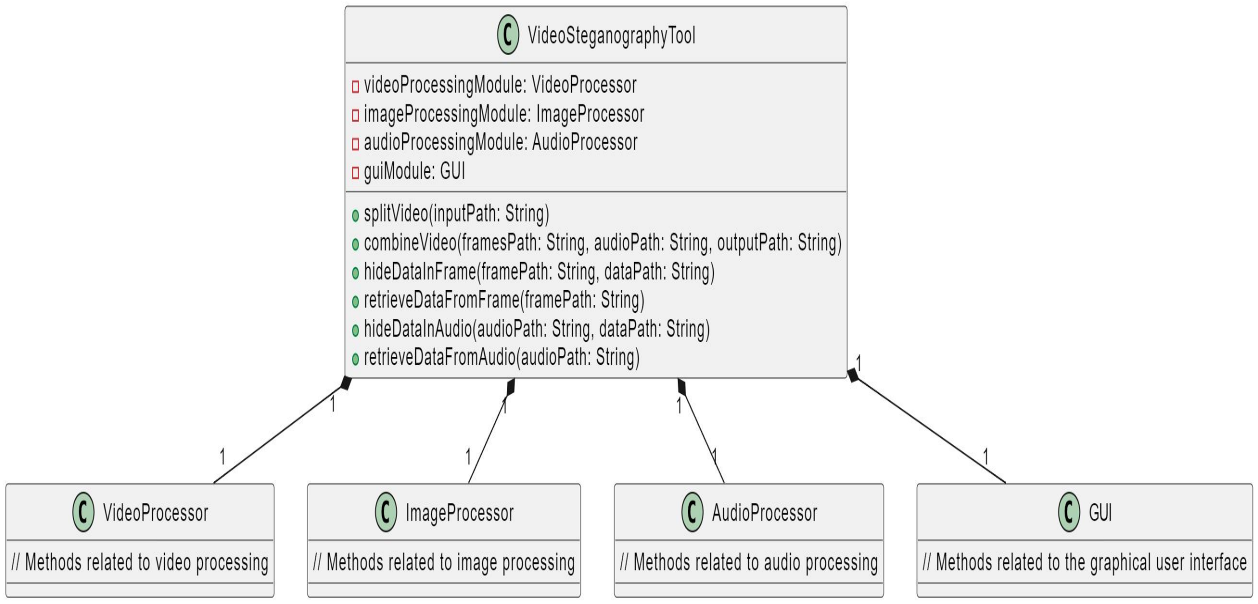

Login
Select Media Type/File
Media Type : image, text,audio and video
Decrypting User
Encrypting User
Message file
Hides message file in Media file
Encrypt
Decrypt
Separates message file from media file

Fig - 4.1 use case diagram

CLASS DIAGRAM

- It contains the classes involved and shows the connections between the variousclasses.
- Class diagram includes classes, which further have a class label or name, attributes of the class and the operations of functions performed by the class.



VideoSteganographyTool
videoProcessingModule: VideoProcessor
imageProcessingModule: ImageProcessor
audioProcessingModule: AudioProcessor
guiModule: GUI
splitVideo(inputPath: String)
combineVideo(framesPath: String, audioPath: String, outputPath: String)
hideDataInFrame(framePath: String, dataPath: String)
retrieveDataFromFrame(framePath: String)
hideDataInAudio(audioPath: String, dataPath: String)
retrieveDataFromAudio(audioPath: String)
1
1
1
1
1
1
1
1
VideoProcessor
// Methods related to video processing
ImageProcessor
// Methods related to image processing
AudioProcessor
// Methods related to audio processing
GUI
// Methods related to the graphical user interface

Fig - 4.2 Class diagram

SEQUENCE DIAGRAM

- It shows the sequence of the steps that are carried out throughout the process of execution.
- It involves lifelines or life time of a process that shows the duration for which the process is alive while the steps are taking place in the sequential manner.
- The sequence diagram specifies the order in which the various steps are executed.

