A

Major Project Report

on

CRYPTIC CANVAS GENERATOR USING IMAGE STEGANOGRAPHY

Submitted in partial fulfillment of the

Requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

By

Murarishetty Akshay Kumar

(20EG105236)

Bilapati Mahendar

(20EG105254)

Ponnagani Maheshwari

(20EG105725)

Under the guidance of

Mrs. Rama Mrudula

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Venkatapur(V), Ghatkesar(M), Medchal(D), Telangana-500088

2023-24

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report entitled **Cryptic Canvas Generator using Image Steganography** being submitted by Murarishetty Akshay Kumar bearing the Hall Ticket number **20EG105236**, Bilapati Mahendar bearing the Hall Ticket number **20EG105254**, Ponnagani Maheshwari bearing the Hall Ticket number **20EG105725** in partial fulfilment of the requirements for the award of the degree of the **Bachelor of Technology** in **Computer Science and Engineering** in **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision for academic year 2023 to 2024.

The results presented in this report have been verified and found to be satisfactory. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree.

Internal Guide

Mrs. Rama Mrudula

Assistant Professor, CSE

Dean, CSE

Dr. G. Vishnu Murthy

External Examiner

DECLARATION

We hereby declare that the Report entitled **Cryptic Canvas Generator using Image Steganography** submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** from **Anurag University** is a record of our original work done by us under the guidance of **Mrs. Rama Mrudula, Assistant Professor, Department of Computer Science and Engineering** and this project work has not been submitted to any University for the award of any other degree.

.

Murarishetty Akshay Kumar
20EG105236

Bilapati Mahendar
20EG105254

Ponnagani Maheshwari
20EG105725

Place: Hyderabad

Date :

ACKNOWLEDGMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Mrs. Rama Mrudula, Assistant Professor**, **Department of Computer Science and Engineering,** Anurag University for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, **Dean School of Engineering,** Anurag University, for their encouragement and timely support of our B.Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy, Dean, Department of Computer Science and Engineering,** Anurag University.

We also express our deep sense of gratitude to **Dr. V. V. S. S. S. Balaram,** Academic co-ordinator, **Dr. Pallam Ravi**, Project in-Charge, **Dr. V. Rama Krishna,** Project Co-ordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

Murarishetty Akshay Kumar
20EG105236

Bilapati Mahendar
20EG105254

Ponnagani Maheshwari
20EG105725

ABSTRACT

Steganography is the process of hiding one file inside another such that others can neither identify the meaning of the embedded object nor even the embedded object nor even recognize its existence. Current trends favor using digital image files as the cover file to hide other digital files that contain the secret message or information. Steganography becomes more important as more people join the cyberspace revolution. Steganography is the art of concealing information in ways that prevent the detection of hidden messages. The goal of steganography is to avoid drawing suspicion to the existence of a hidden message. This approach of information hiding technique has recently become important in a number of application areas. Digital audio, video, and pictures are increasingly furnished with distinguishing but imperceptible marks, which may contain a hidden copyright notice or serial number or even help to prevent unauthorized copying directly. Military communications systems make increasing use of traffic security techniques that, rather than merely concealing the content of a message using encryption, seek to conceal its sender, its receiver, or its very existence. Similar techniques are used in some mobile phone systems and schemes proposed for digital elections. One of the most common methods of implementation is Least Significant Bit Insertion, in which the least significant bit of every byte is altered by the bit-string representing the embedded file.

.

TABLE OF CONTENT

CHAPTERS **Page No.**

LIST OF FIGURES

1 INTRODUCTION

The use of multimedia digital signals has grown in popularity over the last decade as a result of the proliferation of wireless Internet-based services, such as the introduction of fourth-generation mobile communication systems that allow users to transfer data at speeds of up to 1Gbps. Digital data can be easily copied, modified, and retransmitted in the network by any user thanks to the availability of low-cost editing tools. It is critical to developing tools that protect and authenticate digital information in order to effectively support the growth of multimedia communications. We present a novel embedding scheme based on the LSB technique in this contribution. It has no effect if the value of a pixel in an image is changed by a value of 'l'.

However, the easy access to low-cost editing tools has introduced challenges related to the security and integrity of digital data. Users can easily copy, modify, and retransmit digital data, making it essential to develop robust tools to protect and authenticate multimedia content. These tools are critical for ensuring secure and reliable multimedia communications.

One such method to address these challenges is image steganography, which involves hiding data within digital images. Specifically, the least significant bit (LSB) steganography technique embeds data within the least significant bits of an image's pixels. This method offers an efficient way to conceal data while causing minimal visual impact on the image.

1.1 OVERVIEW

The Project demonstrates an application for image steganography using the LSB technique. The code leverages the stegano library to hide and reveal messages within image files. Additionally, it incorporates a user-friendly graphical user interface (GUI) built with the tkinter library, allowing users to interact with the program and perform steganography operations.

The application enables users to open an image file, input a message to be hidden, and save the modified image with the hidden data. Users can also reveal hidden data from an image and display the extracted message. By utilizing the LSB technique, the application ensures that changes to the image are minimal and virtually imperceptible to the human eye.

Overall, this project presents a practical and accessible approach to secure multimedia data transmission. By implementing LSB steganography in a user-friendly application, the project provides a means to safeguard digital information, contributing to the secure and private exchange of sensitive data within multimedia communications.

1.2 PROBLEM STATEMENT

With the increasing use of multimedia digital signals and easy access to editing tools, there is a growing need for secure methods to protect and authenticate digital data. Current methods of transmitting data can be vulnerable to interception, modification, and unauthorized access. The challenge is to develop an efficient and user-friendly solution that allows for the secure embedding of sensitive information within digital images, while minimizing any visual impact on the image and ensuring the confidentiality of the hidden data. The goal is to provide a practical application that allows users to hide and reveal messages in images using steganography techniques.

1.3 OBJECTIVE

The objective of the project is to design and implement a user-friendly Python application that demonstrates image steganography using the least significant bit (LSB) technique. The application aims to allow users to securely hide and reveal sensitive messages within digital images, ensuring the confidentiality and integrity of the hidden data while maintaining the visual quality of the images. Through the use of a graphical user interface (GUI), the project seeks to make steganography accessible and easy to use for anyone interested in securely transmitting data within multimedia content.

Securely Embed Messages: Provide a method for securely embedding sensitive messages within digital images using the LSB technique, which alters only the least significant bits of an image's pixels to hide data.

Minimal Visual Impact: Ensure that the process of embedding data into the image results in minimal visual changes, making the hidden data virtually imperceptible to the human eye.

User-Friendly Interface: Create an intuitive graphical user interface (GUI) using the tkinter library that allows users to easily interact with the application, open image files, input messages, and control the hiding and revealing of data.

Data Confidentiality and Integrity: Maintain the confidentiality and integrity of the hidden data by preventing unauthorized access and modifications. The application should securely handle the process of embedding and extracting data.

Save and Retrieve Modified Images: Enable users to save the modified images containing hidden messages and retrieve the hidden data from images when needed.

Educational and Practical Tool: Provide an educational tool for learning about steganography techniques and how they can be applied to secure multimedia communications. Additionally, the application can serve as a practical tool for securely transmitting sensitive information within images.

By achieving these objectives, the project aims to demonstrate the potential of image steganography as a means of securing digital data and facilitating the safe transmission of sensitive information within multimedia content.

1.4 PROJECT SCOPE

The project focuses on developing a Python application that demonstrates the use of the least significant bit (LSB) steganography technique for hiding and revealing messages within digital images. The scope of the project includes:

1.4.1 Implementation of Steganography: The application uses the stegano library to apply the least significant bit (LSB) technique for steganography.In LSB steganography, data is embedded in the least significant bits of an image's pixels.The process involves adjusting the least significant bit of each pixel's RGB values to encode the message without significantly altering the image's visual quality.This approach ensures that the hidden data is subtle and hard to detect by casual observers.

1.4.2 Graphical User Interface (GUI): The application uses the tkinter library to create an intuitive and user-friendly GUI.Users can easily navigate the application with clearly labeled buttons and interactive widgets.The GUI includes buttons for opening

image files, hiding data in images, revealing hidden data, and saving modified images.Users can input text messages and view the hidden data within text widgets, making the application straightforward to use.

1.4.3 Image and Text Handling: The application supports popular image formats such as PNG and JPG, allowing users to work with a variety of images.Users can input text messages directly into the application, which are then embedded within the selected image using the LSB technique.The program provides a text area where users can see hidden messages that have been revealed from images.

