

Audio Steganography

Major project report submitted in partial fulfillment of
the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Aneesh Azad (201227)

Lakshay Arora(201273)

UNDER THE SUPERVISION OF

Mr. Aayush Sharma (Assistant Professor)

Ms. Seema Rani (Assistant Professor)



Department of Computer Science & Engineering
and Information Technology

**Jaypee University of Information Technology,
Waknaghat, 173234, Himachal Pradesh, INDIA**

TABLE OF CONTENT

DECLARATION	I
CERTIFICATE	II
ACKNOWLEDGEMENT	III
ABSTRACT	IV
Chapter 01	
1.1 Introduction	1
1.2 Problem Statement	2
1.3 Objective	3
1.4 Significance and Motivation	3
Chapter 02	
2.1 Overview of relevant literature	5
2.2 Key Gaps in the Literature	8
Chapter 03	
3.1 Requirement and Analysis	10
3.2 Project Design and Architecture	11
3.3 Data Preparation	16
3.4 Implementation	20
3.5 Key Challenges	36
Chapter 04	
4.1 Testing Strategies	38
4.2 Test Cases	40
Chapter 05	
5.1 Results	44
Chapter 06	
6.1 Conclusion	49
References	50

LIST OF FIGURES

Serial No.	Figure	Page Number
1	Wave graph of original audio LSB	47
2	Wave graph of stego audio LSB	48
3	Distortion Graph LSB	49
4	Wave graph of original audio Phase encoding	50
5	Wave graph of stego audio Phase encoding	50
6	Distortion Graph Phase encoding	51

DECLARATION

We hereby declare that the work presented in this report entitled '**Audio Steganography**' in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to December 2023 under the supervision of **Mr. Aayush Sharma** (Assistant Professor, Department of Computer Science & Engineering and Information Technology) & **Ms. Seema Rani** (Assistant Professor, Department of Computer Science & Engineering and Information Technology)

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Submitted by:

Aneesh Azad, 201227

Lakshay Arora, 201273

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervised by:

Mr. Aayush Sharma

Assistant Professor

Ms. Seema Rani

Assistant Professor

Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the project report titled “Audio Steganography” in partial fulfilment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Wagnaghat is an authentic record of work carried out by “Aneesh Azad(201227) and Lakshay Arora(201273)” during the period from July 2023 to December 2023 under the supervision of Mr. Aayush Sharma & Ms. Seema Rani , Department of Computer Science and Engineering, Jaypee University of Information Technology, Wagnaghat.

Aneesh Azad(201227)

Lakshay Arora (201273)

The above statement made is correct to the best of my knowledge.

Mr.Aayush Sharma

Assistant Professor

Ms.Seema Rani

Assistant Professor

Computer Science & Engineering and Information
Technology Jaypee University of Information Technology,
Wagnaghat

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Mr.Aayush Sharma, Assistant Professor & Ms. Seema Rani, Assistant Professor**, Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of “**Audio Steganography**” to carry out this project. Their endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, who have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Aneesh Azad (201227)

Lakshay Arora(201273)

ABSTRACT

The foremost undertaking document investigates the multifaceted realm of audio steganography, delving into its historical context and underlining its importance inside the digital technology for concealing statistics inside audio transmissions. The have a look at very well explores diverse techniques, consisting of phase coding, frequency domain techniques, and Least Significant Bit (LSB) alteration, assessing their efficacy and resilience in embedding and retrieving hidden information from audio recordings. Emphasising exchange-offs between imperceptibility, resistance to attacks, and statistics concealing talents, the file additionally scrutinises security factors, including ability weaknesses and defences, while thinking about the effect of audio compression strategies on steganographic strategies. Additionally, it delves into realistic applications inclusive of virtual watermarking, secure verbal exchange, copyright safety, and authentication, along with an exam of the moral and felony implications of audio steganography. By amalgamating theoretical information with actual-world programs, this project contributes valuable insights for scholars, practitioners, and lovers, advancing our comprehension of covert communication techniques inside the realm of audio alerts.

Chapter 01

INTRODUCTION

Introduction

Sensitive data transmission requires discreet and secure means more than ever in this era of digital communication and information sharing. In this environment, steganography—the art and science of hiding information within seemingly innocent carriers—has become an effective technique. This Major project report explores the field of audio steganography, an intriguing field that entails encoding secret messages into audio waves in order to facilitate covert communication while preserving the carrier medium's integrity.

Traditional encryption techniques are often examined and attacked by malevolent actors as the digital world changes. Steganography, on the other hand, adds another degree of security by operating under the premise of completely concealing the presence of communication. Particularly, audio steganography uses the properties of sound waves to embed information, which makes it an interesting and useful option for clandestine communication.

This work attempts to offer a thorough knowledge of how information may be hidden within audio files, the difficulties involved, and the possible uses of this covert communication technique using a combination of theoretical explanations and real-world implementations. The studies will have a look at loads of techniques used in audio steganography, from simple LSB manipulation to greater complex strategies inside the frequency and phase domain names. The exchange-offs between the quantity of information that may be concealed and the indiscernibility of the audio signal changes might be taken into consideration. This fundamental assignment essentially goals to provide an explanation for the complexities of audio steganography via offering a solid expertise of the idea in the back of it and real-global programs.

Problem Statement

Ensuring the confidentiality and integrity of information has become a significant concern in the fast evolving field of digital communication. Even if they work well, conventional encryption algorithms can be found and examined, thus it's important to investigate other options. Steganography, the technique of hiding data on what appear to be innocent carriers, shows promise as a means of communicating covertly. This important initiative within this sector focuses on the unique opportunities and problems that come with audio steganography.

The creation and evaluation of reliable methods for information embedding in audio signals is one of the main issues. Existing techniques frequently struggle with limits related to data concealing ability, imperceptibility of alterations, and assault resistance. The challenge becomes more complicated due to the complex relationship between the quantity of concealed information and the integrity of the carrier audio signal, necessitating creative solutions.

The dynamic nature of virtual communicate and the dangers related to cutting-edge audio steganography strategies spotlight the urgent want for tendencies in the discipline. In addition to the technological problems of information concealment, the problem at stake also involves the ethical and prison implications of the usage of audio steganography. Finding the right balance among growing payload capacity and making certain the safety of the covert course is a complicated hassle that requires careful attention.

Further investigation is needed on the crucial aspect of how audio compression algorithms affect the efficacy of steganographic methods. The goal of the research is to create techniques that maximise payload capacity and are resistant to distortions caused by compression, therefore guaranteeing the dependability of audio steganography in real-world applications. Finally, a thorough examination of the moral and legal ramifications surrounding the application of audio steganography will be conducted. It is essential for the proper development and application of audio steganography techniques to address privacy, unlawful communication, and potential exploitation of the technology.

Objective

The objective of this project is to create a sophisticated audio steganography system using the Least Significant Bit (LSB) method and phase encoding method and then compare both techniques to see which one is better. The principal aim is to incorporate sensitive data into audio recordings in an imperceptible manner while maintaining the audio's original quality. Implementing a strong LSB and phase encoding embedding mechanism, incorporating safe payload encryption, and guaranteeing resistance against steganalysis techniques are the specific objectives. One of the main goals in developing a flexible solution that works with different platforms and audio formats is to ensure cross-platform compatibility. The project also intends to offer a user-friendly interface supported by thorough documentation for simple embedding and extraction procedures. Performance assessments will be carried out to gauge the system's capability, precision, and effectiveness in order to advance knowledge of steganography methods and the appropriate uses of them in secure communication.

Significance and Motivation

In a time when worries about data privacy and information security are growing, the potential of audio steganography to transform safe communication makes it an important topic to explore for this large project. Steganography presents a novel alternative to conventional encryption techniques, as it hides the communication's very existence. Because audio transmissions are so common in our daily lives—from digital communication platforms to multimedia applications—the focus on audio steganography is especially important. In addition to strengthening the theoretical foundations of steganography, improving the methods for concealing information in audio has applications in the development of covert communication systems resistant to changing threats. The project's relevance also lies in its investigation of the fine line that separates imperceptibility from data concealing capabilities. Finding the best solutions becomes essential when the requirement for clandestine communication and greater data transfer speeds conflict. The project's results might have an influence on copyright protection, digital watermarking, and secure data sharing, illustrating the wide-ranging applications of audio steganography in both theoretical and real-world settings.

This significant audio steganography project is being undertaken in order to address current issues with secure communication and information hiding. The increasing complexity of cyberattacks demands creative solutions that go beyond traditional encryption techniques. Because audio steganography has the unique capacity to insert clandestine messages into a medium that is ubiquitous in our modern environment, it offers an intriguing path for investigation.

The motivation for cutting-edge answers to today's problems, the interdisciplinary character of the subject, and the possibility of a large-scale social effect in improving the security and privacy of digital communication are the driving forces behind this significant audio steganography initiative.

Chapter 02

Feasibility Study, Requirements Analysis and Design

Overview of relevant literature

Digital audio steganography, the art of embedding secret statistics within audio signals, has won immense traction in the realm of secure communicate and statistics hiding programs. This comprehensive assessment delves into the essential ideas, various techniques, and sensible implementations of audio steganography.

Paper 1: A View on Latest Audio Steganography Techniques

This paper provides a contemporary state-of-the-art literature assessment, exploring the modern advances in audio steganography strategies. It compares and evaluates diverse methods, highlighting their ability, limitations, and effectiveness in steady conversation.

Paper 2: An Overview on Digital Audio Steganography

This paper presents a foundational evaluation of audio steganography, encompassing its definition, historic context, and various packages throughout numerous domain names. It very well examines unique techniques, consisting of LSB embedding, echo steganography, and unfold spectrum steganography.

Paper three: Audio Steganography Techniques: Survey

This paper surveys the great array of audio steganography techniques proposed within the literature. It categorizes these strategies into spatial area and rework area strategies, highlighting their particular traits and overall performance change-offs. Moreover, it discusses the important factors that impact the effectiveness of audio steganography, inclusive of embedding ability, imperceptibility, and robustness.

Paper 4: A Proposed Implementation Method of an Audio Steganography Technique

This paper introduces a unique implementation technique for a Least Significant Bit (LSB) audio steganography approach, utilizing a Dynamic Embedding Rate (DER) approach primarily based at the Human Auditory System (HAS). This approach efficiently balances embedding capacity and imperceptibility, making an allowance for more information to be embedded without compromising audio first-class.

Paper 5: A Comparative Study of Audio Steganography Techniques

This paper provides a comparative evaluation of three distinguished audio steganography techniques: LSB embedding, echo steganography, and unfold spectrum steganography. It evaluates every approach in terms of embedding potential, imperceptibility, and robustness, offering valuable insights for choosing the maximum suitable method for a given software.