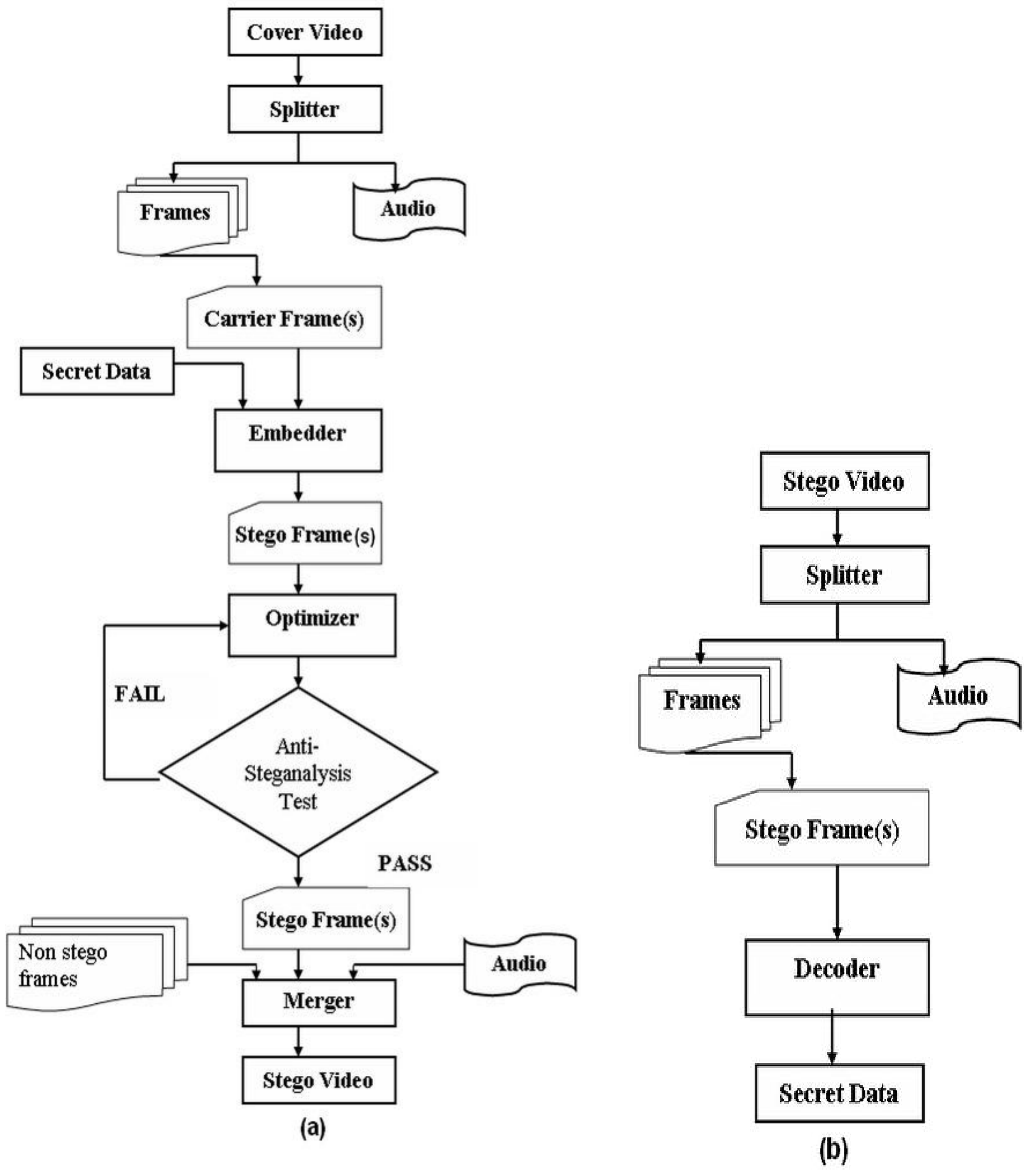


Fig - 4.3 Sequence diagram

CHAPTER-5

5.MODULES USED

Pillow:

Pillow, a powerful Python Imaging Library, serves as a fundamental module for image processing within the Video Steganography Tool. It enables efficient handling of image files, supporting tasks such as loading, saving, and manipulating pixel values crucial for steganographic operations.

Tkinter:

Tkinter, the standard GUI toolkit for Python, is instrumental in crafting the user-friendly graphical interface of the tool. It provides interactive elements, allowing users to navigate functionalities seamlessly. The modular design of Tkinter enhances the overall user experience, making the tool accessible to users with diverse technical backgrounds.

OpenCV:

OpenCV, a versatile computer vision library, plays a central role in video processing within the tool. It facilitates tasks such as splitting videos into frames and combining audio and frames, contributing to the core functionalities of the Video Steganography Tool. OpenCV's robust capabilities in image and video manipulation ensure efficient multimedia processing.

Moviepy:

Moviepy, a Python library for video editing, further augments the tool's capabilities in handling multimedia files. It simplifies video manipulation tasks, allowing users to seamlessly combine audio and frames to reconstruct the original video file. Moviepy enhances the overall efficiency and versatility of the Video Steganography Tool.