1.4.4 Data Security and Confidentiality: The project emphasizes the secure embedding of messages within images using the LSB technique, which alters only the least significant bits of the image's pixels.This subtle alteration helps maintain the confidentiality of the hidden data, making it challenging for unauthorized users to detect the embedded message.By ensuring that the hidden data is stored securely, the application supports the integrity of the embedded message.

1.4.5 Educational and Practical Use: The project provides a practical tool for those interested in exploring and learning about steganography techniques.Users can gain hands-on experience with hiding and revealing messages in images, offering a practical understanding of how steganography can be used for data security.Additionally, the application can serve as a useful tool for securely transmitting sensitive information within images in various practical scenarios.

1.4.6 Save and Retrieve Modified Images: Users can save images with hidden data, allowing them to keep a record of the modified image for later use.This functionality supports the transmission of sensitive data via images, enabling users to share the modified images as needed.The application can also retrieve hidden messages from modified images, making it easy for users to access and view the data that was previously hidden.By providing these functionalities, the project effectively demonstrates how steganography can be applied in a practical and user-friendly manner, contributing to the secure transmission of sensitive data within multimedia content.

2 LITERATURE SURVEY

A literature survey for Steganography, particularly the least significant bit (LSB) technique, has a rich history and has been widely studied as a method of hiding data within digital media such as images, audio, and video files. The primary goal of steganography is to embed secret information within a carrier medium in a way that is not detectable to casual observers. In recent years, research in the field has focused on improving the capacity, security, and robustness of steganographic methods.

1. LSB Steganography: LSB steganography is one of the most popular techniques due to its simplicity and effectiveness. By modifying the least significant bits of each pixel in an image, data can be hidden without significantly altering the visual quality of the image. Researchers have explored various approaches to improve the security and efficiency of LSB steganography, such as randomizing the embedding pattern to enhance security.

LSB Steganography Mechanism: LSB steganography involves embedding data in the least significant bits of an image's pixels, audio samples, or video frames.Pros and Cons: This method is popular for its simplicity and minimal visual impact on images, but it can be vulnerable to detection due to predictable embedding patterns.Enhancements: Research focuses on improving the security of LSB methods by randomizing the embedding process, combining it with other techniques, or using machine learning to enhance data embedding and retrieval.

2. Image Quality and Detection: A major challenge in LSB steganography is balancing the amount of data hidden with the visual quality of the image. Various studies have examined how to minimize image distortion while maximizing data capacity. Additionally, researchers have developed techniques to detect steganography, such as statistical analysis of images, which highlights the need for more advanced hiding techniques.

Impact on Image Quality: While LSB steganography typically introduces minimal visual changes, excessive data embedding can cause noticeable distortions.

Detection Techniques: Various methods exist for detecting steganography, including statistical analysis, histogram examination, and machine learning-based approaches that identify anomalies in image data.

Quality Optimization: Researchers seek to optimize the balance between embedding capacity and image quality to maximize data hiding while minimizing visual impact.

3. Applications in Security:

Steganography has practical applications in digital security, including secure communication and data storage. It can be used to protect sensitive information from unauthorized access and interception. However, concerns remain about the potential misuse of steganography for illegal activities, leading to ongoing research into detection and countermeasures.

Secure Communication: Steganography is used to protect messages from eavesdropping, providing a secure channel for confidential communications.

Data Storage and Transfer: Sensitive information can be safely stored and transferred within steganographic media, safeguarding it from interception.

Concerns: Despite legitimate uses, steganography can be misused for illicit purposes, leading to ethical and legal concerns about its application.

4. Advancements and Hybrid Approaches: Researchers have proposed hybrid methods that combine LSB steganography with other techniques (e.g., cryptography) to improve security and data protection. These approaches aim to increase the robustness of steganographic methods against attacks and detection.

Hybrid Methods: Combining steganography with cryptography or other data protection methods can improve the overall security of hidden data.

Machine Learning and AI: Research explores using AI and machine learning to enhance steganographic techniques, improve data hiding, and strengthen resistance against detection.

Watermarking: Sometimes combined with steganography, digital watermarking adds an additional layer of protection to digital media.

5. Practical Challenges:

Implementing steganography in practical applications poses challenges such as maintaining data integrity and ensuring compatibility with different file formats. Researchers are actively working on overcoming these challenges to create more reliable and efficient steganographic systems.

Data Integrity: Ensuring the integrity of hidden data during embedding, transmission, and extraction is crucial for practical applications.

File Format Compatibility: Different file formats have varying capacities for data embedding, and maintaining compatibility is essential for widespread use.

Robustness Against Attacks: Developing methods that withstand various types of attacks (e.g., compression, noise, or cropping) is vital for the success of steganography.

Legal and Ethical Considerations: Researchers must navigate the ethical implications of steganography, particularly in terms of privacy and security concerns.

Overall, the literature on steganography provides valuable insights into the development and implementation of secure data-hiding techniques. While LSB steganography remains a popular and accessible method, there is ongoing research into more sophisticated and secure approaches to improve the reliability and effectiveness of steganography in digital communications.

In summary, the literature highlights the ongoing evolution and refinement of steganography techniques, particularly LSB-based methods, to enhance data protection while minimizing visual impact. Researchers continue to explore and address challenges related to detection, robustness, and ethical use of steganography in practical applications.

3 ANALYSIS

3.1 EXISTING SYSTEM

In the context of the project focusing on image steganography using the least significant bit (LSB) technique, the existing system encompasses the established methods and approaches that are currently used for hiding and revealing data within digital images. Here's an in-depth examination of the various components and methodologies of the existing system:

1. Data Embedding:

LSB Technique: The least significant bit (LSB) technique is the most common approach for embedding data within digital images. It involves modifying the least significant bits of each pixel's RGB values in an image to hide a secret message.

Embedding Capacity: The amount of data that can be embedded within an image depends on its size and color depth. Typically, each pixel can hold several bits of data without visibly affecting the image.

Impact on Image Quality: When done properly, LSB embedding results in minimal visual changes to the image, making the hidden data nearly undetectable to the human eye.

2. Data Extraction:

Extraction Process: To retrieve hidden messages, the system reads the least significant bits of each pixel in the image. The extracted bits are combined to reconstruct the hidden message.

Accuracy and Reliability: Accurate data extraction is crucial to ensure the integrity of the hidden message. The process needs to account for potential image manipulations such as resizing or format conversion.

3. Graphical User Interface (GUI):

Usability and Accessibility: Existing systems often use a graphical user interface (GUI) built with libraries such as tkinter to make the application user-friendly and easy to navigate.

Components: The GUI typically includes buttons and text areas for opening images, inputting messages, hiding data, revealing data, and saving modified images.

Feedback and Interaction: The GUI provides immediate feedback to users regarding the operations being performed and the state of the application.

4. File Handling:

Image Formats: The system supports opening and saving images in common formats such as PNG and JPG.

Saving Modified Images: Once data is hidden within an image, the system allows the user to save the modified image as a new file.

Compatibility and Performance: The system's ability to handle various file formats and large image sizes is essential for practical applications.

5. Data Security:

Basic Security: The existing system offers a basic level of data security by concealing messages within images, preventing casual interception and unauthorized access.

Lack of Encryption: While data is hidden, the existing system may not provide advanced encryption methods to secure the message further.

Vulnerability to Detection: Predictable patterns in LSB embedding may make the system vulnerable to steganalysis (the detection of hidden data) using statistical and image processing techniques.

6. Limitations:

Embedding Capacity and Robustness: The existing system may have limited capacity for data embedding and may not be robust against image manipulations such as compression, cropping, or resizing.

Detection Risks: Steganography techniques, including LSB, can be detected using various methods such as histogram analysis, frequency domain analysis, and machine learning-based approaches.

Potential for Improvements: Current research focuses on enhancing the robustness and security of steganography methods to mitigate detection risks and increase the capacity and effectiveness of data hiding.

In summary, the existing system demonstrates the basic principles of LSB steganography in digital images and provides a foundation for further development and improvement in the field. Researchers are working on addressing the limitations of the existing system by exploring new methods and hybrid approaches that enhance data security, embedding capacity, and resistance to detection.

3.2 PROPOSED SYSTEM AND ADVANTAGES

A proposed system for image steganography using the least significant bit (LSB) technique aims to address the drawbacks of existing systems by introducing enhancements in security, robustness, and resistance to detection. Here's an in-depth examination of the components and features of a proposed system:

1. Advanced LSB Embedding:

Randomized Embedding Patterns: Instead of using a predictable pattern, the proposed system adopts a randomized approach for embedding data. This technique ensures that the data is not embedded in a linear or predictable manner, reducing the chance of detection by steganalysis tools.