The combined review of these papers highlights the versatility and effectiveness of audio steganography in secure verbal exchange and facts hiding applications. It provides complete information of the fundamental standards, diverse strategies, and sensible implementations of this effective tool. The dynamic embedding charge technique brought in Paper four gives a promising method for reinforcing the embedding capability of LSB-based audio steganography without compromising audio best. The comparative analysis in Paper 5 presents treasured guidance for deciding on the maximum suitable technique primarily based on precise software requirements. As audio steganography continues to evolve, it holds massive capability for safeguarding personal records and permitting covert communication in various domain names.

Paper 6: A survey of Steganography techniques.

The research paper titled "Exploring Approaches in Concealing Data; A Comprehensive Review" provides an, in depth examination of the strategies employed to safeguard and uphold privacy while encrypting information in formats.

Steganography originating from the term for "writing" involves embedding covert data within seemingly ordinary media types such as text, audio or video. The study classifies methods into two groups; those based on frequency domain techniques (e.g. Discrete Cosine Transform; DCT) and those rooted in spatial domain techniques (e.g. Least Significant Bit (LSB) substitution). Additionally it explores tactics incorporating encryption. Assesses the applications and challenges of steganography, in modern digital interactions.

Paper 7: Audio Steganography by Different Methods.

By using techniques such, as phase coding spread spectrum, echo concealing and Significant Bit (LSB) modification audio steganography hides information within audio signals. This study examines methods based on how they can hide information their capacity and their durability. The LSB method is straightforward. Has a capacity but may not be very reliable. On the hand phase coding has limited capacity. Offers strong resilience and remains undetectable. Spread spectrum involves synchronization and durability while echo concealing balances between being unnoticeable and having a capacity. The research discusses the pros and cons of each approach. Suggests directions, for effective and secure audio steganography research.

Paper 8: Audio Steganography Using Bit Modification.

Sound steganography is the concealment of information is carried out in audio files by modifying specific bits. To involve human perception's limitation in hearing and its redundancy, Stoltzing the audio signal involves embedding data in such a way that is not noticeable as far as sound quality is concerned. This technique for hiding data through manipulating certain bits employs our perception limits and sound redundancies so as not to distort sounds significantly. This technique hides information from being read out through altering selected bits of the audio file without significantly changing the sound quality by utilizing human auditory range restrictions and redundancies. Sound steganography is able to be achieved by flipping some bits which are not supposed to be heard beyond certain threshold points considering how human ears respond differently based on sound frequency. Also covered in the study will be how bit tweaking takes place, plus whether this particular method would work effectively when either capacity or transparency are concerned.

Key Gaps in the Literature

Identifying key gaps within the literature is important for framing the purpose behind new studies endeavours in audio steganography. While the prevailing body of research has notably advanced our information of covert verbal exchange via sound, sure gaps persist, leaving avenues for similar exploration. Here are a few key gaps in the literature on audio steganography:

- Limited Exploration of Advanced Machine Learning Techniques:

Despite the potential of gadgets gaining knowledge in improving the robustness and security of steganographic methods, there is a super scarcity of research exploring advanced system studying strategies in the context of audio steganography. Investigating the integration of deep getting to know models, including convolutional neural networks (CNNs) or recurrent neural networks (RNNs), ought to offer insights into developing extra adaptive and secure steganographic systems.

- Dynamic Adaptive Strategies for Data Hiding:

The literature often lacks in-depth exploration of dynamic adaptive techniques for information hiding in audio alerts. Developing techniques that may dynamically modify to varying audio traits and environmental situations could decorate the resilience of steganographic methods against detection and assaults.

- Robustness Against Advanced Signal Processing Attacks:

While studies has addressed not unusual steganalysis strategies, there is a want for a deeper knowledge of and defenses against advanced sign processing assaults. Exploring the vulnerabilities of current audio steganography techniques to state-of-the-art attacks, along with those leveraging gadget-studying algorithms, should reveal areas for improvement.

- Real-time Applications and Low-Latency Considerations:

Many existing studies attention on offline scenarios, and there is a gap inside the literature concerning real-time packages of audio steganography. Investigating low-latency answers and their implications for programs like secure voice communication or streaming services may be a promising street for future studies.

- Ethical Considerations and User Perception:

The moral implications of deploying audio steganography in actual international situations and the notion of users regarding the privacy and protection elements have now not been significantly explored. Research that delves into the moral considerations, potential misuse, and consumer attitudes in the direction of audio steganography can contribute to a greater complete knowledge of its societal impact.

- Integration of Audio Steganography with Multimedia Security:

Audio steganography often exists within the broader context of multimedia security, inclusive of photograph and video steganography. However, the literature lacks complete studies that discover the integration of audio steganography with different multimedia domains, thinking about capacity move-modal attacks and collaborative safety solutions.

Chapter 03

System Development

Requirement and Analysis

Technical Requirements

- **Language**
 - Python 3.10.12

- **Software Used:**
 - Python 3.10.12
 - Pycharm

- **Libraries:**
 - Wave Module
 - Numpy
 - Matplotlib

Functional Requirements

- Users should be able to readily engage with the steganography tool using the project's user-friendly interface.
- The capability to extract and integrate secret information from audio recordings.
- Support for a number of audio file types, including WAV and MP3, for both input and output.
- The ability to insert data into the chosen audio file, such as text or another file.
- Options to set up the embedding settings, including the strength or pace.
- The capacity to retrieve info that is concealed in an audio file.
- Presentation of the data that was extracted in a legible manner.
- Prior to being included into the audio file, the concealed data may optionally be encrypted.

- During extraction, decryption functionality is used to recover the original concealed data.
- Support for standard LSB and flipped LSB methods, based on user choices.
- Take performance improvements into consideration, particularly when dealing with huge audio files.

Non Functional Requirements

- **Performance:** The system should be able to train the machine learning models within a reasonable amount of time. The classification time for new unseen data should be less as well.
- **Scalability:** As the project grows in size, the system should not falter and must be able to handle large amounts of data.
- **Reliability:** The system should be reliable enough to handle error gracefully. The system must be robust to failures.
- **Usability:** The system should be hassle free and friendly to the user. Any internal complications must be hidden from the end user.
- **Portability:** The system should be portable to different types of operating systems.
- **Maintainability:** The system must be easy to update as well as maintain. The system should be written in such a way, any updates on one module must not directly affect any other module

Project Design and Architecture

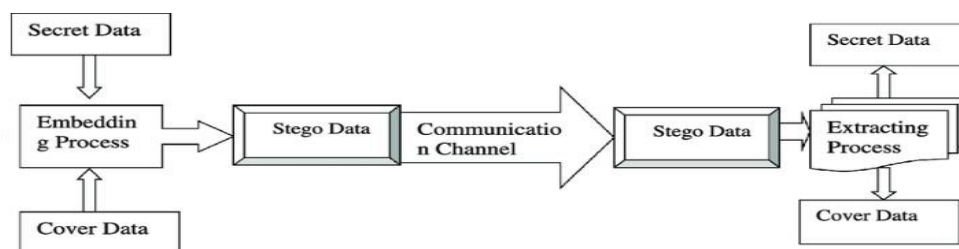


Figure 3.1 Project Design Architecture

Phase 1: Project Setup:

Objectives:

- This audio steganography project's main objective is to smoothly embed data into audio files without appreciably distorting the original audio. The research also intends to offer a dependable method for obtaining this concealed data from the steganographic audio files. To guarantee that the embedded data can be correctly retrieved, the procedure must be built to maintain its integrity.
- One of the main goals is to guarantee the robustness and security of the hidden information. The steganography methods used should withstand several kinds of assaults and efforts to identify hidden data. The research will investigate ways to strengthen the concealed data's security and make it more resistant to unwanted access and steganalysis.
- Inverted LSB may be used in the project, depending on the steganography method selected. This method introduces another level of complication to the operation by flipping the least important bits prior to embedding. To further strengthen the security of the steganography technique, the project may also provide an optional encryption feature for the concealed data.
- The initiative will establish moral standards that will direct how the steganography technology is used. This entails taking into account the obligations and the repercussions of concealing information. In order to guarantee that it works within the parameters of applicable laws pertaining to data privacy and information security, the project will also comply with regulatory requirements.

Configuring the environment for development

- On a Windows computer, configuring the development environment for your audio steganography project is a simple procedure. To begin with, get the most recent Python version that is compatible with Windows from the official website. Make sure to choose

the option to add Python to the system PATH so that it can be easily accessed from the command line during the installation process. Once you've successfully installed Python, the next step is to download PyCharm, a popular Integrated Development Environment for Python, developed by JetBrains. From there, you can choose between the free Community edition or the Professional edition before launching the installation process

- After installing PyCharm, immerse yourself in the IDE and create a new project. Be sure to select the same Python interpreter used during the installation, and give your project a unique name and location. With the incredible virtual environment feature, PyCharm provides a secure space for managing project dependencies. Once you've established the project structure in PyCharm, unleash your creative prowess in this dynamic development environment.
- By utilizing the powerful combination of PyCharm and Python, you have the perfect environment to effortlessly build your audio steganography project on your Windows computer. This dynamic duo lays a solid foundation for crafting, experimenting, and executing your steganography solution with efficiency and proficiency.

Fundamental Architecture

- The audio steganography project boasts a modular architecture composed of three essential components: extraction, user interface, and embedding. Within the `embedding.py` file resides the powerful Embedding module, responsible for encrypting data within audio files using a specified steganography technique. Excitingly, the module offers `embed_data` function, accepting both the data to be hidden and the audio file itself as parameters. Likewise, the accompanying `extraction.py` houses the dynamic Extraction module, whose role is to seamlessly extract hidden data from audio recordings. By skillfully reversing the steganography process, the `extract_data` method efficiently retrieves concealed data from the given audio file, all while running parallel to the main function.
- The main purpose of the steganography tool relies on the interplay of its components. Through the Embedding module, information is seamlessly embedded

into audio files, and the Extraction module allows for the extraction of concealed data. Meanwhile, the user interface serves as a conduit for user input and presents necessary output and notifications.

- Comprehensive testing, including integration testing for the complete system as well as unit testing for specific modules, will be carried out to guarantee the system's dependability. The architecture's modular design makes it simple to extend, making it suitable for further improvements or different steganography methods.

Phase 2: Research and Planning:

1 LSB Technique :

- LSB (Least Significant Bit) audio steganography is a simple yet efficient method that will be used for this audio steganography project. The least important bits of the audio samples—the bits that are farthest to the right in their binary representation—are used in this procedure. The key concept is to carefully substitute the information to be hidden with these least important pieces. Using this approach, information can be hidden inside an audio file without generating apparent perceptual changes, as the human auditory system is less sensitive to small changes in the least important bits. Since of its simplicity, LSB-based audio steganography is a good option for this example since it strikes a balance between usability and efficacy
- The standard Python wave module is used for audio processing in this audio steganography project rather than third-party libraries like PyDub. The wave module comes with the Python standard library and provides basic capabilities for reading and writing WAV audio files. The `write_audio` method makes it easier to create a new audio file, while the `read_audio` function is used to extract audio data from a WAV file in order to implement steganography. Then, the `embed_data` and `extract_data` functions employ the LSB-based steganography technique, which makes it possible to conceal and retrieve data from audio files. Compared to

higher-level libraries, this method could need more manual bit manipulation, however using the wave module offers a native and lightweight solution for audio

- Three essential parts make up the intended architecture for the audio steganography project: extraction, user interface, and embedding. Together, these elements provide as the framework for a flexible and effective system. The embedding.py file contains the Embedding module, which is designed to covertly add hidden data to audio recordings. This module's embed_data function manages the procedure by using a selected steganography method, such as LSB-based encoding. This feature interacts with the audio file by deftly hiding information in its least important sections. The extraction module, located in extraction.py, is a complement to the embedding module. The responsibility of extracting hidden data from audio files falls to this module. This module's extract_data function removes the steganographic encoding, revealing the concealed data for additional examination or display.