Matplotlib:

While not explicitly mentioned in the provided details, Matplotlib might be employed for optional features such as visualizing changes in Least Significant Bits (LSB) during image steganography. If uncommented in the code, Matplotlib could offer users insights into the alterations made to pixel values during the steganographic process.

Other Custom Modules:

The project might incorporate custom modules specific to the Video Steganography Tool's unique functionalities. These could include modules for steganalysis, data encryption, and other tailored features that enhance the overall capabilities of the tool.

CHAPTER-6

6. IMPLEMENTATION CODE

RunStartHere.py

```python
import os

# Checks if output Directory Exists, otherwise Create It
if not os.path.exists('output'):
    os.makedirs('output')

# Menu
print("\n1: Video Splitter and Combiner")
print("2: Hide Data in Audio")
print("3: Recover Data in Audio")
print("4: Hide Data in Frames")
print("5: Recover Data in Frames")

# User Selection
try:
    start_step = int(input("\nSelect the Program to Run: "))

    if start_step == 1:
        print("Starting Program...\n")
        os.system("python aviGUI.py")

    if start_step == 2:
        print("Starting Program...\n")
        print("=== Hide Data in Audio ===")
        file_text = input("Text File to Hide: ")
        file_audioOld = input("Original Audio File (inc. extension): ")
        bits_int = str(input("LSB Bits: "))
        line_string = ("python wav-steg.py -h -d \"" + file_text + "\" -s \"" + file_audioOld
+ "\" -o  output\steg_audio.wav" + " -n " + bits_int)
        os.system(line_string)

    elif start_step == 3:
        print("Starting Program...\n")
        print("=== Recover Data in Audio ===")
        audio_file = input("File to Recover From (inc. extension): ")
        no_of_lsb_bits = str(input("LSB Bits: "))
        no_of_bytes = str(input("Number of Bytes: "))
                line_string  =  ("python  wav-steg.py  -r  -s  \""  +  audio_file  +  "\"  -o
output\decoded_audio.txt" + " -n " + no_of_lsb_bits + " -b " + no_of_bytes)
        os.system(line_string)

    elif start_step == 4:
        print("Starting Program...\n")
        print("=== Hide Data in Frames ===")
        os.system("python Encoder.py")

    elif start_step == 5:
        print("Starting Program...\n")
        print("=== Recover Data in Frames ===")
        os.system("python Decoder.py")

    else:
        print("\nInvalid Input! Exiting...\n")
        quit()

except ValueError:
    print("Non-Integer Input Entered. Exiting...\n")
except KeyboardInterrupt:
    print("\nUser canceled, exiting...")
    quit()
```