Adaptive LSB Embedding: The system dynamically adapts the LSB technique based on the image's characteristics, such as texture and color variations. This approach allows for more efficient hiding of data with minimal impact on image quality.

2. Data Security and Encryption:

Encryption of Hidden Data: Before embedding the data in the image, the system encrypts the data using strong encryption algorithms such as AES. This ensures that even if the data is extracted, it cannot be understood without the correct decryption key.

Secure Key Management: The system includes robust key management protocols for securely handling encryption and decryption keys. This feature ensures that only authorized parties can access the hidden data.

3. Robustness and Error Handling:

Resilience to Image Manipulations: The system incorporates error detection and correction codes to maintain the integrity of hidden data even if the image undergoes manipulations such as compression, cropping, or resizing.

File Format Compatibility: The proposed system is designed to work seamlessly with various image file formats, such as PNG, JPG, and BMP, ensuring compatibility and data integrity during format conversions.

4. Detection Resistance and Countermeasures:

Noise and Filtering Techniques: To resist detection, the system may use noise addition, pixel shuffling, or filtering techniques to obscure the presence of hidden data.

Dynamic Embedding Parameters: The system dynamically adjusts embedding parameters, such as bit depth and patterns, according to the image's characteristics and the amount of data being hidden.

5. User-Friendly Interface and Automation:

Intuitive GUI: The proposed system features a user-friendly graphical user interface (GUI) with clear options for opening, saving, and processing images, as well as inputting and extracting data.

Automation and Customization: The system offers options for batch processing multiple images and customizing embedding and extraction parameters, enhancing efficiency and flexibility.

6. Legal and Ethical Compliance:

Compliance Checks: The system includes mechanisms to ensure compliance with legal and ethical guidelines for data hiding, such as restricting use to authorized and legitimate purposes.

Education and Awareness: Users are educated on the ethical implications and legal constraints of using steganography to promote responsible use and avoid misuse.

7. Performance and Efficiency:

Optimized Algorithms: The proposed system uses optimized algorithms to improve the efficiency of data hiding and extraction, minimizing the time required to process images.

Resource Management: Efficient use of memory and processing power ensures smooth operation even when handling large image files or significant amounts of data.

By implementing these advanced features, the proposed system provides a more secure, robust, and user-friendly steganography solution. It effectively addresses the limitations of existing systems and offers enhanced protection for hidden data against detection and unauthorized access.

Advantages:

A well-designed image steganography system using the least significant bit (LSB) technique can provide several in-depth advantages. Here's an exploration of these advantages, accompanied by examples to illustrate how the technique works and its benefits:

1. Minimal Visual Impact:

Preservation of Image Quality: LSB steganography alters only the least significant bits of the image's pixel values, which causes minimal changes to the overall image quality. For example, in a 24-bit color image, changing the least significant bit of each color channel (red, green, and blue) for each pixel may not result in noticeable visual differences.

Stealthy Data Hiding: Because the visual impact is minimal, the hidden data remains concealed from the human eye. For instance, a photograph of a sunset can have secret text embedded using LSB steganography without affecting the beauty of the image.

2. Simplicity and Accessibility:

Ease of Implementation: The LSB technique is easy to understand and implement, even for beginners. For example, an application can use simple code to modify the least significant bits of each pixel in an image to embed data.

Compatibility with Common Formats: LSB steganography can be applied to popular image formats such as PNG and JPG. For instance, a user can embed data in a standard PNG image and send it to another party, who can then extract the data from the same image format.

3. High Embedding Capacity:

Significant Data Volume: The capacity to hide data is directly related to the size and color depth of the image. For example, a high-resolution photograph (such as a 4K image) can hold a substantial amount of data without affecting the image's visual quality.

Multiple Bits per Pixel: LSB steganography can be adjusted to hide multiple bits of data per pixel, increasing the overall data capacity. For instance, in a 24-bit color image, up to three bits (one per color channel) can be hidden in each pixel.

4. Enhanced Security and Privacy:

Secure Communication: LSB steganography provides a method for concealing messages within images, allowing for covert communication. For example, sensitive business communications can be hidden within ordinary family photos and sent securely.

Encryption Integration: When combined with encryption, LSB steganography ensures that even if the hidden data is intercepted, it remains protected. For instance, a user can encrypt a message before embedding it in an image, adding an extra layer of security.

5. Robustness and Resilience:

Resistance to Detection: By using advanced embedding techniques such as randomized or adaptive LSB, the system can minimize the chances of detection by steganalysis tools. For example, hiding data in less predictable patterns within the image can prevent statistical analysis from easily identifying the presence of hidden data.

Robustness Against Manipulations: Implementing error detection and correction codes helps maintain the integrity of hidden data even if the image undergoes changes

such as resizing or compression. For instance, embedded data in a photograph can remain intact despite compression for email attachment.

6. User-Friendly Applications:

Intuitive Interfaces: Many modern steganography tools offer user-friendly graphical user interfaces (GUIs), making it easy for users to hide and reveal data within images. For example, a desktop application might have buttons for opening images, inputting secret messages, and saving the resulting image with hidden data.

Automation and Customization: Advanced applications may offer features such as batch processing of multiple images and customizable parameters for embedding and extraction, increasing efficiency and flexibility. For instance, a user might automate hiding a series of messages in a set of images for large-scale communication.

7. Legal and Ethical Compliance:

Guidance and Awareness: By providing information and guidance on ethical use, the system helps users comply with legal and ethical standards in data hiding. For instance, informing users of the appropriate use cases for steganography can prevent misuse for criminal purposes.

8. Performance and Efficiency:

Optimized Algorithms: Efficient data hiding and extraction algorithms minimize the time and computational resources required for processing images. For example, a system might use parallel processing or optimized data handling to quickly embed data in high-resolution images.

Low Overhead: LSB steganography typically requires minimal computational overhead, making it suitable for various devices and applications, including mobile devices and low-power systems.

These advantages demonstrate how a well-designed image steganography system using LSB can provide significant benefits in terms of security, privacy, ease of use, and efficiency, making it a practical and effective method for covert data hiding and secure communication.

Overall, the proposed system for RecoverEase using deep image search technology offers numerous advantages, including efficiency, accuracy, user-friendliness, security, and accessibility. By harnessing the power of advanced technology, RecoverEase aims to revolutionize the management of lost and found items, providing a seamless and effective solution for individuals seeking to reclaim their belongings.

3.3 SYSTEM REQUIREMENTS

3.3.1 Software requirements

An in-depth explanation of the software requirements for an image steganography project using the least significant bit (LSB) technique is provided below:

1. Programming Language:

Python: Python is a high-level programming language known for its readability, ease of use, and vast library support. This makes it an excellent choice for implementing image steganography projects. Python allows for rapid development and prototyping while offering extensive libraries for image manipulation, data encryption, and GUI creation.

Libraries: The project should utilize libraries such as Pillow for image manipulation and stegano for steganography. Pillow provides tools for opening, editing, and saving images, while stegano includes methods specifically for hiding and revealing data within images.

2. Graphical User Interface (GUI):

Framework: tkinter is a built-in Python library that allows for creating graphical user interfaces (GUIs). It provides widgets such as buttons, text fields, and labels, making it straightforward to design an intuitive interface for the project.

Components: The GUI should include buttons for opening and saving images, text fields for inputting messages, and options to hide or reveal data. These components should be arranged in a user-friendly manner, allowing for seamless interaction with the application.

3. Image Processing:

Image Formats: The software should support common image formats such as PNG, JPG, and BMP to ensure compatibility with a variety of images. Pillow can handle multiple formats, making it a suitable choice for this requirement.

Image Manipulation: Functions for loading, saving, and manipulating images are necessary, including methods to read and write pixel data. Image manipulation tools allow for the extraction and embedding of data within the least significant bits of the image's pixel values.

4. Steganography Techniques:

LSB Embedding and Extraction: The software should include functions for embedding data in the least significant bits of pixel values and extracting it. LSB steganography involves modifying the least significant bit of each pixel's color values to hide data.

Error Handling: The software should handle potential errors during embedding or extraction, such as image format incompatibilities, invalid data input, or corrupted files.

5. Security and Encryption:

Data Encryption: To enhance security, the software can incorporate encryption for hidden data using strong encryption algorithms such as AES. Libraries like cryptography can provide encryption and decryption functions.

Key Management: Proper handling of encryption keys is essential for maintaining data security. This includes secure storage and transmission of keys, as well as proper access control.