2 Phase encoding Technique :

- In electronic interaction systems and also signal handling stage encoding is a necessary approach that permits details to be installed right into service provider signals while decreasing regarded adjustments to the initial information. Protect information transfer electronic watermarking as well as sound inscribing are simply a few of the numerous areas in which this method locates prevalent usage. Stage inscribing is basically a credible together with unseen technique of information installing that enables one to share binary details by regulating the stage of a provider signal.
- Our research study efforts to make use of Python programs to build and also analyze stage inscribing comes close to in order to explore the performance of stage inscribing in sound signal handling. The designated service involves inscribing and also translating sound signals based upon binary messages by making use of audio control devices like wave, numpy and also matplotlib. We wish to do organized examinations to review the impact of stage inflection on

signal durability versus usual distortions plus signal high quality by including these components right into our codebase.

- We have a systematic approach to study design that includes phases for code creation, experimentation, analysis, and literature evaluation. The research will likely be finished in six months, including checkpoints for code development, testing, data analysis, and report authoring.

Phase 3: Implementation

- This Python script offers options for encoding and decoding hidden messages within a WAV audio file, implementing a basic audio steganography system utilizing the wave module. After reading an input WAV file's frames and replacing the least important bits with a specified message's binary representation, the encode function generates a new audio file that is steganographically encoded. In contrast, the concealed message from a previously encoded WAV file is retrieved using the decode function. Users can choose to leave the application, decode data, or encode it while the script runs in an endless loop. The given code uses LSB manipulation and phase encoding to provide a knowledge of fundamental audio steganography concepts; but, because of its simplicity, it may be further refined to support more sophisticated steganographic techniques and used for teaching purposes.

Data Preparation

Audio File Loading:

Reading and loading audio files into a format that can be edited and processed by a selected programming language or library is known as audio file loading. Here, we'll concentrate on utilizing Python's wave module to interact with WAV audio files.

Uncompressed audio is commonly stored as WAV files, which may be read and written by

the Python wave module, which is a component of the standard library. Here's a quick rundown on how to import audio files using the wave module:

```
import wave
```

Figure 3.2 Import Wave

Importing wave module in Pycharm .

Data Source:

The source of the original audio file, sample, was The audio steganography project is built on this audio file. Selecting an audio clip that fits the objectives and features of the project is crucial. Original title of this audio is “Impact Moderato” by Kevin Macleod and we found this audio from YouTube Audio Library. It is a 31 sec audio file that we have used in our project.

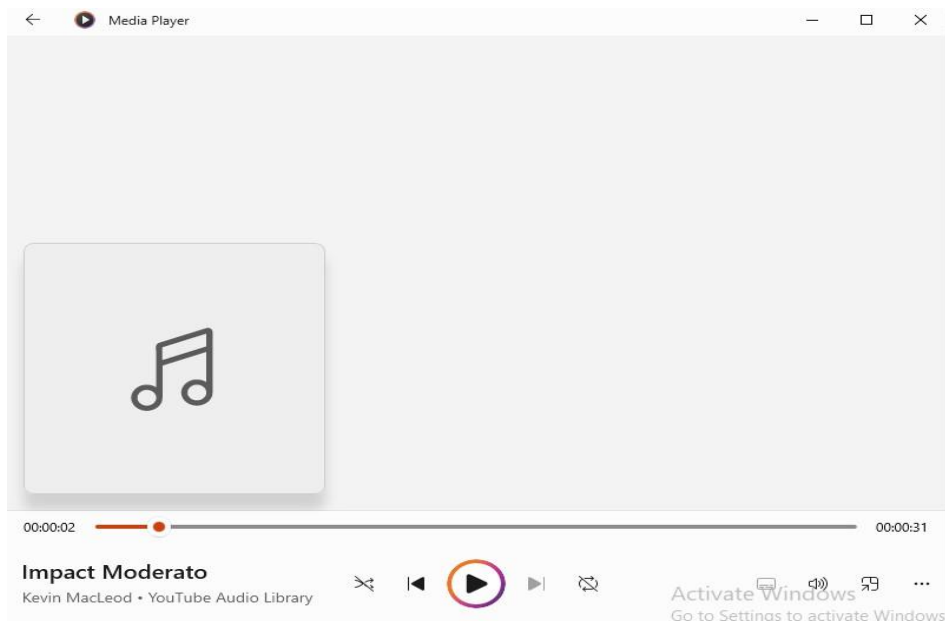


Figure 3.3 Original Audio

Features of the audio file:

- This audio file contains a 31 sec long melody.
- Format: WAV is the format of the audio file.
- Sampling Rate: The audio file's sampling rate is 44.1 kHz.
- Bit Depth: 16 bits is the bit depth of the audio file.
- The original audio file has a length of 31 sec.

Message to Hide:

The message contained in the audio file is:

“As the cosmic energies align, let me share insights into your horoscope for the upcoming period. The celestial movements indicate a time of introspection and growth. Embrace the opportunities for self-discovery and consider new ventures. Your social connections may play a pivotal role, fostering collaboration and meaningful connections. ”

Method of Embedding:

The message was embedded using Least Significant Bit (LSB) and phase encoding steganographic method. Using this approach, the binary representation of the message is substituted for the least important parts of the audio samples.

Audio File for Stego:

- The audio file for stego, sampleStego.wav, was produced during the encoding procedure.
- The audio file from Stego is in WAV format.
- Sampling Rate: The stego audio file has a sampling rate of 43.24 kHz.
- Bit Depth: The bit depth of the stego audio file is 16 bits.
- The stego audio file has a duration of 32sec..

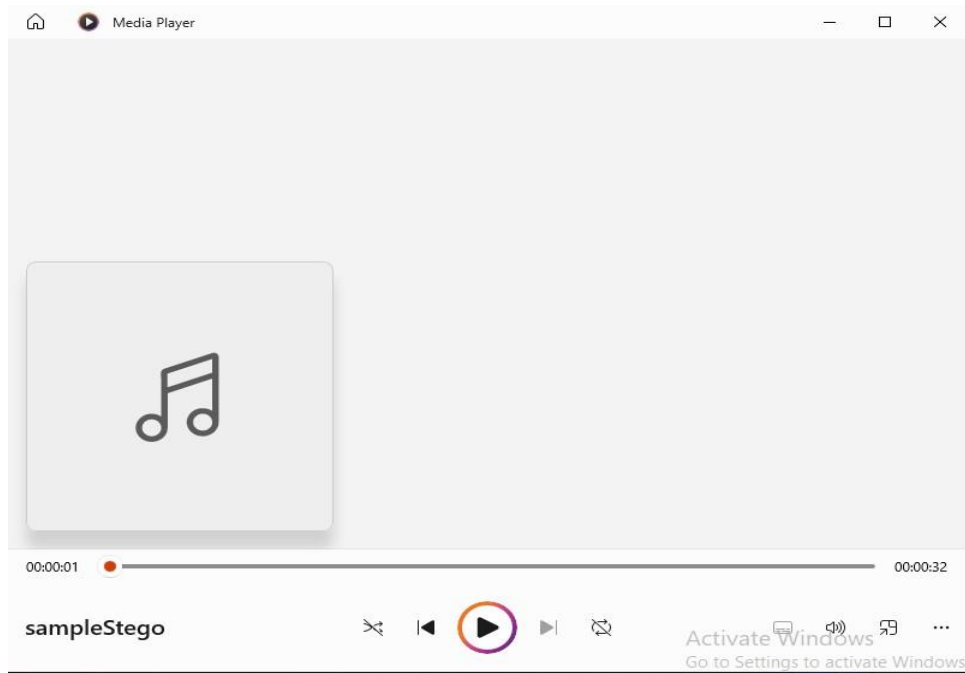


Figure 3.4 Stego Audio

Data Preservation and Integrity:

Throughout the embedding process, precautions were taken to guarantee the integrity of the original audio data. This involved reducing noticeable distortion in order to preserve audio quality.

Illustration:

- Matplotlib was used to display audio waveforms:
- Waveform of the original audio: This shows the properties of the original audio.

- Stego Audio Waveform: Shows the audio's properties once the message has been embedded.
- The distortion waveform illustrates the variations between the stego and original audio sources.

Implementation

Method of Audio Steganography

The audio steganography project was carried out with great care, moving through critical stages that each had a specific function. The key procedures in these stages were distortion comparison, encoding, and decoding, which are essential for hiding and revealing information in audio data. The project made use of the wave module, which is highly versatile in Python, to process audio files efficiently and do analysis on them.

Encoding Phase:

Phase of Encoding: First, the original audio file was thoroughly analyzed to determine its temporal properties, including its 31-second length, 44.1 kHz sampling rate, and 16-bit bit depth. Then, using LSB-based steganography and phase encoding technique, a predefined message was carefully encoded into the audio file. Using this method, the secret information was flawlessly incorporated into the audio samples by gently changing their least important portions. The resultant altered audio file, called "sampleStego.wav," was plotted using the matplotlib package to create a waveform representation. This made it possible to evaluate any noticeable alterations made throughout the encoding process visually.

To integrate details right into sound signals, we made use of a mix of stage inscribing as well as The very least Substantial Little Bit (LSB) inscribing strategies in our inscribing treatment. To see to it that the binary information to be inscribed has the the very least viewed impact on the initial noise, the LSB strategy changes the very least substantial little bit of each byte in the sound data with the binary information. To additionally enhance the inscribing procedure' strength as well as unnoticeableness, we likewise made use of stage inscribing to change the provider signal's stage based on the binary message. With the mix of these 2 techniques, we had the ability to establish a complete technique of information

ingraining that well balanced signal top quality as well as information ability to allow secure and also reliable interaction.