aviGUI.py

```
import tkinter as tk
import sys
import os
import cv2

from PIL import Image
from moviepy.editor import *

def wrapper():
    """Wrapper function to call subfunction"""
    vf, base_filename = get_video_filename_base()
    option = var.get()
    if option == 1:
        video_object = VideoFileClip(vf)
        get_video(base_filename, video_object)
    elif option == 2:
        video_object = VideoFileClip(vf)
        get_audio(base_filename, video_object)
    elif option == 3:
        video_file = video_filename.get()
        audio_file = audio_filename.get()
        og_file = og_filename.get()
        combine_audio_video(video_file, audio_file, og_file)
    else:
        video_object = VideoFileClip(vf)
        get_frames(video_object, base_filename)

def get_video_filename_base():
    """Returns filename and base filename"""
    vf = video_filename.get()
    return vf, os.path.splitext(os.path.basename(vf))[0]

def get_audio_filename_base():
    """Returns audio filename"""
    return audio_filename.get()

def get_video(base_filename, video_object):
    """Returns the video track only of a video clip"""
            video_object.write_videofile(filename=f'output\\{base_filename}_video_only.mp4', 
audio=False)

def get_audio(base_filename, video_object):
    """Returns the audio track only of a video clip"""
    video_object.audio.write_audiofile(filename=f'output\\{base_filename}_audio.wav')
    video_object.audio.write_audiofile(filename=f'output\\{base_filename}_audio.mp3')

def combine_audio_video(video_path, audio_path, og_path):
    """Combines an audio and a video object together"""
    capture = cv2.VideoCapture(og_path) # Stores OG Video into a Capture Window
    fps = capture.get(cv2.CAP_PROP_FPS) # Extracts FPS of OG Video

    video_path_real = video_path + "\\%d.png" # To Get All Frames in Folder

    os.system("ffmpeg-4.3.1-2020-10-01-full_build\\bin\\ffmpeg -framerate %s -i \"%s\" -codec 
copy output\\combined_video_only.mkv" % (str(int(fps)), video_path_real)) # Combining the 
Frames into a Video
                           os.system("ffmpeg-4.3.1-2020-10-01-full_build\\bin\\ffmpeg       -i 
output\\combined_video_only.mkv -i \"%s\" -codec copy output\\combined_video_audio.mkv" % 
audio_path) # Combining the Frames and Audio into a Video

    print("Combining Complete!")

def get_frames(video_object, base_filename):
    """Returns all frames in the video object"""
    directory = "output\\" + base_filename + '_frames\\'
    if not os.path.isdir(directory):
        os.makedirs(directory)
    for index, frame in enumerate(video_object.iter_frames()):
        img = Image.fromarray(frame, 'RGB')
        img.save(f'{directory}{index}.png')

def sel():
    """Helper function to handle radio button selection"""
    if var.get() == 3:
        audio_filename_widget.config(state=tk.NORMAL)
        og_filename_widget.config(state=tk.NORMAL)
    else:
        audio_filename_widget.config(state=tk.DISABLED)
        og_filename_widget.config(state=tk.DISABLED)

window_main = tk.Tk()
window_main.title('AVI-Extractor')
window_main.geometry("400x200")
window_main.grid_columnconfigure((0,1), weight=1)

video_filename = tk.StringVar()

video_label = tk.Label(window_main, text="Video File/Frame Path: ")
video_label.grid(row=1, column=0)

video_filename_widget = tk.Entry(window_main, textvariable=video_filename)
video_filename_widget.grid(row=1, column=1)

audio_filename = tk.StringVar()

audio_label = tk.Label(window_main, text="Audio File: ")
audio_label.grid(row=2, column=0)

audio_filename_widget = tk.Entry(window_main, textvariable=audio_filename, state=tk.DISABLED)
audio_filename_widget.grid(row=2, column=1)

og_filename = tk.StringVar()

og_label = tk.Label(window_main, text="Original Video File: ")
og_label.grid(row=3, column=0)

og_filename_widget = tk.Entry(window_main, textvariable=og_filename, state=tk.DISABLED)
og_filename_widget.grid(row=3, column=1)

var = tk.IntVar()
R1 = tk.Radiobutton(window_main, text="Extract Video Without Audio", variable=var, value=1,
command=sel)
R1.grid(row=4, columnspan=2, sticky=tk.W)

R2 = tk.Radiobutton(window_main, text="Extract Audio Without Video", variable=var, value=2,
command=sel)
R2.grid(row=5, columnspan=2, sticky=tk.W)

R3 = tk.Radiobutton(window_main, text="Combine Audio and Video", variable=var, value=3,
command=sel)
R3.grid(row=6, columnspan=2, sticky=tk.W)

R4 = tk.Radiobutton(window_main, text="Get Frames of Video", variable=var, value=4,
command=sel)
R4.grid(row=7, columnspan=2, sticky=tk.W)

run = tk.Button(window_main, text="Run", command=wrapper)
run.grid(row=7, columnspan=2, sticky=tk.N)

window_main.mainloop()
```