6. Performance and Efficiency:

Optimized Algorithms: The software should use optimized algorithms for efficient embedding and extraction of data. This can include parallel processing or optimization techniques to minimize processing times and resource usage.

Batch Processing: If applicable, the software may offer batch processing capabilities, allowing users to handle multiple images or data files at once. This feature can save time and improve efficiency for large-scale data hiding.

7. Compliance and Legal Considerations:

Ethical Use Guidelines: The software should provide information and guidelines on the ethical and legal use of steganography to promote responsible practices and prevent misuse.

Compliance Checks: The software should include checks to ensure that data hiding adheres to legal and ethical standards, such as prohibiting the use of steganography for malicious purposes.

8. Documentation and User Support:

User Manual: Clear documentation and a user manual should explain how to use the application, including instructions on hiding and revealing data within images.

Help and Support: The software may offer help and support options such as FAQs, tutorials, or customer support to assist users in understanding the system and resolving any issues.

9. Cross-Platform Compatibility:

Operating Systems: The software should work on multiple operating systems, such as Windows, macOS, and Linux, to increase accessibility and reach a broader user base. This requires testing and adjustments to ensure consistent performance across platforms.

10. Testing and Quality Assurance:

Testing: Thorough testing is essential to ensure the software functions correctly and efficiently across different scenarios. This includes unit tests, integration tests, and user acceptance tests.

Quality Assurance: Quality assurance measures such as code reviews, automated testing, and user feedback can help improve the overall reliability and usability of the software.

By addressing these requirements, the project can deliver a secure, efficient, and user-friendly image steganography system that supports hiding and revealing data within images effectively.

3.3.2 Hardware Requirements:

When considering the hardware requirements for the image steganography project implemented in the provided Python code, it is important to focus on the performance and capabilities of the underlying system. The requirements can vary based on the size of the images being processed and the complexity of the data being hidden or revealed. Below is an in-depth explanation of the hardware requirements:

1. Processor: A modern multi-core processor is recommended for efficient handling of image processing tasks and any associated computational operations such as embedding and extracting data. Although the code may work on older processors, a faster CPU will enhance the overall performance and speed up operations.

2. Memory (RAM): Sufficient RAM is important to ensure smooth operation of the application, especially when working with larger images. Adequate RAM (at least 4GB, with 8GB or more being preferable) helps prevent potential memory issues and allows the application to handle multiple images and data simultaneously.

3. Storage: The project requires storage space to store images, particularly during the process of opening and saving images with hidden data. The size of storage depends on the number and size of the images being processed. A minimum of several gigabytes of free storage space is recommended for temporary file handling and saving images.

4. Display: While not directly impacting the code's functionality, a high-resolution display can enhance the user experience by displaying images and text more clearly. A larger screen size may also improve usability.

5. Peripherals: Standard input devices such as a keyboard and mouse are necessary for user interaction with the application. Additional peripherals such as external storage devices may be useful for saving or loading images.

6. Graphics Processing Unit (GPU): Though the provided code does not utilize advanced image processing that would benefit from GPU acceleration, a dedicated GPU can improve overall system performance, especially when dealing with high-resolution images or more complex image processing tasks.

7. Operating System Compatibility: The hardware should be compatible with the chosen operating system (e.g., Windows, macOS, Linux), which can affect the

performance and functionality of the application. Modern hardware is typically compatible with recent versions of major operating systems.

8. Network Connectivity: While the code itself does not require network connectivity, it may be helpful for downloading necessary libraries, sharing images, or working with cloud storage.

9. Power Supply: For stable and consistent operation, a reliable power supply is essential, particularly for desktop computers that may run the application for long periods.

10. Cooling: Proper cooling, whether through air or liquid cooling systems, is necessary to prevent overheating during long or intensive processing sessions, especially when working with high-resolution images.

Overall, the hardware requirements for this project are generally modest, as the application primarily handles image processing and text data, which are not overly demanding tasks on modern hardware. Nonetheless, having higher hardware specifications can improve the speed and responsiveness of the application, especially when working with larger images or more complex data.

3.4 FUNCTIONAL REQUIREMENTS

Functional requirements of an image steganography project using the least significant bit (LSB) technique is as follows:

1. Image File Handling:

Load Image: The system must enable users to select and load image files from their local file system. It should support popular image formats like PNG, JPG, and BMP. Image loading involves reading the file's metadata and pixel data, and then displaying the image within the application.

Save Image: Users should be able to save modified images (with hidden data) back to their file system in various formats such as PNG or JPG. The system should handle saving images efficiently and allow the user to specify the file path and format.

2. Data Embedding:

Hide Data: The system must provide the functionality to embed text data within an image using the LSB technique. This involves modifying the least significant bits of pixel color values in the image to encode the text data.

Select Data: Users should be able to input text data directly into a text field in the application or import it from a file. The system should provide tools for editing and managing the data to be embedded.

3. Data Extraction:

Reveal Data: The system should allow users to extract hidden data from images that contain embedded data. This involves reading the least significant bits of the pixel data in the image and reconstructing the hidden text.

Display Data: The extracted text data should be displayed in a user-friendly manner, either in a text field within the application or saved to a file as chosen by the user.

4. User Interface:

Intuitive Layout: The user interface should be designed to be user-friendly and straightforward, with a logical layout of buttons, menus, and text fields. The application should provide guidance or tooltips for user actions.

Control Buttons: Buttons for tasks such as opening and saving images, hiding and revealing data, and optionally encrypting or decrypting data should be clearly labeled and easily accessible.

5. Security:

Data Encryption (Optional): The system may provide an option to encrypt the text data before embedding it within the image. This adds an additional layer of security by making the hidden data unreadable without a key.

Key Management: Proper management of encryption keys is essential for maintaining data security. This includes secure storage and access to keys, ensuring only authorized users have access to the keys.

6. Performance and Efficiency:

Processing Speed: The system should perform data embedding and extraction tasks efficiently, even with high-resolution images or large data files. This includes optimizing algorithms and leveraging parallel processing when possible.

Batch Processing (Optional): The ability to hide or reveal data in multiple images at once can save time and increase efficiency, particularly for users who need to process large volumes of images.

7. Error Handling:

Error Messages: Clear and informative error messages should be provided to the user in case of issues such as invalid file format, corrupted image data, or data extraction failures.

Input Validation: The system should validate user inputs such as image files and data to ensure they meet the expected criteria, reducing the likelihood of errors during processing.

8. Compatibility:

Cross-Platform Compatibility: The application should run smoothly on various operating systems, such as Windows, macOS, and Linux. This requires testing and ensuring the application works consistently across different platforms.

Support for Image Formats: The system must support multiple image formats such as PNG, JPG, and BMP, enabling users to work with a variety of images.

9. Documentation and Help:

User Manual: A comprehensive user manual should explain how to use the application, including step-by-step instructions for hiding and revealing data within images.

Online Help: Additional online resources such as FAQs, tutorials, and customer support should be available to assist users and answer common questions.

10. Compliance and Ethical Use:

Ethical and Legal Guidelines: The system should provide information on ethical and legal use to promote responsible use of steganography and avoid misuse for malicious purposes.

Compliance Checks: Compliance checks may be included to ensure the user follows ethical and legal guidelines, potentially preventing the use of steganography for harmful or illegal activities.

By meeting these functional requirements, the project can deliver an efficient and user-friendly steganography system that effectively hides and reveals data within images while maintaining ethical and legal standards.

3.5 NON- FUNCTIONAL REQUIREMENTS

In-depth explanations of the non-functional requirements for an image steganography project using the least significant bit (LSB) technique are provided below:

1. Performance:

Processing Time: The system should hide and reveal data within images in a timely manner, ensuring that the application is responsive even when handling large data sets or high-resolution images. Optimization techniques such as efficient algorithms and caching can reduce processing time.

Resource Usage: The application should efficiently manage system resources (CPU, memory, and storage) to avoid excessive usage that can slow down the system or other applications. For instance, image processing should be memory-efficient to prevent crashes on devices with limited RAM.

2. Usability:

Ease of Use: The user interface should be straightforward, with clear instructions and labels. Users should be able to perform tasks such as loading, saving, hiding, and revealing data without confusion. Tutorials and tooltips can provide additional guidance.

Accessibility: The application should follow accessibility standards to accommodate users with disabilities. This includes keyboard navigation, screen reader compatibility, and visual aids for users with visual impairments.

3. Scalability:

Handle Large Data Volumes: The system should be able to handle large images and data sets efficiently. This may involve optimization techniques such as batch processing and parallel processing to manage high workloads.

Scalable Processing: The application should adjust to increased processing demands, potentially using distributed computing or cloud services to handle heavy workloads without significant performance degradation.