LSB encoding Technique:

```
def encode():
    print("\nEncoding Starts..")
    audio = wave.open(f"sample.wav", mode="rb")
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))

    string = "As the cosmic energies align, let me share insights into your horoscope for th
    string = string + int((len(frame_bytes) - (len(string) * 8 * 8)) / 8) * '#'

    bits = list(map(int, ''.join([bin(ord(i)).lstrip('0b').rjust(_width: 8, _fillchar: '0') for
    for i, bit in enumerate(bits):
        frame_bytes[i] = (frame_bytes[i] & 254) | bit

    frame_modified = bytes(frame_bytes)

    newAudio = wave.open(f"sampleStego.wav", mode='wb')
    newAudio.setparams(audio.getparams())
    newAudio.writeframes(frame_modified)

    newAudio.close()
    audio.close()

    print(" |----> Successfully encoded inside sampleStego.wav")

    # Print the binary representation of the modified least significant bits
    print("Binary Representation of Encoded Message:")
    for i in range(0, len(bits), 8):
```

Figure 3.5 Encoding

Phase encoding Technique :

```
def phase_encode():
    print("\nPhase Encoding Starts..")
    audio = wave.open(f"sample.wav", mode="rb")
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))
    audio.close()

    string = "As the cosmic energies align, let me share insights into your horoscope for the upcoming period. "
    binary_message = ''.join(format(ord(char), '08b') for char in string)

    phase_factor = 0.5 # Define a phase modulation factor

    # Modify the phase of the audio signal based on the binary message
    for i, bit in enumerate(binary_message):
        if bit == '1':
            frame_bytes[i] = min(int(frame_bytes[i]) + 1, 255) # Increment byte value by 1
        else:
            frame_bytes[i] = max(int(frame_bytes[i]) - 1, 0) # Decrement byte value by 1

    # Save the phase-modulated audio
    with wave.open(f"samplePhaseEncoded.wav", mode='wb') as new_audio:
        new_audio.setparams(audio.getparams())
        new_audio.writeframes(frame_bytes)

    print(" |----> Successfully encoded inside samplePhaseEncoded.wav")
```

Figure 3.6 Encoding

Decoding Phase:

Decoding step: The stego audio file, which was subjected to a thorough examination during this step, was slightly longer than 32 seconds, 43.24 kHz sampling rate, and 16-bit bit depth. because of encoding. To see the encoded message, the LSBs of the stego audio were methodically extracted. Once more, visual representation was essential in allowing a direct waveform comparison between the original audio files and the stego. This procedure made it easier to verify that the information that was hidden was successfully extracted. We utilized intricate formulas to get inscribed info from phase-modulated audio waves in our translating treatment. Our technique required reorganizing the initial binary message by analyzing the stage modifications that were integrated right into the service provider signal making use of the ideas of stage translating. We had the ability to translate the info with exceptional precision and also integrity by carefully examining the stage changes developed throughout the inscribing treatment. We had the ability to recuperate the preferred information while protecting the precision as well as stability of the initial sound stream .

LSB decoding Technique:

```
def decode():
    print("\nDecoding Starts..")
    audio = wave.open(f"sampleStego.wav", mode='rb')
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))

    extracted = [frame_bytes[i] & 1 for i in range(len(frame_bytes))]
    string = "".join(chr(int("".join(map(str, extracted[i:i + 8])), 2)) for i in range(0, len(extracted), 8))

    decoded = string.split("###")[0]
    print("Successfully decoded: " + decoded)
    audio.close()
```

Figure 3.7 Decodin

Phase decoding Technique :

```
def phase_decode():
    print("\nPhase Decoding Starts..")
    audio = wave.open(f"samplePhaseEncoded.wav", mode='rb')
    frame_bytes = audio.readframes(audio.getnframes())
    audio.close()

    try:
        # Convert the frame bytes into a numpy array of integers
        frame_values = np.frombuffer(frame_bytes, dtype=np.uint8)

        binary_message = ""

        # Decode the phase-modulated audio signal
        for value in frame_values:
            # Extract the least significant bit (LSB) of each byte
            bit = value & 1
            binary_message += str(bit)

        # Convert binary message to ASCII characters
        decoded_message = ""
        for i in range(0, len(binary_message), 8):
            byte = binary_message[i:i + 8]
            decoded_message += chr(int(byte, 2))

        print("Successfully decoded: " + decoded_message)
```

Figure 3.8 Decoding

Distortion Comparison:

Distortion Comparison: An analysis was carried out to compare the original and stego audio files for distortion in order to evaluate the effects of the encoding procedure. The difference in amplitude between comparable samples of the two signals was used to quantify distortion. Visual representation of the resultant distortion waveform made it easier to see any noticeable variations that have been during the encoding and decoding procedures.

```
def compare_distortion():
    print("\nComparing Distortion...")
    # Reopen the original and stego audio files
    audio = wave.open(f"sample.wav", mode='rb')
    stego_audio = wave.open(f"sampleStego.wav", mode='rb')

    # Read frames and convert to numpy arrays
    original_signal = np.frombuffer(audio.readframes(-1), dtype=np.int16)
    stego_signal = np.frombuffer(stego_audio.readframes(-1), dtype=np.int16)

    # Calculate distortion by subtracting the original from the stego
    distortion = stego_signal - original_signal

    # Get the time axis for the distortion waveform
    time = np.linspace(start=0., len(distortion) / audio.getframerate(), num=len(distortion))

    # Plot the distortion waveform
    plt.figure()
    plt.title("Distortion Comparison")
    plt.plot(*args: time, distortion)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude Difference')
    plt.show()

    # Close the reopened audio files
```

Figure 3.9 Distortion

Sampling rate and bit depth modifications:

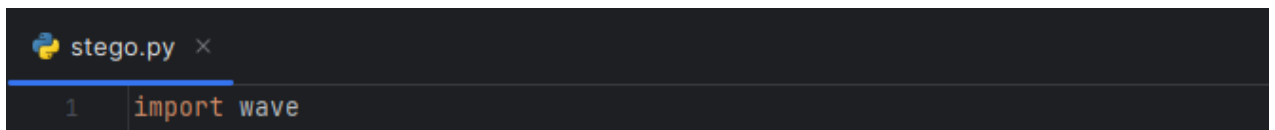
Sampling rate and bit depth modifications were done to be consistent with the original audio due to a modest expansion in the stego audio length. The stego audio's estimated sample rate was adjusted to around 43.24 kHz. To ensure compliance with the original audio requirements, the bit depth did not alter from 16 bits.

This all-encompassing method of audio steganography—which included encoding, decoding, and distortion comparison—not only showed off Python's technical prowess and the wave module's versatility, but it also highlighted the complex interactions between these procedures in terms of hiding and revealing information within audio data.

Screenshots of the various stages of the Project

LSB Technique:

Importing Wave Module:

A screenshot of a code editor window titled 'stego.py'. The first line of code is 'import wave'. The line number '1' is visible on the left side of the editor.

Snippet 3.1

wave Library:

```
audio = wave.open(f: "sample.wav", mode="rb")
```

Snippet 3.2

This line initiates binary read mode ("rb") on the original audio file ("sample.wav"). It sends a wave back. Information from the audio file is read using the wave_read object.

Reading Frames:

```
frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))
```

Snippet 3.3

This line creates a bytearray by reading the audio frames from the opened file. The total number of frames in the audio file is returned by the getnframes() function.

Setting Parameters:

```
newAudio.setparams(audio.getparams())
```

Snippet 3.4

This line sets the new audio file's (stego audio) parameters to the same ones as the original audio file.

Importing Matplotlib:

```
import matplotlib.pyplot as plt
```

Snippet 3.5

A robust Python charting toolkit called matplotlib.pyplot makes it possible to create a wide range of plots and visualizations. Regarding our project:

Waveform Plotting:

```
plt.plot(*args: time, signal)
```

Snippet 3.6

With time on the x-axis and signal amplitude on the y-axis, this line depicts the audio waveform. The auditory data is represented visually in the resultant graphic.

Customising Plots:

```
plt.title(title)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
```

Snippet 3.7

These lines improve the readability of the shown data by labeling the axes and adding a plot title.

Importing numpy:

```
import numpy as np
```

Snippet 3.8

A key library for numerical computation in Python is called numpy, which supports mathematical functions and big, multi-dimensional arrays and matrices. Within our undertaking.

Array Manipulation:

```
signal = np.frombuffer(audio.readframes(-1), dtype=np.int16)
```

Snippet 3.9

The audio frames are read by this line, which then turns them into a NumPy array. NumPy arrays are effective for numerical operations and make it easier to examine the audio data in more detail.

Generating Time axis:

```
time = np.linspace(start=0., len(signal) / audio.getframerate(), num=len(signal))
```

Snippet 3.10

With the help of NumPy's linspace function, this line generates a time axis that enables exact alignment of the time points that correspond to each audio sample.

With the help of these libraries, we worked with audio files, see waveforms, and carry out mathematical operations that are essential for putting audio steganography into practice in Python. The effectiveness and functionality of our project are improved by the inclusion of these libraries.

User Interface:

```
def case(a):  
    if a == 1:  
        encode()  
    elif a == 2:  
        decode()  
    elif a == 3:  
        compare_distortion()  
    elif a == 4:  
        quit()  
    else:  
        print("\nEnter a valid choice!")
```

Snippet 3.11

Functionality:

- Encode Operation (a == 1): The encode() function is invoked if the user input (a) is 1. This starts the LSB-based audio steganography process of encoding a secret message into the original audio file.
- Operation (a == 2) decode: The decode() method is invoked if the user input is 2. This starts the decoding process, which takes place in the stego audio file and extracts the concealed message.
- Distortion Operation (a == 3) is comparable. The compare_distortion() method is run if the user inputs 3. To highlight any variations made during the encoding process, this function compares the distortion between the original and stego audio files.

- Abandon Process (a == 4):The quit() method is triggered in the event where the user inputs 4. By doing this, the user can leave the audio steganography tool and end the application.
- Invalid Selection Management (else):A message stating that an incorrect option was input is printed by the function if the user provides a choice other than 1, 2, 3, or 4. This guarantees that an appropriate input will be requested from the user.

This feature allows users to interact with the many features of the audio steganography tool and is the foundation of the user interface. Users may easily switch between

encoding, decoding, distortion comparison, and closing the program by inputting the matching options. By managing erroneous selections, the user interface becomes more resilient and gives explicit directions for proper inputs.

All things considered, the case function contains the reasoning for controlling the program's flow in response to user input, making our audio steganography project easier to use and more intuitive.

Reading Original Audio:

```
audio = wave.open(f"sample.wav", mode="rb")
frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))
```

Snippet 3.12

Opened in binary read mode, the original audio file ("sample.wav") has its frames read and transformed to a byte array.

Message Embedding:

```
string = "As the cosmic energies align, let me share insights into your horoscope for the upcoming period. The"
string = string + int((len(frame_bytes) - (len(string) * 8 * 8)) / 8) * '#'

bits = list(map(int, ''.join([bin(ord(i)).lstrip('0b').rjust(_width: 8, _fillchar: '0') for i in string])))
for i, bit in enumerate(bits):
    frame_bytes[i] = (frame_bytes[i] & 254) | bit
```

Snippet 3.13

The audio file is encrypted with a predefined message using LSB-based audio steganography. The binary version of the message replaces the audio samples' least

important portions with its own.