wav-steg.py

```
import getopt, os, sys, math, struct, wave

def print_usage():
    print("Error Occured")
    #print("\nUsage options:\n",
       # "-h, --hide     If present, the script runs to hide data\n",
       # "-r, --recover  If present, the script runs to recover data\n",
       # "-s, --sound    What follows is the name of carrier wav file\n",
       # "-d, --data     What follows is the file name having data to hide\n",
       # "-o, --output   Output filename of choice\n",
       # "-n, --nlsb     Number of LSBs to use\n",
       # "-b, --bytes    Number of bytes to recover\n"
        #" --help         Display help\n")

def prepare(sound_path):
    global sound, params, n_frames, n_samples, fmt, mask, smallest_byte
    sound = wave.open(sound_path, "r")

    params = sound.getparams()
    num_channels = sound.getnchannels()
    sample_width = sound.getsampwidth()
    n_frames = sound.getnframes()
    n_samples = n_frames * num_channels

    if (sample_width == 1):  # samples are unsigned 8-bit integers
        fmt = "{}B".format(n_samples)
        # Used to set the least significant num_lsb bits of an integer to zero
        mask = (1 << 8) - (1 << num_lsb)
        # The least possible value for a sample in the sound file is actually
        # zero, but we don't skip any samples for 8 bit depth wav files.
        smallest_byte = -(1 << 8)
    elif (sample_width == 2):  # samples are signed 16-bit integers
        fmt = "{}h".format(n_samples)
        # Used to set the least significant num_lsb bits of an integer to zero
        mask = (1 << 15) - (1 << num_lsb)
        # The least possible value for a sample in the sound file
        smallest_byte = -(1 << 15)
    else:
        # Python's wave module doesn't support higher sample widths
        raise ValueError("File has an unsupported bit-depth")

def hide_data(sound_path, file_path, output_path, num_lsb):
    global sound, params, n_frames, n_samples, fmt, mask, smallest_byte
    prepare(sound_path)
    # We can hide up to num_lsb bits in each sample of the sound file
    max_bytes_to_hide = (n_samples * num_lsb) // 8
    filesize = os.stat(file_path).st_size

    if (filesize > max_bytes_to_hide):
        required_LSBs = math.ceil(filesize * 8 / n_samples)
        raise ValueError("Input file too large to hide, "
                         "requires {} LSBs, using {}"
                         .format(required_LSBs, num_lsb))

    print("Using {} B out of {} B".format(filesize, max_bytes_to_hide))


    # Put all the samples from the sound file into a list
    raw_data = list(struct.unpack(fmt, sound.readframes(n_frames)))
    sound.close()

    input_data = memoryview(open(file_path, "rb").read())

    # The number of bits we've processed from the input file
    data_index = 0
    sound_index = 0

    # values will hold the altered sound data
    values = []
    buffer = 0
    buffer_length = 0
    done = False

    while(not done):
        while (buffer_length < num_lsb and data_index // 8 < len(input_data)):
            # If we don't have enough data in the buffer, add the
            # rest of the next byte from the file to it.
            buffer += (input_data[data_index // 8] >> (data_index % 8)
                        ) << buffer_length
            bits_added = 8 - (data_index % 8)
            buffer_length += bits_added
            data_index += bits_added

        # Retrieve the next num_lsb bits from the buffer for use later
        current_data = buffer % (1 << num_lsb)
        buffer >>= num_lsb
        buffer_length -= num_lsb

        while (sound_index < len(raw_data) and
               raw_data[sound_index] == smallest_byte):
            # If the next sample from the sound file is the smallest possible
            # value, we skip it. Changing the LSB of such a value could cause
            # an overflow and drastically change the sample in the output.
            values.append(struct.pack(fmt[-1], raw_data[sound_index]))
            sound_index += 1

        if (sound_index < len(raw_data)):
            current_sample = raw_data[sound_index]
            sound_index += 1

            sign = 1
            if (current_sample < 0):
                # We alter the LSBs of the absolute value of the sample to
                # avoid problems with two's complement. This also avoids
                # changing a sample to the smallest possible value, which we
                # would skip when attempting to recover data.
                current_sample = -current_sample
                sign = -1

            # Bitwise AND with mask turns the num_lsb least significant bits
            # of current_sample to zero. Bitwise OR with current_data replaces
            # these least significant bits with the next num_lsb bits of data.
            altered_sample = sign * ((current_sample & mask) | current_data)

            values.append(struct.pack(fmt[-1], altered_sample))

        if (data_index // 8 >= len(input_data) and buffer_length <= 0):
            done = True

    while(sound_index < len(raw_data)):
        # At this point, there's no more data to hide. So we append the rest of
        # the samples from the original sound file.
        values.append(struct.pack(fmt[-1], raw_data[sound_index]))
        sound_index += 1

    sound_steg = wave.open(output_path, "w")
    sound_steg.setparams(params)
    sound_steg.writeframes(b"".join(values))
    sound_steg.close()
    print("Data hidden over {} audio file".format(output_path))

def recover_data(sound_path, output_path, num_lsb, bytes_to_recover):
    # Recover data from the file at sound_path to the file at output_path
    global sound, n_frames, n_samples, fmt, smallest_byte
    prepare(sound_path)

    # Put all the samples from the sound file into a list
    raw_data = list(struct.unpack(fmt, sound.readframes(n_frames)))
    # Used to extract the least significant num_lsb bits of an integer
    mask = (1 << num_lsb) - 1
    output_file = open(output_path, "wb+")

    data = bytearray()
    sound_index = 0
    buffer = 0
    buffer_length = 0
    sound.close()

    while (bytes_to_recover > 0):

        next_sample = raw_data[sound_index]
        if (next_sample != smallest_byte):
            # Since we skipped samples with the minimum possible value when
            # hiding data, we do the same here.
            buffer += (abs(next_sample) & mask) << buffer_length
            buffer_length += num_lsb
        sound_index += 1

        while (buffer_length >= 8 and bytes_to_recover > 0):
            # If we have more than a byte in the buffer, add it to data
            # and decrement the number of bytes left to recover.
            current_data = buffer % (1 << 8)
            buffer >>= 8
            buffer_length -= 8
            data += struct.pack('1B', current_data)
            bytes_to_recover -= 1

    output_file.write(bytes(data))
    output_file.close()
    print("Data recovered to {} text file".format(output_path))

try:
    opts, args = getopt.getopt(sys.argv[1:], 'hrs:d:o:n:b:',
                               ['hide', 'recover', 'sound=', 'data=',
                                'output=', 'nlsb=', 'bytes=', 'help'])
except getopt.GetoptError:
    print_usage()
    sys.exit(1)

hiding_data = False
recovering_data = False

for opt, arg in opts:
    if opt in ("-h", "--hide"):
        hiding_data = True
    elif opt in ("-r", "--recover"):
        recovering_data = True
    elif opt in ("-s", "--sound"):
        sound_path = arg
    elif opt in ("-d", "--data"):
        file_path = arg
    elif opt in ("-o", "--output"):
        output_path = arg
    elif opt in ("-n", "--nlsb"):
        num_lsb = int(arg)
    elif opt in ("-b", "--bytes"):
        bytes_to_recover = int(arg)
    elif opt in ("--help"):
        print_usage()
        sys.exit(1)
    else:
        print("Invalid argument {}".format(opt))

try:
    if (hiding_data):
        hide_data(sound_path, file_path, output_path, num_lsb)
    if (recovering_data):
        recover_data(sound_path, output_path, num_lsb, bytes_to_recover)
```