4. Security:

Data Protection: The system should protect hidden data and user data from unauthorized access. This can involve encryption for data security during embedding and extraction, as well as secure storage of sensitive information.

Authentication: If the system supports encryption, it should verify user credentials or encryption keys before granting access to hidden data. This adds an extra layer of security and prevents unauthorized users from accessing data.

5. Reliability:

Consistency: The application should function consistently across different tasks, image formats, and operating environments. This consistency ensures that the application behaves predictably and meets user expectations.

Error Recovery: The system should gracefully handle errors, such as image format issues or corrupted data, and provide clear error messages. It should also offer recovery options, such as reverting changes or reloading the original image.

6. Maintainability:

Modularity: The system should be designed with modular components that are easy to update, maintain, or replace. This makes it easier to add new features or fix issues without affecting other parts of the application.

Documentation: Comprehensive code and system documentation should facilitate maintenance and updates by providing clear explanations of the system's architecture, functions, and dependencies.

7. Compatibility:

Operating System Compatibility: The application should be tested and compatible with different operating systems such as Windows, macOS, and Linux. This ensures a broader user base and ease of deployment.

Image Format Compatibility: The system should support multiple common image formats such as PNG, JPG, and BMP. This increases the range of images the application can work with and enhances user convenience.

8. Portability:

Cross-Platform Support: The application should work across different platforms with minimal changes, allowing for easy deployment and use in diverse environments.

Environment Independence: The application should avoid reliance on specific hardware or software environments. This ensures portability across different setups.

9. Scalability:

Handle Multiple Users: If the application is deployed as a web service, it should handle multiple users concurrently without significant performance degradation.

Processing Capacity: The system should scale its processing capacity to handle increased workloads efficiently. This might involve distributed systems or cloud-based solutions.

10. User Feedback:

Feedback Mechanisms: The application should offer users ways to provide feedback on their experience, such as reporting bugs or suggesting improvements.

Error Reporting: Users should be able to report errors easily, and the system should log these reports for further analysis and debugging. This helps improve the system over time and enhances user satisfaction.

Meeting these non-functional requirements ensures the system's efficiency, reliability, and user satisfaction, making the project successful and sustainable in the long run.

4 SYSTEM DESIGN

System design components for an image steganography project using the least significant bit (LSB) technique are provided below:

1. Software Architecture:

Client-Server Architecture (Optional): This approach involves separating the client-side and server-side responsibilities. The client handles user interface, user input, and output display, while the server performs data processing and storage. If implemented, this architecture allows for centralized data management and scalability.

Monolithic or Modular Design: In a monolithic design, the entire application is built as one cohesive unit. A modular design breaks the application into independent modules that perform specific tasks. Modular design is preferred for ease of maintenance, testing, and future enhancements.

2. User Interface:

Graphical User Interface (GUI): The application should provide a clear and user-friendly interface for users to interact with. This includes buttons for actions such as loading and saving images, text input for hidden data, and controls for hiding and revealing data. The GUI should follow best practices in usability and accessibility.

Input and Output Management: The system should handle user inputs and outputs effectively. This includes loading images from files, displaying hidden data, and saving images with hidden data. It should support user interaction through text fields, buttons, and menus.

3. Data Flow:

Input Handling: The system receives inputs such as image files and text data from the user. The application should validate and sanitize these inputs to prevent errors during processing.

Data Embedding: The system uses the LSB technique to hide data within the image. This involves modifying the least significant bits of the pixel values in the image with the data to be hidden.

Data Extraction: The system extracts hidden data from an image by reading the least significant bits of the pixel values and reconstructing the hidden data.

4. Components:

Image Processing: This component handles tasks such as loading, displaying, and saving images, as well as applying the LSB technique to hide or reveal data within the image. Efficient image processing is essential for a responsive user experience.

Text Processing: This component manages the input and output of text data, including any optional encryption or decryption. It may include functionality for handling different character encodings.

Error Handling: The system should handle errors gracefully and provide clear, helpful error messages. It should also have mechanisms for logging errors and exceptions for further analysis.

5. Data Storage:

Temporary Storage: The application may use temporary storage for intermediate data during processing, such as images with hidden data. Temporary storage should be managed efficiently to minimize resource usage.

Persistent Storage: Users may save images with hidden data to persistent storage, such as the local file system or a database. The system should handle persistent storage securely to prevent unauthorized access to data.

6. Security:

Encryption and Decryption (Optional): If implemented, encryption can secure the hidden data within images. Decryption should be available when revealing data. Strong encryption algorithms should be used to ensure data security.

Key Management: Proper handling of encryption keys is essential for securing data. The system should manage keys securely, including storage, access, and usage policies.

7. External Libraries and APIs:

Image Libraries: The application may use external libraries for image processing, such as the Python Imaging Library (PIL) or OpenCV. These libraries provide robust and optimized functions for handling images.

Steganography Libraries: Libraries such as Stegano offer functions for hiding and revealing data in images using the LSB technique. Integrating such libraries can simplify the development process and improve reliability.

8. Testing and Quality Assurance:

Unit Testing: Individual components of the system should be tested in isolation to ensure they work as expected. This includes testing functions for image and data processing.

Integration Testing: The interactions between different components should be tested to verify they work together seamlessly. This includes testing data flow and communication between modules.

User Acceptance Testing: The system should be tested by end-users to ensure it meets their needs and expectations. Feedback from these tests can inform improvements and adjustments.

9. Documentation:

Code Documentation: Detailed comments and explanations in the codebase should make the system easier to understand and maintain. This documentation should cover functions, classes, and interactions between components.

User Documentation: User manuals and tutorials should guide users on how to use the application effectively. Documentation should include step-by-step instructions and troubleshooting tips.

10. Maintenance and Future Enhancements:

Maintainability: The system should be designed with maintainability in mind, including clear code structure, modular design, and comprehensive documentation. This allows for easier updates and bug fixes.

Future Enhancements: The system should be flexible enough to accommodate future enhancements, such as supporting additional image formats, data hiding techniques, or other features. Design considerations should include extensibility and adaptability.

By adhering to these design principles, the image steganography project can be developed to offer robust and efficient functionality, a user-friendly interface, and the potential for future enhancements and scalability.

4.1 SYSTEM ANALYSIS

System analysis involves examining the requirements, current processes, and potential improvements in an image steganography project using the least significant bit (LSB) technique. This analysis helps identify how the system should be structured and what functionalities it needs to include. Here's a breakdown of the system analysis for the project:

1. Requirements Analysis:

Functional Requirements: Identifying what specific tasks the system must perform, such as hiding data within images, revealing hidden data, and handling different image formats.

Non-functional Requirements: Determining the qualities the system should possess, such as performance, security, usability, and scalability.

User Requirements: Understanding what end-users need from the system, including ease of use, data protection, and clear feedback mechanisms.

2. Current Processes:

Existing Systems: Analyzing any existing steganography systems to identify their strengths and weaknesses, such as speed, data protection, and ease of use.

Data Flow: Understanding the current data flow within the system, from user input to data processing to output.

3. Data Analysis:

Data Sources: Identifying the sources of data, such as image files and text data, and their characteristics, such as format, size, and quality.

Data Processing: Examining the steps involved in processing data, such as image manipulation and data hiding or extraction.

Data Storage: Evaluating how data is stored, both temporarily and permanently, and how it can be accessed securely.

4. Performance Analysis:

Processing Time: Assessing how long it takes for the system to hide and reveal data, and how it scales with different data sizes and image resolutions.

Resource Usage: Evaluating the system's use of resources such as CPU, memory, and storage, particularly during intensive tasks.

5. Security Analysis:

Data Protection: Analyzing the methods used to protect hidden data and user data, such as encryption and secure storage.

Authentication and Authorization: Examining any authentication mechanisms used to control access to the system and hidden data.

6. Usability Analysis:

User Interface: Evaluating the user interface's ease of use, including layout, navigation, and accessibility features.

Error Handling: Assessing how the system handles errors, such as invalid data inputs or processing failures, and the clarity of error messages.

7. Scalability and Compatibility Analysis:

System Scalability: Analyzing the system's ability to handle increasing workloads, such as larger data volumes or more concurrent users.

Compatibility: Evaluating the system's compatibility with different operating systems, image formats, and hardware environments.

8. Maintenance and Future Enhancements:

Maintainability: Assessing how easy it is to maintain and update the system, including modular design and comprehensive documentation.

Future Enhancements: Identifying potential future enhancements, such as additional features or support for new data formats and image types.