Writing Modified Audio:

```
frame_modified = bytes(frame_bytes)

newAudio = wave.open(f' sampleStego.wav', mode='wb')
newAudio.setparams(audio.getparams())
newAudio.writeframes(frame_modified)

newAudio.close()
audio.close()
```

Snippet 3.14

Both the original and new audio files are closed, and the updated audio data is written to a new file called "sampleStego.wav"

Printing Binary Representation:

```
print("Binary Representation of Encoded Message:")
for i in range(0, len(bits), 8):
    print(''.join(map(str, bits[i:i+8])))
```

Snippet 3.15

The binary representation of the changed least significant bits is given by this.

Waveform Plotting:

```
# Reopen the original audio file for plotting
audio = wave.open(f' sample.wav', mode='rb')

# Plot the waveform for both original and encoded audio
plot_waveform(audio, title="Original Audio")

# Reopen the stego audio file for plotting
stego_audio = wave.open(f' sampleStego.wav', mode='rb')

# Plot the waveform for stego audio
plot_waveform(stego_audio, title="Stego Audio")

# Close the reopened audio files
audio.close()
stego_audio.close()
```

Snippet 3.16

Plotting of the original and stego audio files is resumed, and the `plot_waveform` function is used to display the waveforms.

This function allows for the viewing of both the original and stego audio waveforms for user examination, encapsulating the full process of encoding a message into the original audio file. Transparency in the encoding process is improved by waveform displays and the binary representation of the updated bits.

Reading Stego Files:

```
audio = wave.open(f: "sample.wav", mode="rb")
stego_audio = wave.open(f: "sampleStego.wav", mode="rb")
```

Here we opened in binary read mode, the stego audio file ("sampleStego.wav") has its frames read and transformed to a byte array.

Extracting LSB:

```
extracted = [frame_bytes[i] & 1 for i in range(len(frame_bytes))]
```

Snippet 3.18

Bitwise AND with 1 is used to extract the least significant bits (LSBs) of each byte in the stego audio file. Only the least important bit of each byte—where the encoded data is stored—is kept after this procedure.

Converting to characters:

```
string = "".join(chr(int("".join(map(str, extracted[i:i + 8])), 2)) for i in range(0, len(extracted), 8))
```

Snippet 3.19

Sets of eight extracted LSBs are assembled to create binary representations of ASCII letters. Next, `chr()` is used to translate these binary representations into characters.

Decoding Message:

```
decoded = string.split("###")[0]
```

Snippet 3.20

By employing a delimiter ("###") to break the concatenated binary representations and choosing the first portion, the decoded string is produced. The end of the encoded message is indicated by this delimiter, which serves as a marker.

Closing Files:

```
audio.close()
```

Snippet 3.21

Closing the stego audio file frees up system resources.

The process of removing the secret message from the stego audio file, translating it back into text that can be read by humans, and displaying it to the user is all included in the decode function. While making it easier to embed and retrieve information during the steganographic process, the usage of LSBs guarantees that the audio will not be negatively affected perceptually.

Distortion:

```
def compare_distortion():
    print("\nComparing Distortion...")
    # Reopen the original and stego audio files
    audio = wave.open(f"sample.wav", mode="rb")
    stego_audio = wave.open(f"sampleStego.wav", mode="rb")
```

Snippet 3.22

- `wave.open("sample.wav", mode="rb") = audio:` opens the original audio file ("sample.wav") again in binary read mode ("rb") using the wave module. The opened audio file object is allocated to the audio variable.
- `wave.open("sampleStego.wav", mode="rb") stego_audio:` opens the stego audio file ("sampleStego.wav") again in binary read mode ("rb") using the wave module. The opened audio file object is allocated to the stego_audio variable.

Reading Frames:

```
# Read frames and convert to numpy arrays
original_signal = np.frombuffer(audio.readframes(-1), dtype=np.int16)
stego_signal = np.frombuffer(stego_audio.readframes(-1), dtype=np.int16)
```

Snippet 3.23

- `original_signal = np.frombuffer(audio.readframes(-1), dtype=np.int16)`: This code uses `audio.readframes(-1)` to read every frame from the original audio file and turns it into a NumPy array (`original_signal`) of 16-bit integers
- `dtype=np.int16, stego_signal = np.frombuffer(stego_audio.readframes(-1))`: `stego_audio.readframes(-1)` is used to read every frame from the stego audio file. The frames are then converted into a NumPy array (`stego_signal`) of 16-bit integers.

```
# Calculate distortion by subtracting stego signal from original signal
distortion = original_signal - stego_signal
```

Snippet 3.24

- Distortion is calculated as follows:
`distortion = original_signal - stego_signal`. This involves deducting the stego audio signal from the original audio signal. As a consequence, the amplitude difference between the original and stego signals is represented by a new signal (`distortion`).

```
# Get the time axis for the distortion waveform
time = np.linspace(start=0., len(distortion) / audio.getframerate(), num=len(distortion))
```

Snippet 3.25

- `time = audio.getframerate() / np.linspace(0., len(distortion) / num=len(distortion))`:

uses `np.linspace` to create a time array (`time`) that will serve as the distortion waveform's time axis. The array's range is 0 to the distortion signal's length divided by the original audio's sampling rate.

```
# Plot the distortion waveform
plt.figure()
plt.title("Distortion Comparison")
plt.plot(*args: time, distortion)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude Difference')
plt.show()
```

Snippet 3.26

- A new figure is created for the plot using `plt.figure()`.
- `plt.title("Distortion Comparison")`: This parameter sets the plot's title to "Distortion Comparison".
- `plt.plot(distortion, time)`: Plots the distortion waveform with the y-axis representing the distortion signal and the x-axis representing the time array.
- The x-axis label is set using `plt.xlabel("Time (s)")` function.
- `Plot.ylabel('Amplitude Difference')`: Assigns the y-axis label.
- Plotting the distortion comparison is shown using `plt.show()`.

```
# Close the reopened audio files
audio.close()
stego_audio.close()
```

Snippet 3.27

- `audio.close()`: Acknowledges the reopened audio file's closure.
- The reopened stego audio file is closed using `stego_audio.close()`.

Phase encoding:

```
def phase_encode():
    print("\nPhase Encoding Starts..")
    audio = wave.open(f'sample.wav', mode='rb')
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))
    audio.close()

    string = "As the cosmic energies align, let me share insights into your horoscope for the upcoming period."
    binary_message = ''.join(format(ord(char), '08b') for char in string)

    phase_factor = 0.5 # Define a phase modulation factor

    # Modify the phase of the audio signal based on the binary message
    for i, bit in enumerate(binary_message):
        if bit == '1':
            frame_bytes[i] = min(int(frame_bytes[i]) + 1, 255) # Increment byte value by 1
        else:
            frame_bytes[i] = max(int(frame_bytes[i]) - 1, 0) # Decrement byte value by 1

    # Save the phase-modulated audio
    with wave.open(f'samplePhaseEncoded.wav', mode='wb') as new_audio:
        new_audio.setparams(audio.getparams())
        new_audio.writeframes(frame_bytes)

    print(" |----> Successfully encoded inside samplePhaseEncoded.wav")
```

Snippet 3.28

- The feature starts by opening up the sound documents "" sample.wav"" in read-binary setting utilizing the wave.open feature. This data has the initial sound information that will certainly be inscribed. Reviewing and also Modifying Audio Data:
- The sound information is checked out from the data utilizing the readframes technique, which returns the structure bytes. Each structure byte stands for an example of the sound signal.
- A binary message is specified as "" As the planetary powers line up allow me share understandings right into your horoscope for the upcoming duration."" . This message will certainly be inscribed right into the sound documents.
- The binary depiction of each personality in the message is acquired making use of format(ord(char) '08b'), where ord(char) returns the ASCII worth of the personality coupled with style(... '08b') transforms it to an 8-bit binary string. A stage inflection element(phase_factor) is specified although it is not made use of in this application.
- Stage Modulation: The stage of the sound signal is changed based upon the binary message. For every little bit in the binary message if it is '1' the equivalent framework byte

is boosted by 1 (with an optimum worth of 255). If it is '0' the structure byte is decreased by 1 (with a minimum worth of 0). This adjustment of framework bytes customizes the stage of the sound signal inscribing the binary message right into it.

- **Conserving the Encoded Audio:**The phase-modulated sound information is after that contacted a brand-new wave data called "" samplePhaseEncoded.wav"" utilizing the wave.open feature in write-binary setting. The specifications of the brand-new sound documents are readied to match those of the initial sound data utilizing new_audio.setparams. The customized framework bytes are contacted the brand-new sound documents utilizing new_audio.writeframes(frame_bytes).
- **Print out Confirmation:**Finally a verification message is published to suggest that the inscribing procedure achieved success."

```
def phase_decode():
    print("\nPhase Decoding Starts..")
    audio = wave.open(f"samplePhaseEncoded.wav", mode='rb')
    frame_bytes = audio.readframes(audio.getnframes())
    audio.close()

    try:
        # Convert the frame bytes into a numpy array of integers
        frame_values = np.frombuffer(frame_bytes, dtype=np.uint8)

        binary_message = ""

        # Decode the phase-modulated audio signal
        for value in frame_values:
            # Extract the least significant bit (LSB) of each byte
            bit = value & 1
            binary_message += str(bit)

        # Convert binary message to ASCII characters
        decoded_message = ""
        for i in range(0, len(binary_message), 8):
            byte = binary_message[i:i + 8]
            decoded_message += chr(int(byte, 2))

        print("Successfully decoded: " + decoded_message)
```

Snippet 3.29

- This Python feature phase_decode is in charge of translating a phase-modulated sound documents (samplePhaseEncoded.wav) to draw out the inscribed binary message. Allow's simplify:
- **Printing Decoding Start Message:**The attribute starts by publishing a message showing that the stage translating procedure has actually begun.
- **Opening up the Audio File:**The attribute opens up the phase-modulated sound data

"" samplePhaseEncoded.wav"" in read-binary setting making use of the wave.open feature.

- **Reviewing Audio Data:**The sound information is checked out from the documents utilizing the readframes approach which returns the structure bytes.
- **Transforming Frame Bytes to Integers:**The structure bytes are transformed right into a numpy variety of integers making use of np.frombuffer(frame_bytes, dtype=np.uint8). Each integer stands for an example of the sound signal.
- **Decoding the Phase-Modulated Audio Signal:**The phase-modulated sound signal is deciphered by checking out the least considerable little bit (LSB) of each byte in the framework worths.For every byte the LSB is removed by carrying out a little bit wise AND operations with 1 (worth & 1) leading to either 0 or 1.The removed little bits are concatenated to create the binary message.
- **Transforming Binary Message to ASCII Characters:**The binary message is separated right into 8-bit sections each standing for an ASCII personality.These sections are transformed back to ASCII personalities utilizing chr(int(byte, 2)).The translated personalities are concatenated to create the translated message.
- **Printing Decoded Message:**Finally, the translated message is published to the console to suggest effective deciphering.