Encoder.py

```
import math

from PIL import Image

# Convert encoding data into 8-bit binary ASCII
def generateData(data):
    newdata = []
    for i in data: # list of binary codes of given data
        newdata.append(format(ord(i), '08b'))
    return newdata

# Pixels modified according to encoding data in generateData
def modifyPixel(pixel, data):
    datalist = generateData(data)
    lengthofdata = len(datalist)
    imagedata = iter(pixel)
    for i in range(lengthofdata):
        # Extracts 3 pixels at a time
          pixel = [value for value in imagedata.__next__()[:3] + imagedata.__next__()[:3] +
imagedata.__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 pixel[j]% 2 != 0):
                pixel[j] -= 1
            elif (datalist[i][j] == '1' and pixel[j] % 2 == 0):
                if(pixel[j] != 0):
                    pixel[j] -= 1
                else:
                    pixel[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 == lengthofdata - 1):
            if (pixel[-1] % 2 == 0):
                if(pixel[-1] != 0):
                    pixel[-1] -= 1
                else:
                    pixel[-1] += 1
        else:
            if (pixel[-1] % 2 != 0):
                pixel[-1] -= 1
        pixel = tuple(pixel)
        yield pixel[0:3]
        yield pixel[3:6]
        yield pixel[6:9]

def encoder(newimage, data):
    w = newimage.size[0]
    (x, y) = (0, 0)
    for pixel in modifyPixel(newimage.getdata(), data):

        # Putting modified pixels in the new image
        newimage.putpixel((x, y), pixel)
        if (x == w - 1):
            x = 0
            y += 1
        else:
            x += 1

def encode(start, end, filename, frame_loc):
    total_frame = end - start + 1
    try:
        with open(filename) as fileinput: # Store Data to be Encoded
            filedata = fileinput.read()
    except FileNotFoundError:
        print("\nFile to hide not found! Exiting...")
        quit()
    datapoints = math.ceil(len(filedata) / total_frame) # Data Distribution per Frame
    counter = start
    print("Performing Steganography...")
    for convnum in range(0, len(filedata), datapoints):
        numbering = frame_loc + "\\" + str(counter) + ".png"
            encodetext  =  filedata[convnum:convnum+datapoints]  #  Copy  Distributed  Data  into
Variable
        try:
            image = Image.open(numbering, 'r')
        except FileNotFoundError:
            print("\n%d.png not found! Exiting..." % counter)
            quit()
        newimage = image.copy() # New Variable to Store Hiddend Data
        encoder(newimage, encodetext) # Steganography
        new_img_name = numbering # Frame Number
          newimage.save(new_img_name,  str(new_img_name.split(".")[1].upper()))  #  Save  as  New
Frame
        counter += 1
    print("Complete!\n")

# Runtime
while True:
    try:
        print("Please Enter Start and End Frame where Data will be Hidden At")
        frame_start = int(input("Start Frame: "))
        frame_end = int(input("End Frame: "))
        if frame_start < frame_end:
            break
        else:
            print("\nStarting Frame must be larger than ending Frame! Please try again...")
    except ValueError:
        print("\nInteger expected! Please try again...")
frame_location = input("Frames Location: ")
filename = input("File to Hide (inc. extension): ")
encode(frame_start, frame_end, filename, frame_location)
```