9. Testing and Quality Assurance:

Testing Methods: Analyzing the types of testing the system undergoes, such as unit, integration, and user acceptance testing.

Quality Assurance: Evaluating the overall quality assurance processes in place to ensure the system meets specified standards and user expectations.

10. Documentation Analysis:

Code and System Documentation: Reviewing the quality and comprehensiveness of code comments, system documentation, and user guides.

User Feedback Mechanisms: Analyzing how user feedback is collected and used to improve the system.

System analysis is crucial for understanding the current state of the project, identifying areas for improvement, and informing the design and implementation of the system. By thoroughly analyzing the above aspects, the project can be developed to meet user needs and achieve the desired functionality and performance.

4.2 ARCHITECTURE DIAGRAM

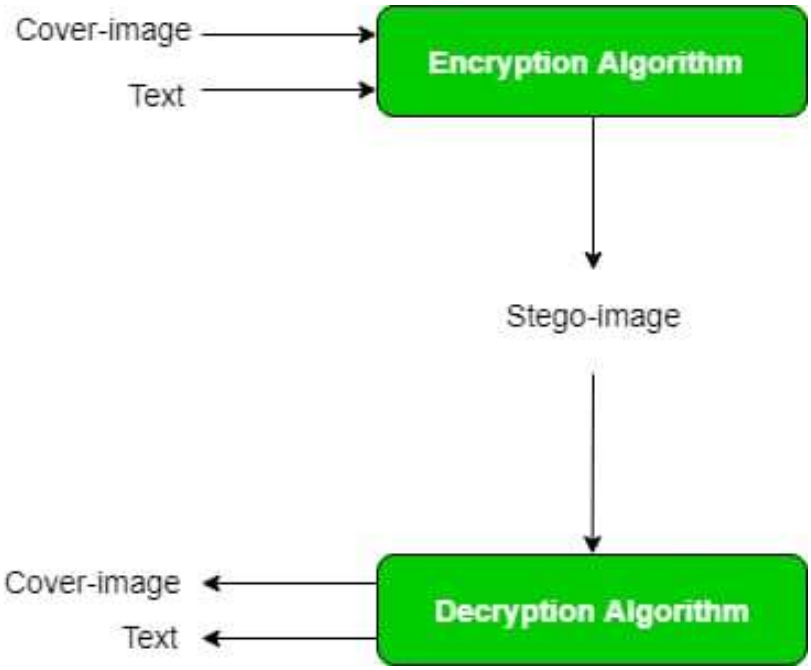

Figure 4.2.1: *Block diagram*

System Flow Diagram:

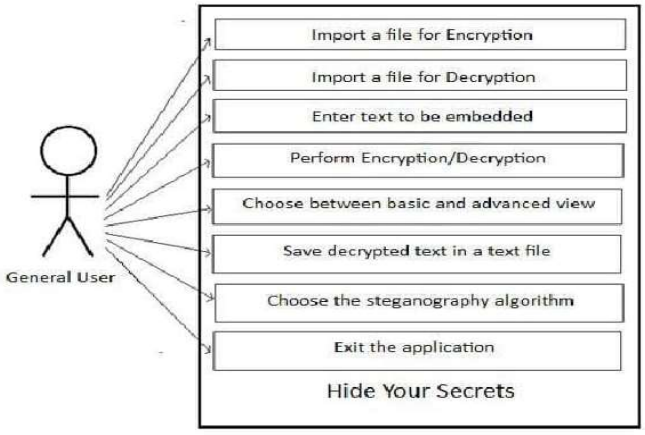

Figure 4.2.2: *System flow diagram*

Flowchart outlining the general steps involved in using a steganography application. Here's a breakdown of the steps:

Import a file for Encryption/Decryption: This step allows you to select a file that you want to hide or reveal hidden data within.

Import a file for Decryption: This step is likely for if you're trying to extract hidden data from a file and it requires a separate key file for decryption.

Enter text to be embedded: This step allows you to input the secret message you want to hide within the cover file.

Perform Encryption/Decryption: This step likely refers to the process of hiding the secret message within the cover file or vice versa, depending on whether you're encrypting or decrypting.

Choose between basic and advanced view: This option allows you to choose a simplified or more complex user interface.

Save decrypted text in a text file: This step allows you to save the extracted secret message as a separate text file.

Choose the steganography algorithm: This step allows you to select the specific algorithm you want to use to hide the data within the cover file. Different algorithms

may offer different advantages and disadvantages in terms of hiding capacity and imperceptibility.

Exit the application: This step terminates the program.

Hide Your Secrets: This is the overall title of the flowchart, likely referring to the application's purpose of steganography, which is the art of hiding data within other media.

4.3 DATA FLOW DIAGRAM

Data flow diagrams are the basic building blocks that define the flow of data in a system to the particular destination and difference in the flow when any transformation happens. It makes whole procedure like a good document and make simple and easy to understand for both programmers and non-programmers by dividing into the sub process.The data flow diagrams are the simple blocks that reveal the relationship between various components of the system and provide high level overview, boundaries of particular system as well as provide detailed overview of system elements.

Process: Process defines the source from where the output is generated for the specified input. It states the actions performed on data such that they are transformed, stored or distributed.

Data store: It is the place or physical location where the data is stored after extraction from the data source.

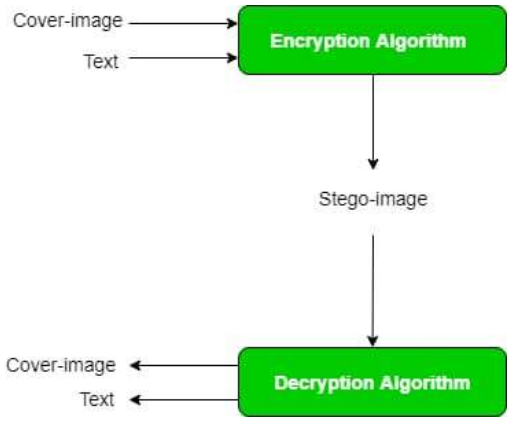

Figure 4.3.1: *Data Flow Diagram*

4.3.1 Level 0 Data Flow Diagram

'DFD level 0' is the highest-level view of the system, containing only one process which represents the whole function of the system. It doesn't contain any data stores and the data is stored within the process.

For constructing DFD level 0 diagram for the proposed approach we need two sources one is for 'source' and another is for 'destination' and a 'process'.

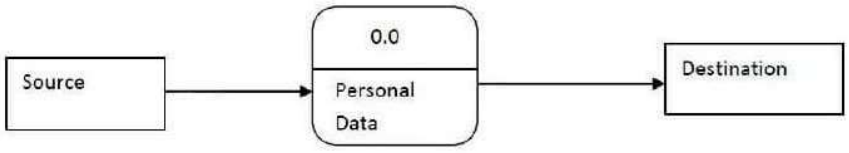

Figure 4.3.1.1: *Level 0 Data Flow Diagram*

DFD level 0 is the basic data flow process, the main objective is to transfer the data from sender to receiver after encryption.

4.3.2 Level 1 Data Flow Diagram

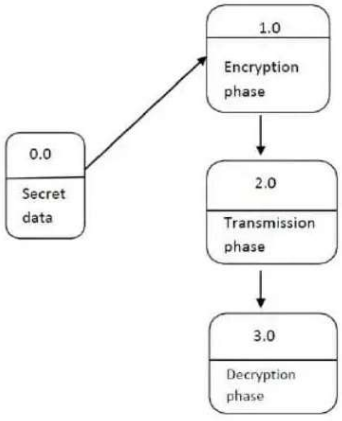

Figure 4.3.2.1: *Level 1 Data Flow Diagram*

In this data flow diagram, the secret data is sent to the encryption phase for embedding the data into the image for generating the carrier image. In the next phase the carrier image is sent to the decryption phase through the transmission phase. The final phase is the decryption phase where the data is extracted from the image and displays the original message.

4.3.3 Level 2 Data Flow Diagram

The image and the text document are given to the encryption phase. The encryption algorithm is used for embedding the data into the image.

The resultant image acting as a carrier image is transmitted to the decryption phase using the transmission medium. For extracting the message from the carrier image, it is sent to the decryption section. The plain text is extracted from the carrier image using the decryption algorithm.