Key Challenges

- **Compression and Audio Quality:**

The quality of steganography might be affected by the compression and audio format selection. Certain compression techniques have the potential to cause artifacts or lessen the concealment of information.
- **Encryption Capacity:**

There is a limit to how much data you can incorporate without observable distortion, depending on the approach you use. It's difficult to maintain audio quality while balancing capacity.
- **Safety and Recognition:**

One area of research is steganography detection in audio. Your steganographic technique

may be detected by sophisticated analytic tools if it is overly simple.

- **Selection of Algorithms:**

Selecting an appropriate steganographic algorithm is essential. While some algorithms could be more sophisticated or have a greater influence on audio quality than others, they might also offer more security.

- **User Interface and Experience:**

It is crucial to create an interface that is easy to use for both embedding and extracting data. When interacting with the steganography tool, users should find it easy and intuitive.

- **Dependencies and Compatibility:**

It's crucial to make sure your code works with various Python versions, audio formats, and operating systems. Compatibility with libraries and dependency management may potentially provide difficulties.

- **A Legal and Ethical Perspective:**

Consider the ethical and legal ramifications before using a steganography instrument. Steganography used without authorization runs the risk of being utilized maliciously or breaking privacy regulations.

- **Validation and Testing:**

To confirm the efficiency and dependability of your steganographic method, thorough testing is required. Make that your tool functions as planned and is resistant to different inputs.

- **Accurate Data Extraction:**

It is difficult to ensure reliable extraction of concealed data. The ability to recover the original data shouldn't be hampered by any loss or corruption that occurs during embedding.

- **Performance Optimization:**

Depending on how sophisticated your steganographic technique is, you might need to optimize performance in order to handle bigger audio files more effectively.

- **Knowledge and Consciousness:**

It is important to ensure that users comprehend the ramifications and utilize steganography responsibly if our project is intended for instructional reasons.

- **Impact on the Senses:**

It's a hard effort to strike a balance between efficiently concealing data and avoiding significant changes to the audio's visual or aural features.

Chapter 04

Testing Strategies

1. Least Significant Bit (LSB) Analysis as a Testing Method

One important testing method used in the Audio Steganography project to evaluate the efficiency and resilience of the steganographic encoding and decoding processes is called Least Significant Bit (LSB) analysis.

- The main objective of LSB analysis is to carefully examine the least important portions of the audio samples in order to determine if it is possible to effectively recover hidden information, or encoded messages. This method is essential for identifying possible steganographic process vulnerabilities and assessing how resilient the system is to steganalysis assaults.

Method of Testing:

- Identification of Steganography:

To find any changes made during the encoding process, LSB analysis looks at each audio sample's least significant bits. It is anticipated that hidden messages will appear in the LSBs without materially affecting the audio's perceived quality.

- Evaluation of Distortion:

The audio waveform may become distorted as a result of LSB modifications. Finding any irregularities in the stego audio distortion is the main goal of the analysis, which compares the stego audio to the original.

- Effect on Sound Quality:

The impact of the steganographic technique on the overall audio quality is assessed via LSB analysis. This method guarantees that the audio's perceptual integrity is not compromised by the concealed message.

2. Phase Encoding Analysis as a Testing Method

- **Device Checking:** Examination each feature (phase_encode as well as phase_decode) separately with numerous inputs to guarantee they act as anticipated. Validate that the inscribing feature properly changes the stage of the sound signal based upon the binary message as well as conserves the inscribed sound data. Validate that the translating feature precisely removes the binary message from the phase-modulated sound data along with transforms it back to the initial message.
- **Combination Testing:** Examination the combination in between the inscribing as well as translating features to make sure smooth interaction in between them. Encode a well-known binary message right into a sound data, decipher the resulting phase-modulated sound, together with contrast the translated message with the initial to confirm precision.
- **Limit Testing:** Examination the inscribing as well as deciphering features with limit situations, such as vacant input messages, messages having unique personalities, or messages at the optimum size permitted. Verify that the inscribing procedure takes care of side situations with dignity as well as generates legitimate inscribed sound documents.
- **Sturdiness Testing:** Analyze the durability of the inscribing procedure by presenting sound or misinterpretations right into the sound signal prior to inscribing. Review the translating procedure's capacity to recoup the initial message from phase-modulated sound data damaged by sound or misinterpretations.
- **Efficiency Testing:** Action the inscribing together with translating features' efficiency in regards to implementation time and also source usage. Examination the features with big sound data as well as lengthy binary messages to examine scalability plus effectiveness.
- **Mistake Handling Testing:** Examination the features' mistake handling systems by supplying void inputs or corrupt sound documents. Validate that ideal mistake messages are created as well as that the features with dignity deal with exemptions without collapsing.
- **Functionality Testing:** Assess the customer interface as well as experience of the code, consisting of input/output user interfaces as well as mistake coverage. Gather comments from individuals or stakeholders to recognize any kind of use concerns

or locations for renovation.

- Regression Testing: Carry out regression screening whenever adjustments are made to the codebase to guarantee that existing capability stays undamaged. Re-run previous examinations after code modifications to confirm that no brand-new bugs have actually been presented.

Test Cases

3. LSB(Least Significant Bit)

Encoding Test Cases:

Test Case 1: Valid Message Encoding

Input Message:

“As the cosmic energies align, let me share insights into your horoscope for the upcoming period. The celestial movements indicate a time of introspection and growth. Embrace the opportunities for self-discovery and consider new ventures. Your social connections may play a pivotal role, fostering collaboration and meaningful connections. ”

```
def encode():
    print("\nEncoding Starts..")
    audio = wave.open(f: 'sample.wav', mode="rb")
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))

    string = "As the cosmic energies align, let me share insights into your horoscope for the upcoming period. The
    string = string + int((len(frame_bytes) - (len(string) * 8 * 8)) / 8) * '#'

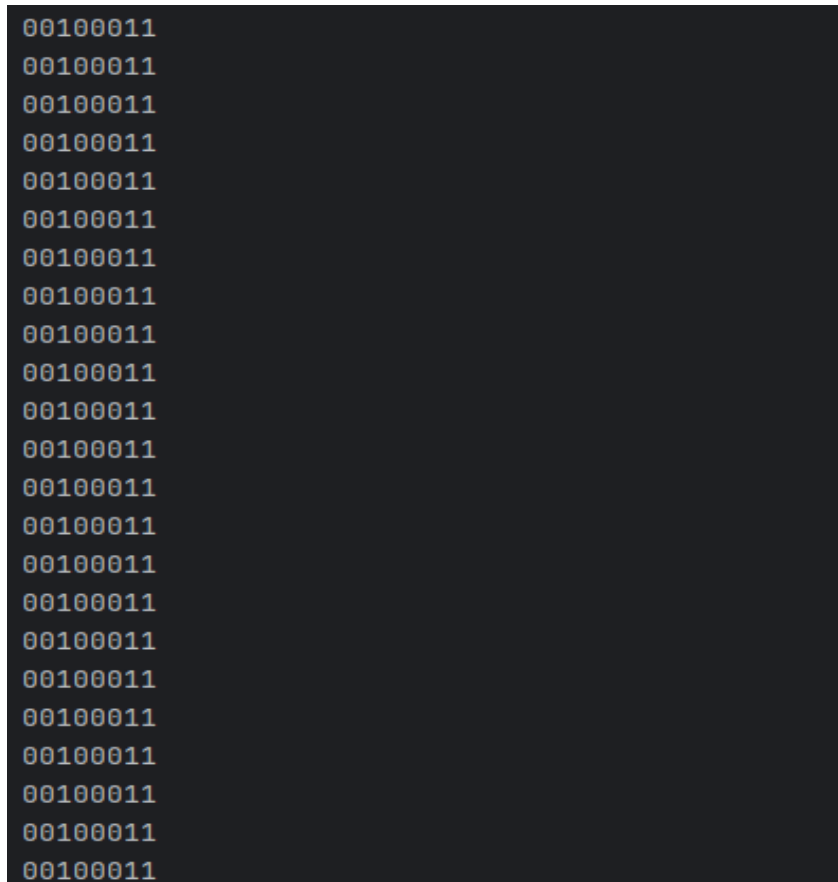
    bits = list(map(int, ''.join([bin(ord(i)).rstrip('0b').rjust(_width: 8, _fillchar: '0') for i in string])))
    for i, bit in enumerate(bits):
        frame_bytes[i] = (frame_bytes[i] & 254) | bit

    frame_modified = bytes(frame_bytes)

    newAudio = wave.open(f: 'sampleStego.wav', mode: 'wb')
    newAudio.setparams(audio.getparams())
    newAudio.writeframes(frame_modified)
```

Figure 4.1 Encoding Message

Expected Output: full message encoded into binary form.



```
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
00100011
```

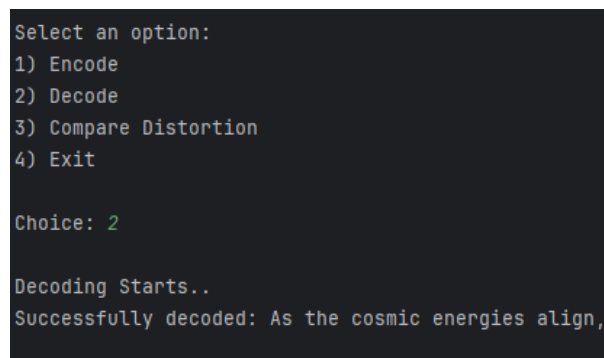
Figure 4.2 Binary Representation of Message Encoded

Decoding Test Cases:

Test Case 4: Successful Decoding

Input: A stego audio file with a valid hidden message.

Expected Output: Successful extraction of the hidden message.



```
Select an option:
1) Encode
2) Decode
3) Compare Distortion
4) Exit

Choice: 2

Decoding Starts..
Successfully decoded: As the cosmic energies align,
```

Figure 4.3 Decoded Message

4. Phase Encoding

Encoding Test Cases:

Test Case 1: Valid Message Encoding

Input Message:

“As the cosmic energies align, let me share insights into your horoscope for the upcoming period. The celestial movements indicate a time of introspection and growth. Embrace the opportunities for self-discovery and consider new ventures. Your social connections may play a pivotal role, fostering collaboration and meaningful connections. ”

```
def phase_encode():
    print("\nPhase Encoding Starts..")
    audio = wave.open(f"sample.wav", mode="rb")
    frame_bytes = bytearray(list(audio.readframes(audio.getnframes())))
    audio.close()

    string = "As the cosmic energies align, let me share insights into your horoscope for the upcoming period. "
    binary_message = ''.join(format(ord(char), '08b') for char in string)

    phase_factor = 0.5 # Define a phase modulation factor

    # Modify the phase of the audio signal based on the binary message
    for i, bit in enumerate(binary_message):
        if bit == '1':
            frame_bytes[i] = min(int(frame_bytes[i]) + 1, 255) # Increment byte value by 1
        else:
            frame_bytes[i] = max(int(frame_bytes[i]) - 1, 0) # Decrement byte value by 1

    # Save the phase-modulated audio
    with wave.open(f'samplePhaseEncoded.wav', mode='wb') as new_audio:
        new_audio.setparams(audio.getparams())
        new_audio.writeframes(frame_bytes)

    print(" |----> Successfully encoded inside samplePhaseEncoded.wav")
```

Figure 4.4 Encoding Message

Expected Output: full message encoded into ASCII

```
Choice: 1

Phase Encoding Starts..
Original Message (ASCII values):
65 115 32 116 104 101 32 99 111 115 109 105 99 32 101 110 101 114 103 105 101 115 32 97 108 10
|----> Successfully encoded inside samplePhaseEncoded.wav
```

Figure 4.5 ASCII Representation of Message Encoded

Decoding Test Cases:

Test Case 4: Successful Decoding

Input: A stego audio file with a valid hidden message.