Decoder.py

```
import re

from PIL import Image

# Global Variable
global frame_location

# Decode the data in the image
def decode(number):
    data = ''
    numbering = str(number)
    decoder_numbering = frame_location + "\\" + numbering + ".png"
    image = Image.open(decoder_numbering, 'r')
    imagedata = iter(image.getdata())
    while (True):
           pixels = [value for value in imagedata.__next__()[:3] + imagedata.__next__()[:3] +
imagedata.__next__()[:3]]
        # string of binary data
        binstr = ''
        for i in pixels[:8]:
            if (i % 2 == 0):
                binstr += '0'
            else:
                binstr += '1'
        if re.match("[ -~]", chr(int(binstr,2))) is not None: # only decode printable data
            data += chr(int(binstr, 2))
        if (pixels[-1] % 2 != 0):
            return data

# Runtime
print("Please Enter Start and End Frame where Data is Hidden At")
frame_start = int(input("Start Frame: "))
frame_end = int(input("End Frame: "))
frame_location = input("Frames Location: ")
print("Extracting Data...")
decodedtextfile = open('output\decoded_frame.txt', 'a')
decodedtextfile.write('Decoded Text:\n')
for convnum in range(frame_start, frame_end + 1):
    try:
        decodedtextfile.write(decode(convnum))
        print("Data found in Frame %d" % convnum)
    except StopIteration:
        print("No data found in Frame %d" % convnum)
decodedtextfile.close()
print("\nExtraction Complete!")
```

CHAPTER -7

7. PROCEDURE AND RESULTS

Step 1: Install Dependencies

Open a terminal or command prompt.

Navigate to the tool's directory.

Run the following commands to install the required dependencies:

pip install pillow

pip install tkinter

pip install opencv-python

pip install moviepy

pip install matplotlib

Step 2: Run the Tool

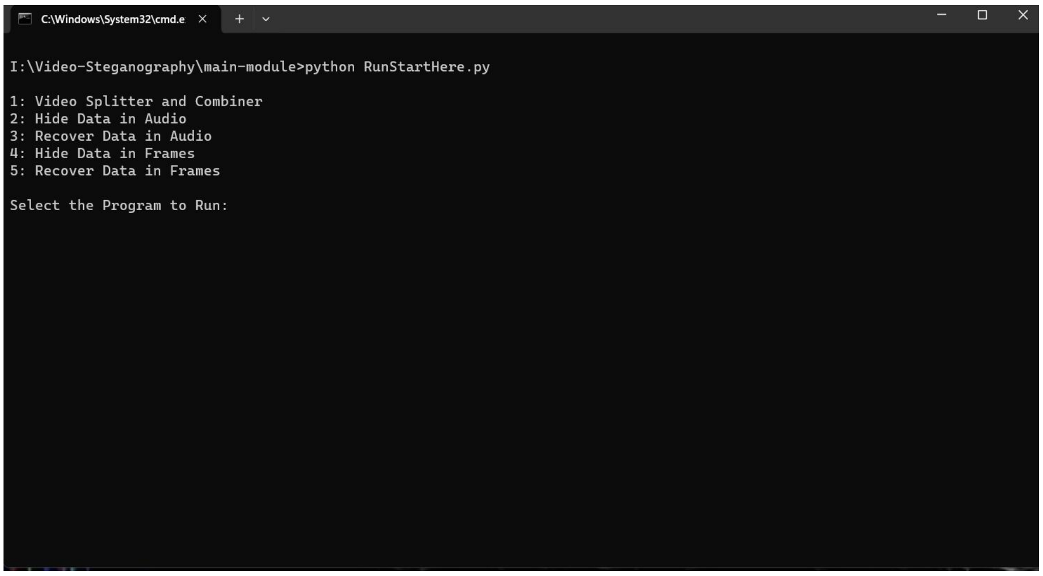
Open a terminal or command prompt.