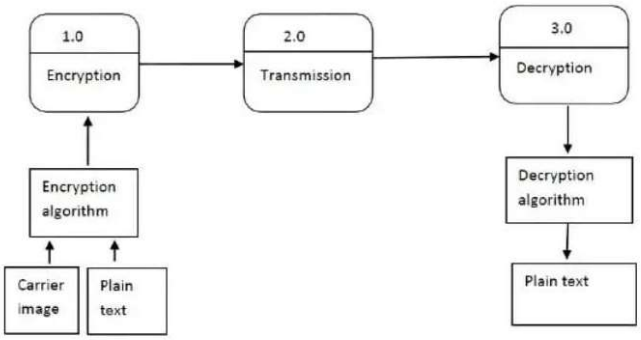

Figure 4.3.3.1: *Level 2 Data Flow Diagram*

4.4 UML DIAGRAMS

Creating Unified Modeling Language (UML) diagrams for an image steganography project using the least significant bit (LSB) technique involves visualizing the system's architecture, data flow, and interactions. Below are the types of UML diagrams that can be useful for this project, along with brief explanations of what each diagram represents:

4.4.1 Use case diagram

Overview: A use case diagram provides an overview of the system's functionality from the user's perspective. It shows different user roles (actors) and how they interact with the system's various functions (use cases).

Use Cases: Common use cases in an image steganography project might include:

Loading an image, hiding data in an image, revealing data from an image, Saving an image with hidden data, Error handling and reporting.

Actors: Possible actors in the system include the end-user and the system itself (for automated processes).

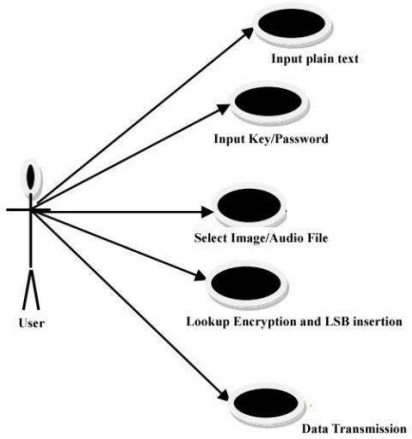

Figure 4.4.1.1: *Use case diagram*

4.4.2 Class Diagram

Overview: A class diagram shows the static structure of the system by representing classes, their attributes, methods, and relationships. It illustrates how different components of the system interact at a high level.

Classes:

Classes in the project might include:

ImageProcessor: Handles loading, saving, and manipulating images.

Steganography: Manages hiding and revealing data using the LSB technique.

TextProcessor: Processes text data and may handle encryption/decryption.

UserInterface: Manages the user interface and user input.

Relationships: The diagram shows relationships such as inheritance, composition, and association between classes.

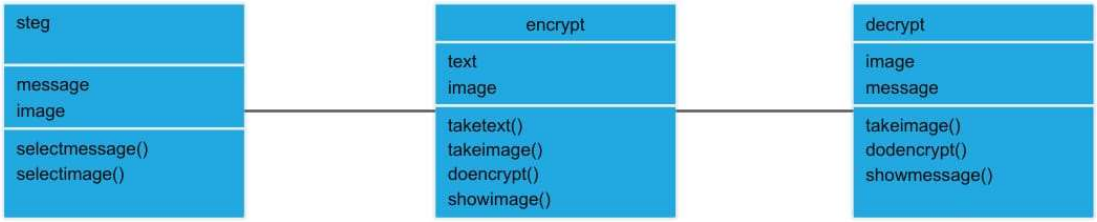

Figure 4.4.2.1: *Class diagram*

4.4.3 Activity Diagram

Overview: An activity diagram shows the workflow and the sequence of actions in a process. It is useful for visualizing how different functions are carried out and how data flows through the system.

Example: An activity diagram for hiding data in an image might show the user selecting an image, providing text data, calling functions to embed the data, and saving the modified image.

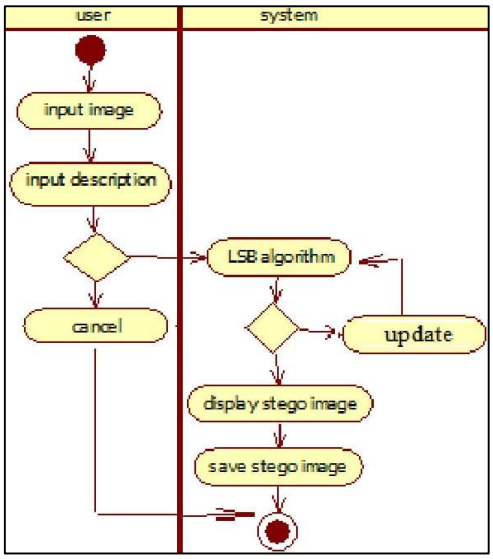

Figure 4.4.3.1: *Activity diagram*

5 PROPOSED METHOD

The proposed method for an image steganography project using the least significant bit (LSB) technique involves concealing secret data within digital images without noticeable changes to the visual quality of the image. This is achieved by manipulating the least significant bits of the image's pixel values to encode the secret data. The proposed method can be broken down into the following steps:

1. Image Loading and Preparation:

Loading: The system begins by loading a digital image file specified by the user. This file can be in various common formats such as PNG, JPEG, BMP, etc.

Verification and Conversion: Once the image is loaded, the system verifies that it is a suitable file type for the steganography process and that it meets any format requirements, such as being uncompressed. Conversion may be performed to achieve the desired format and color mode, such as RGB or grayscale, which will be essential for further processing.

2. Data Encoding:

Input of Secret Data: The user provides the data to be hidden, which can be text or other binary data such as files.

Data Conversion: The secret data is converted into binary form to prepare for embedding. This may involve encoding text into ASCII or UTF-8 binary representation.

Encoding Process: The binary data is ready for embedding within the least significant bits of the image’s pixel values. The encoding process is controlled by the chosen steganography technique, usually embedding one or more bits per pixel.

3. Data Embedding:

Sequential Embedding: The system embeds the binary data sequentially into the image’s pixel values. This means modifying the least significant bit (or bits) of each pixel in the image to encode a portion of the secret data.

Capacity Check: Before embedding begins, the system checks if the image has enough capacity to hide the entire secret data. Insufficient capacity may require the user to choose a different image.

Error Handling: If there is a mismatch in data size or any other embedding error, the system should provide appropriate error messages and offer options such as choosing a different image or data input.

4. Image Saving:

Output Options: Once data embedding is complete, the modified image can be saved as a new file or overwrite the original file, depending on user preference.

File Format: The user may have the option to specify the output format for the saved image, which could be the same as the input format or different (e.g., converting JPEG to PNG).

5. Data Retrieval:

Input of Modified Image: The user provides the modified image file as input to the system.

Data Extraction: The system reads the least significant bits of the image's pixel values to extract the embedded binary data.

Data Reconstruction: The extracted binary data is converted back to its original form, such as ASCII or UTF-8 text, depending on the type of data that was hidden.

6. Error Handling and Validation:

Error Messages: The system should offer clear and descriptive error messages in case of any issues, such as invalid file types, unsupported formats, or insufficient capacity.

Data Validation: When retrieving data, the system should validate the output to ensure it matches the expected format or type.

5.1 ALGORITHMS

The least significant bit (LSB) embedding technique is the algorithm used in the image steganography project. This technique allows secret data to be hidden within digital images by modifying the least significant bits of pixel values. Below is an in-depth explanation of the algorithm's steps:

1. Image Loading and Preparation:

Loading: The system begins by loading a digital image file from the user's specified path. This file can be in common formats such as PNG, JPEG, or BMP.

Image Format: Ensure the image format is suitable for steganography. For example, uncompressed formats (e.g., BMP) are preferred over compressed formats (e.g., JPEG) because compression may alter the embedded data.

2. Data Encoding:

Secret Data Input: The user inputs the secret data to be hidden, such as text or binary data.

Data Conversion: The secret data is converted into binary form (e.g., ASCII or UTF-8 encoding for text).

Binary Length Check: Determine the length of the binary data to check if the image can accommodate it.

3. Data Embedding:

Capacity Check: Calculate the total number of least significant bits available in the image and verify that it can hide the secret data.

Embedding Process: Iterate through the pixels in the image, embedding one or more bits of the secret data in the least significant bits of each pixel's color channels (e.g., RGB or grayscale).

Maintain Quality: The algorithm must ensure that the visual quality of the image is maintained during the embedding process.

4. Image Saving:

Save Modified Image: Once data embedding is complete, the modified image with hidden data is saved as a new file.

Output File: The output file can be saved with the same or different file format, depending on the user's choice.

5. Data Retrieval:

Load Modified Image: The user provides the modified image file as input to the system.

Extract Data: The system reads the least significant bits of each pixel in the image and retrieves the hidden binary data.

Data Reconstruction: Convert the extracted binary data back to its original form (e.g., text or binary file).