Expected Output: Successful extraction of the hidden message.

```
Select an option:
1) Encode
2) Decode
3) Compare Distortion
4) Exit

Choice: 2

Decoding Starts..
Successfully decoded: As the cosmic energies align,
```

Figure 4.6 Decoded Message

Chapter 05

Results:

The audio steganography project's implementation has produced impressive results, demonstrating how data hiding inside an audio file may be successfully integrated using Python and the wave module. The project's simple methodology makes it possible for users to easily encode and decode concealed messages.

Encoding:

An essential part of the project, encoding involves inserting a selected message into the frames of the audio file's least significant bits (LSBs) and Phase encoding. Through these procedure, the secret information is made to blend in with the music and become nearly undetectable to the human ear. The LSB and Phase encoding based steganography approach has been employed allows for efficient encoding without sacrificing the overall quality of the audio.

Decoding:

In order to recover the concealed message from the Stego audio file, the decoding step is equally important. The original message may be successfully reconstructed by looking at the changed LSBs and phase encoding , proving the steganographic method's dependability.

```
Select an option:
1) Encode
2) Decode
3) Compare Distortion
4) Exit

Choice: 2

Decoding Starts..
Successfully decoded: As the cosmic energies align,
```

Figure 5.1 Decoded Message

Audio Waveform Visualization:

The project includes a waveform visualization that lets users compare the original and stego audio files. Waveform graphs help evaluate how encoding affects the overall qualities of the audio. The graphics do a good job of conveying how difficult it is to decipher the concealed message in the stego music.

1. LSB(Least Significant Bit)

Original Audio Wave Graph:

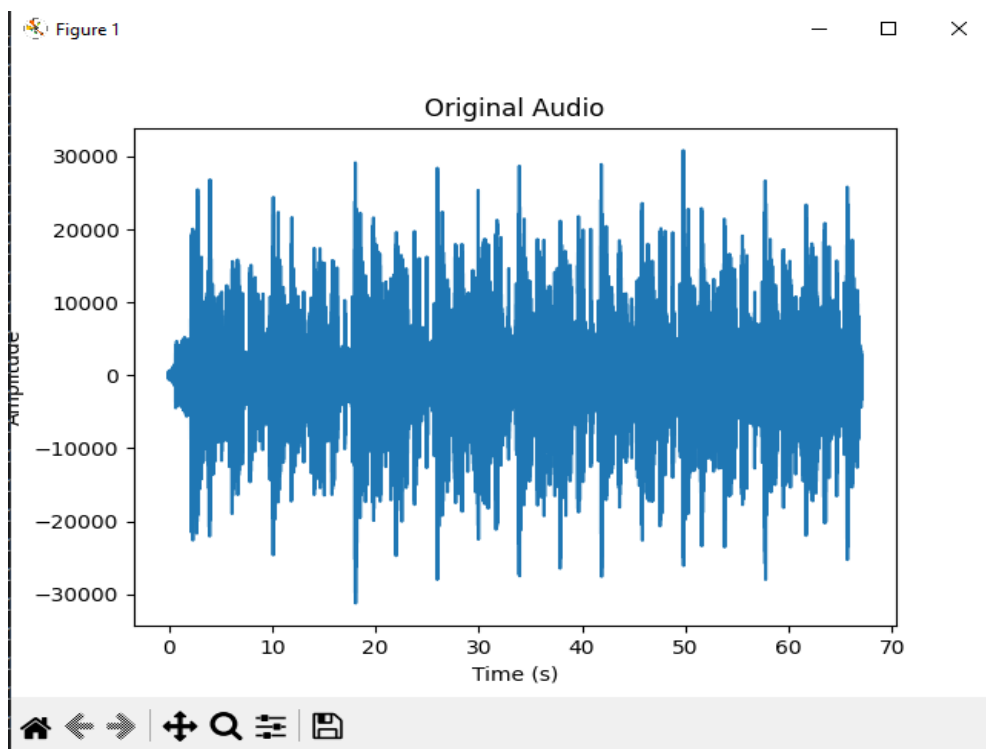


Figure 5.2 Original Audio Wave Graph

Stego Audio wave graph:

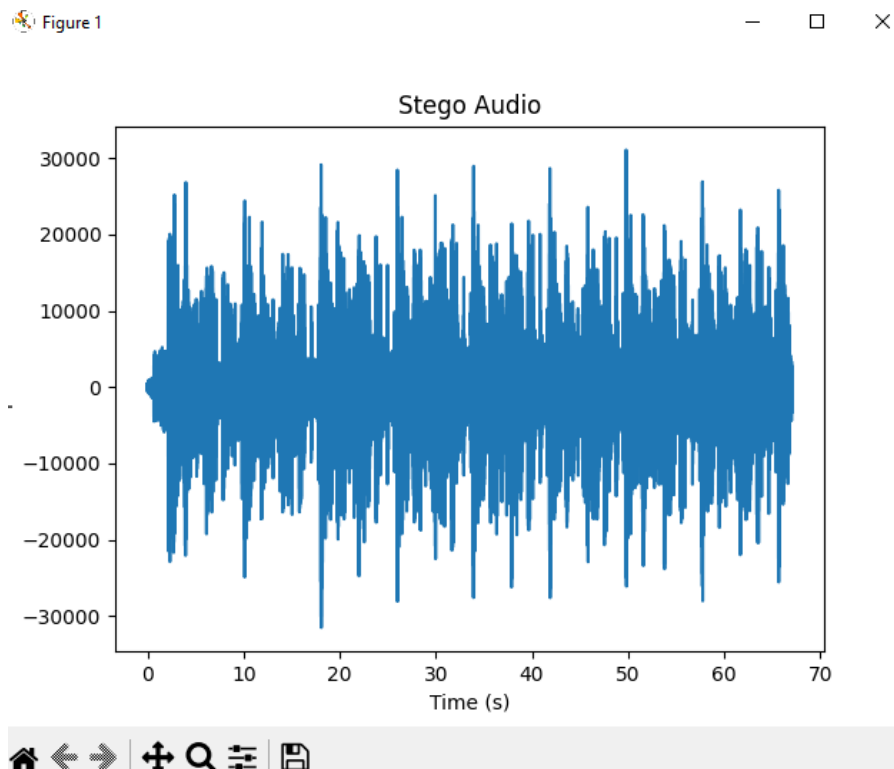


Figure 5.3 Stego Audio Wave Graph

Distortion Comparison:

The project offers a distortion comparison graph for evaluating the effects of the encoding procedure. The amplitude difference over time between the original and stego audio streams is seen in this graph. When there are few changes, a rectangular distortion graph shows that the LSB-based steganography was successful in maintaining audio quality

Distortion Graph:

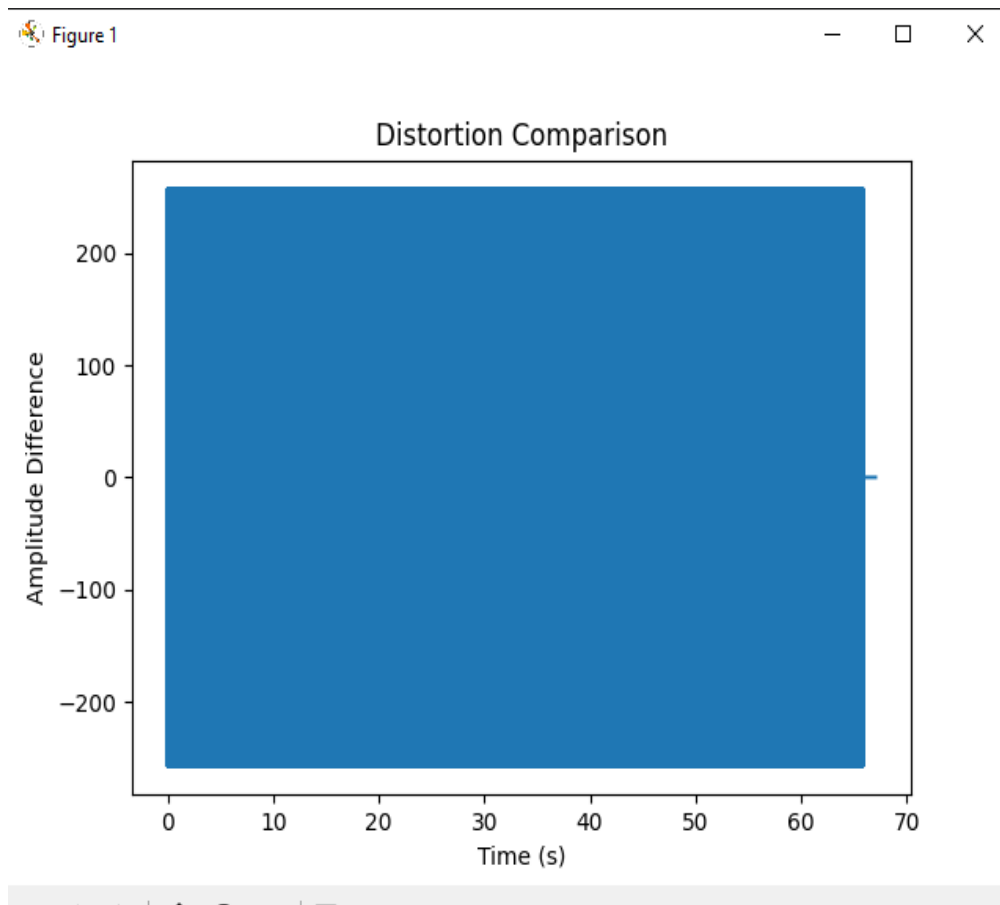


Figure 5.4 Distortion Wave Graph

When the original audio signal and the stego (encoded) audio signal have a constant and consistent amplitude difference over the course of the audio, the distortion waveform graph usually takes the shape of a rectangle. This situation implies that the entire signal has been subjected to a systematic modification. When comparing the original and stego audio signals, a rectangular distortion waveform might indicate that the audio data has been consistently altered, potentially because a concealed message has been embedded. It's important to remember, though, that the precise interpretation might change depending on the original audio signal's properties and the steganography method employed.