Navigate to the directory where the Video Steganography Tool is installed.

Run the command to launch the tool, typically: python RunStartHere.py

Upon running the tool, the main menu will be displayed in the terminal.

Select the desired functionality by entering the corresponding number.



```
I:\Video-Steganography\main-module>python RunStartHere.py

1: Video Splitter and Combiner
2: Hide Data in Audio
3: Recover Data in Audio
4: Hide Data in Frames
5: Recover Data in Frames

Select the Program to Run:
```

Step 3: Splitting a Video

Choose option '1' for video splitting.

Select the 4th radio button for extracting frames.

Enter the path to the video file (e.g., C://path/filename.mp4).

Click 'Run'.

Extracted frames will be stored in /output/filename_frames/ (e.g., C://path/output/filename_frames/).

Step 4: Splitting the Audio

Choose option '1' for video splitting.

Select the 2nd radio button for extracting audio.

Enter the path to the video file (e.g., C://path/filename.mp4).

Click 'Run'.

Extracted audio will be stored as /output/filename_audio.wav (e.g., C://path/output/filename_audio.wav).

Step 5: Combining Video and Audio

Choose option '2' for video combining.

Select the 3rd radio button for combining video frames and audio.

Enter the paths to the video frames, audio file, and the original video file.

Click 'Run'.

Combined video will be stored as /output/combined_video_audio.mkv
(e.g., C://path/output/combined_video_audio.mkv).

Step 6: Hiding Data in Audio

Choose option '3' for hiding data in the audio file.

Enter the path of the text file to be hidden and the audio file.

Enter the LSB bits to hide data at (e.g., 3).

The new audio file with hidden data will be stored as /output/steg_audio.wav
(e.g., C://path/output/steg_audio.wav).

Step 7: Recovering Data from Audio

Choose option '4' for recovering data from the audio file.

Enter the path of the audio file to recover from.

Enter the LSB bits (e.g., 3) and the number of bytes used to store the data (obtained during hiding).

Retrieved data will be stored as /output/decoded_audio.txt (e.g., C://path/output/decoded_audio.txt).

Step 8: Hiding Data in Frames

Choose option '5' for hiding data in video frames.

Enter the initial and last frame to hide the data.

Enter the path to the frames folder and the text file to hide.

The tool will inform that the data has been hidden.

Step 9: Recovering Data from Frames

Choose option '6' for recovering data from video frames.

Enter the initial and last frame where the data is hidden.

Enter the path to the frames folder.

Retrieved data will be stored as /output/decoded_frames.txt (e.g., C://path/output/decoded_frames.txt).

Step 10: Explore Output

Check the /output folder for the results of your operations.

The output folder will contain various files created during the tool's execution.

CHAPTER -8

.

8. CONCLUSION

The Video Steganography Tool presents a powerful synthesis of theoretical steganographic principles and practical implementation. Utilizing the Least Significant Bit (LSB) technique within a spatial domain, the tool ensures efficient and imperceptible data concealment in video files. The incorporation of Python and key libraries such as Pillow, Tkinter, OpenCV, and Moviepy establishes a robust foundation for multimedia processing. The user-friendly graphical interface enhances accessibility, catering to a diverse user base.

The modular design of the tool, encompassing video manipulation, data hiding, reflects a systematic and versatile approach. With features like data recovery, LSB visualization, and frame detection, the tool addresses a spectrum of user needs in digital forensics and secure information exchange. The provided user guide ensures a seamless and intuitive user experience.

In essence, the Video Steganography Tool stands as a testament to the evolving landscape of information security solutions. Its commitment to efficiency, user-friendliness, and adaptability positions it as a valuable asset for both security professionals and enthusiasts navigating the intricate realm of digital information concealment.

CHAPTER -9

9. REFERENCES

Saurabh Singh and Gaurav Agarwal, “Hiding image to video: a new approach of LSB replacement”, International Journal of Engineering Science and Technology, Vol. 2(12), pp. 6999-7003, 2010.

M Abhilash Reddy, P. Sanjeeva Reddy and GS Naveen Kumar, “DWT and LSB algorithm base image hiding in a video”, International Journal of Engineering Science & Technology, Vol.3, Issue 4, pp.170-175,2013

N. Provos and P. Honeyman, "Hide and seek: an introduction to steganography", IEEE Security & Privacy Magazine, Vol. 1, Issue 3, pp. 32-44, June 2003.

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

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