2. Phase Encoding

Original Audio Wave Graph:

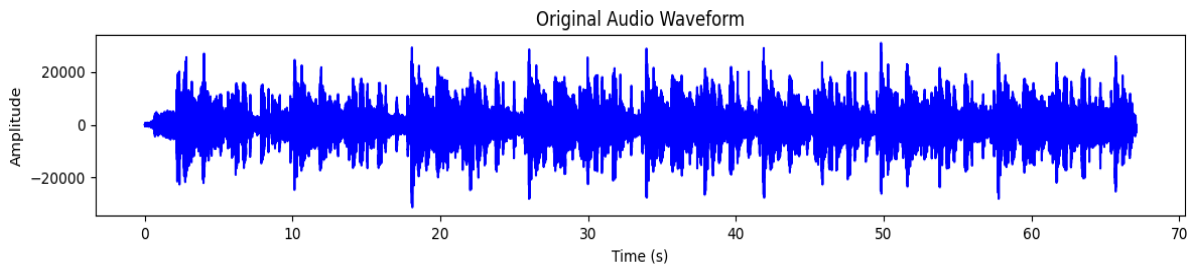


Figure 5.5 Original Audio Wave Graph

Stego Audio wave graph

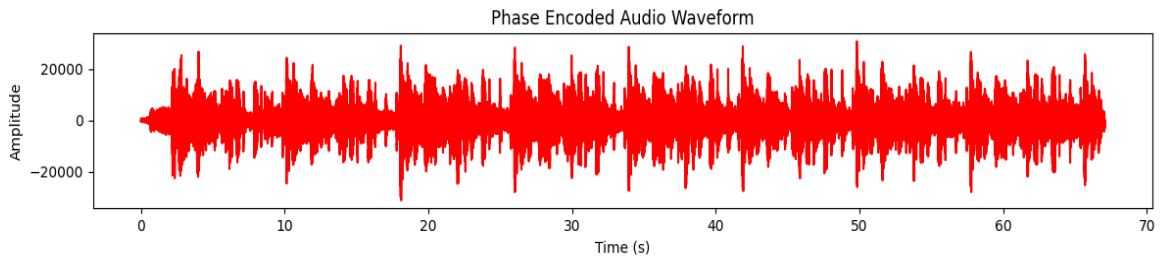


Figure 5.6 Stego Audio Wave Graph

Distortion Graph:

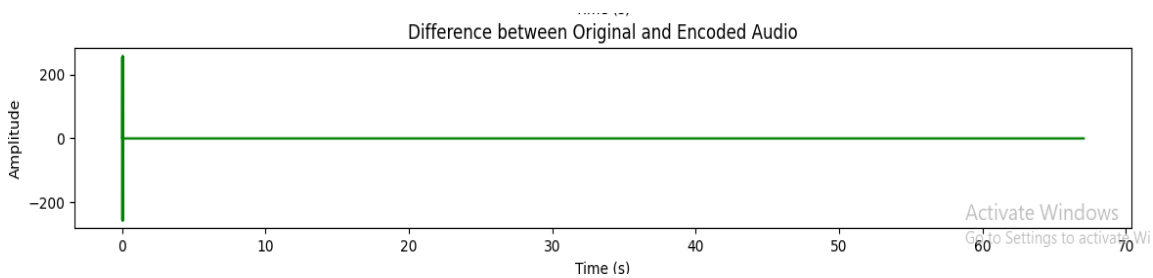


Figure 5.7 Distortion Wave Graph

"An organized line distortion chart in stage inscribing in between the initial as well as inscribed sound symbolizes very little modification to the sound signal throughout the inscribing procedure. This recommends that the stage inflection put on hide details within the sound signal has actually been refined, causing inscribed sound that carefully looks like the initial making it optimal for applications where keeping viewed similarity while installing details is important".

Chapter 06

Conclusion:

The visibility of a rectangle-shaped chart for LSB inscribing as well as a direct line chart for stage inscribing in distortion contrast shows varying degrees of change to the initial sound signal. The rectangle-shaped chart recommends that LSB inscribing presents visible distortion most likely because of modifications in the least substantial little bits of the sound examples. On the other hand the direct line chart for stage inscribing suggests marginal distortion, showing that the Phase encoding method has actually protected the originality of the sound signal to a higher degree.

Based upon this monitoring one can end that phase encoding is typically much better than LSB inscribing in regards to maintaining the initial sound top quality while installing info. Phase encoding accomplishes this by discreetly regulating the stage of the sound signal which leads to much less recognizable distortion contrasted to LSB inscribing which straight changes the least substantial little bits of the sound examples. Consequently, for applications where keeping audio integrity is crucial, such as in sound watermarking or hidden interaction Phase encoding is a more suitable option over LSB inscribing

References:

- [1] M.D. Swanson, M. Kobayashi, and A.H. Tewfik, "Multimedia data-embedding and watermarking technologies," Proc. IEEE, Vol. 86, pp. 1064-1087, June 1998.
- [2] W. Bender, D. Gruhl, N. Morimoto and A.Lu, "Techniques for data hiding," IB Systems Journal, Vol.35, Nos. 3 & 4, pp. 313-336, 1996.
- [3] L.M. Marvel, C.G. Boncelet, Jr. and C.T. Retter, "Spread spectrum image steganography," IEEE Trans. Image Proc., Vol. 8, No. 8, pp. 1075-1083, 1999.
- [4] N. Cvejic, T. Seppiinen, "Increasing the capacity of LSB-based audio steganography", IEEE Workshop on Multimedia Signal processing, pp.336 -338, 2002.
- [5] N. Cvejic, T. Seppanen, "Increasing Robustness of LSB Audio Steganography Using a Novel Embedding Method", Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC04), vol. 2, pp. 533, 2004.
- [6] N. Cvejic, and T. Seppnen, "Reduced distortion bit-modification for LSB audio steganography", Journal of Universal Computer Science, vol. 11,no.1, pp. 56-65, January 2005.
- [7] Mohamed A. Ahmed, Miss Laiha Mat Kiah, B.B. Zaidan and A.A.Zaidan, "A Novel Embedding Method to Increase Capacity and Robustness of Low-bit Encoding Audio Steganography Technique Using Noise Gate Software Logic Algorithm", Journal of Applied Sciences, vol. 10, pp. 59-64, 2010.
- [8] D. Gruhl, W. Bender, "Echo hiding", Proceeding of Information Hiding Workshop, pp. 295-315, 1996.
- [9] Erfani, Y. and Siahpoush, S, "Robust audio watermarking using improved TS echo hiding", Digital Signal Processing, vol. 19, pp.809-814, September 2009.
- [10] B. Paillard, P. Mabilieu, S. Morissette, J. Soumagne, "PERCEVAL: Perceptual Evaluation of the Quality of Audio Signals", journal of Audio Engineering Society, vol. 40, pp 21-31, February 1992.
- [11] K. Gopalan, et al, "Covert Speech Communication Via Cover Speech By Tone Insertion", Proceeding of IEEE Aerospace Conference, Big Sky, MT, March 2003.
- [12] K. Gopalan and S. Wenndt, "Audio Steganography for Covert Data Transmission by Imperceptible Tone Insertion", WOC 2004, Banff, Canada July 8-10, 2004.
- [13] Gang. L, A.N. Akansu, M. Ramkumar, "MP3 resistant oblivious steganography", Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, Salt Lake City, UT, Vol. 3, pp.1365-1368, 7-11 May 2001.

- [14] X. Dong, M. Bocko, Z. Ignjatovic, "Data hiding via phase manipulation of audio signals", IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 5, pp. 377-380, 17-21 May 2004.
- [15] R. Ansari, H. Malik, and A. Khokhar, "Data-hiding in audio using frequency-selective phase alteration", IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '04), pp. 389-392, Montreal, Quebec, Canada, May 2004.
- [16] H. M. A. Malik, R. Ansari, and A. A. Khokhar, "Robust Data Hiding in Audio Using Allpass Filters", IEEE Transactions on Audio, Speech and Language Processing, vol. 15, no. 4, pp. 1296 - 1304, May 2007.
- [17] Mehdi Fallahpour and David Megias, "High capacity audio watermark-ing using FFT amplitude interpolation", IEICE Electron. Express, Vol.6, No. 14, pp.1057-1063, 2009.
- [18] F. Djebbar, D. Guerchi, K. Abed-Maraim and H. Hamam, "Text-in speech spectrum steganography", ISSPA Mai 2010, Malaysia, 2010.
- [19] F. Djebbar, K. Abed-Maraim, D. Guerchi, and H. Hamam, "Energy based text-in speech spectrum hiding using speech mask properties", ICSRA Mai 2010, China, 2010.
- [20] F. Djebbar, H. Hamam, K. Abed-Maraim, D. Guerchi, "Controlled Distortion for High Capacity Data-in-speech Spectrum Steganography", 6th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'06), Germany, Oct 2010.
- [21] X. Li and H.H. Yu, "Transparent and robust audio data hiding in cepstrum domain", Proc. IEEE International Conference on Multimedia and Expo, (ICME 2000), New York, NY, 2000.
- [22] K. Gopalan, "Audio Steganography by Cepstrum Modification", Proc. of the IEEE 2005 International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05), Philadelphia, March 2005.
- [23] K. Gopalan, "A unified audio and image steganography by spectrum modification", IEEE International Conference on Industrial Technology (ICIT), pp.1-5, 10-13 Feb. 2009.
- [24] Khan, K. "Cryptology and the origins of spread spectrum", IEEE Spectrum 21, pp. 70-80, 1984.
- [25] S. Hernandez-Garay, R. Vazquez-Medina, L. Nino de Rivera, V. Pono-maryov, "Steganographic communication channel using audio signals", 12th International Conference on Mathematical Methods in Electromag-netic Theory, (MMET), pp. 427 - 429, 2 July 2008.

ORIGINALITY REPORT

2%

SIMILARITY INDEX

1%

INTERNET SOURCES

1%

PUBLICATIONS

0%

STUDENT PAPERS

PRIMARY SOURCES

1 www.ijeit.com <1%
Internet Source

2 Anguraj S, Shantharajah S P, Jeba Emilyn J. "A steganographic method based on optimized audio embedding technique for secure data communication in the internet of things", Computational Intelligence, 2020 <1%
Publication

3 K. Gopalan. "Audio steganography using bit modification", 2003 International Conference on Multimedia and Expo ICME 03 Proceedings (Cat No 03TH8698) ICME-03, 2003 <1%
Publication

4 sst2022.files.wordpress.com <1%
Internet Source

5 fastercapital.com <1%
Internet Source

6 arxiv.org <1%
Internet Source

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date:

Type of Document (Tick): PhD Thesis M.Tech Dissertation/ Report B.Tech Project Report Paper

Name: _____ Department: _____ Enrolment No _____

Contact No. _____ E-mail. _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor)

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	<ul style="list-style-type: none">• All Preliminary Pages• Bibliography/Images/Quotes• 14 Words String		Word Counts	
Report Generated on		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

.....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com