6. Error Handling and Validation:

Error Handling: Provide clear error messages in case of invalid input (e.g., incompatible file type) or issues with embedding/retrieving data.

Validation: Validate the size and format of secret data and image file before processing.

7. Security Considerations:

Encryption: Optionally encrypt the secret data before embedding and decrypt after extraction for added security.

Access Control: Implement measures to restrict unauthorized access to the hidden data.

8. Performance Optimization:

Efficiency: Optimize the algorithm for performance, ensuring efficient handling of large image files and data.

The image steganography project using the least significant bit (LSB) technique relies on specific algorithms to hide and reveal secret data within digital images. Here are the main algorithms used in the project, along with explanations of each:

Algorithm for Hiding Data:

1. Input:

An image file (in formats such as PNG, JPEG, or BMP).

Secret data (text or binary data) to be hidden.

2. Convert Secret Data:

Convert the secret data into binary form (e.g., ASCII or UTF-8 encoding for text data).

3. Load Image:

Load the image file and read its pixel data.

4. Check Image Capacity:

Calculate the total number of least significant bits available in the image.

Check if the image has sufficient capacity to hide the secret data.

5. Embed Data:

Iterate through each pixel of the image.

For each pixel, modify the least significant bit(s) to embed a portion of the secret data.

Continue embedding until all the secret data is hidden within the image.

6. Save Modified Image:

Save the modified image with hidden data as a new file.

7. Output:

Return the path to the saved image file with hidden data.

Pseudocode for Hiding Data:

plaintext

```
function hide_data(image_path, secret_data):
    # Load the image
    image = load_image(image_path)
    # Convert secret data to binary form
    binary_data = convert_to_binary(secret_data)
    # Check if the image has enough capacity
    if not check_capacity(image, binary_data):
        raise Exception("Image capacity insufficient")

    # Embed the binary data into the image
    index = 0
    for pixel in image.pixels:
        if index < length(binary_data):
            # Modify the least significant bit(s) of the pixel to hide binary data
            pixel = embed_binary_in_pixel(pixel, binary_data[index])
            index += 1
    # Save the modified image
    save_image(image, "output_image.png")
    return "output_image.png
```

Algorithm for Revealing Data:

1. Input:

An image file with hidden data.

2. Load Image:

Load the image file and read its pixel data.

3. Extract Data:

Iterate through each pixel of the image.

For each pixel, extract the least significant bit(s) to retrieve hidden binary data.

Continue extracting until the end of the hidden data is reached (e.g., using a predefined delimiter).

4. Convert Binary Data:

Convert the extracted binary data back into its original form (e.g., text or binary file).

5. Output:

Return the revealed secret data.

Pseudocode for Revealing Data:

plaintext

```
function reveal_data(image_path):
    # Load the image
    image = load_image(image_path)
    # Initialize an empty string for binary data
    binary_data = ""
    # Extract binary data from the image
    for pixel in image.pixels:
```

```
            # Retrieve least significant bit(s) from the pixel
            binary_data += extract_binary_from_pixel(pixel)
    # Convert binary data to its original form (e.g., text)
    secret_data = convert_from_binary(binary_data)
    return secret_data
```

Advantages of LSB Embedding:Minimal Visual Distortion: Because the LSBs only slightly influence the color intensity, embedding data in them results in minimal visible changes to the image.Simplicity: The technique is relatively straightforward and easy to implement.High Capacity: The method can hide a large amount of data, especially when using multiple channels in an RGB image.

Example:

Embedding: If you have a pixel with RGB values (45, 78, 112) and a binary data bit '1' to hide, you could modify the LSB of the red value. Changing the red value from 45 (binary: 00101101) to 44 (binary: 00101100) embeds the data bit '1' in the least significant bit of the red value.Revealing: To retrieve the hidden data, extract the LSBs of the image's pixel values and convert them back to their original format.The LSB embedding technique is a widely used method in steganography projects due to its simplicity and efficiency. However, for sensitive applications, you may need to consider more advanced methods or encryption to enhance security.

5.2 WORKFLOW

To provide a deep and in-depth explanation of the workflow of an image steganography project using the least significant bit (LSB) technique, let's examine each step-in detail:

1. Input Image and Secret Data:

User Selection:

The user starts the workflow by selecting a digital image file from their system using a file dialog, commonly supported by graphical interfaces such as Tkinter.

The user can also input the secret data to be hidden within the image. This data can be text or binary data, such as a file.

2. Load Image:

Image Loading: The system reads the selected image file and loads its pixel data into memory for processing.The image file format (e.g., PNG, JPEG, BMP) is verified for compatibility with the steganography process.

Image Validation: The system checks the image's resolution, color depth, and format to ensure it can support the LSB technique.

3. Convert Secret Data:

Binary Conversion: The secret data provided by the user is converted into binary form. For text data, this may involve converting it to ASCII or UTF-8 binary representation.

Length Calculation: Calculate the length of the binary data to determine how much data will be embedded into the image.

4. Check Image Capacity:

Capacity Evaluation: Determine the available embedding capacity of the image by calculating the total number of least significant bits across all pixels.

Comparison: Compare the length of the binary data with the available capacity of the image to check if there is sufficient room to embed the data.

5. Data Embedding:

Iterative Embedding Iterate through the image's pixels and modify the least significant bit(s) of each pixel to embed portions of the binary data.

Use a controlled approach to avoid visibly altering the image's appearance.

Channel Selection: In RGB images, embedding can occur in one or more color channels (red, green, blue). In grayscale images, embedding occurs in the single channel.

Data Continuity: Continue embedding binary data into the pixels until all secret data is hidden within the image.

6. Save Modified Image:

File Saving: Once data embedding is complete, save the modified image file as a new file.

Verification: Verify that the modified image has been saved correctly, and the data has been embedded as intended.

7. Retrieve Hidden Data:

Input Image: The user provides the modified image file that contains the hidden data.

Data Extraction: The system loads the modified image and iterates through its pixels.

Data Continuity: Continue extracting data until the entire hidden data has been retrieved.

8. Convert Binary Data:

Reconstruction: Convert the extracted binary data back to its original form, such as text or a binary file.

Output: Provide the retrieved secret data to the user in its original format.

9. Error Handling and Validation:

Error Messages: Provide clear and descriptive error messages to guide the user in case of invalid inputs or issues during data embedding or retrieval.

Data Validation: Validate the format and size of the secret data and the image file before processing.

10. Security and Optimization:

Security Measures: Optionally, implement security features such as encrypting the secret data before embedding and decrypting after retrieval.

Access Control: Include access control measures to restrict unauthorized access to the hidden data.

Performance Optimization: Optimize the process for speed and efficiency, especially when handling large image files.

5.3 SELECTION OF THE SOFTWARE

When selecting software for an image steganography project using the least significant bit (LSB) technique, you need to consider several factors such as compatibility, performance, ease of use, and any additional features that may enhance the project's functionality. Here are some considerations and recommendations for the software components that can be used in this project:

1. Programming Language:

Python: Python is a popular choice due to its simplicity and extensive library support for image processing and graphical user interfaces.

Libraries like PIL (Pillow) are helpful for image manipulation.

Tkinter can be used for creating graphical user interfaces.

Libraries like stegano offer built-in functions for LSB steganography.

2. Image Manipulation Libraries:

Pillow (PIL): A popular Python library for image manipulation and processing. Provides support for a variety of image formats and easy access to pixel data.

OpenCV: A powerful image processing library that supports advanced image manipulation and computer vision capabilities.

3. Steganography Libraries:

Stegano: A Python library that provides simple tools and methods for steganography using various algorithms, including LSB embedding.

4. Graphical User Interface (GUI) Libraries:

Tkinter: Python's standard GUI library, which is easy to use and integrate with other Python code.

PyQt or PySide: These are advanced GUI libraries for Python that offer more sophisticated and modern UI design options compared to Tkinter.

5. File Dialog Libraries:

Tkinter or PyQt File Dialogs: These libraries offer file dialog support, allowing the user to select images and other files conveniently.

6. Encryption Libraries (Optional):

Cryptography: This Python library provides a variety of encryption tools to enhance the security of hidden data before embedding and after extraction.

7. Other Software Considerations:

IDE: Use an integrated development environment (IDE) such as PyCharm, VS Code, or Jupyter Notebook for efficient coding and testing.

Version Control: Utilize version control software such as Git to manage code changes and collaborate effectively.

8. Factors to Consider in Software Selection:

Compatibility: Ensure that the software libraries and tools work well together and are compatible with your development environment.

Performance: Choose software that offers efficient processing of images, especially when dealing with large files.

Ease of Use: Select software with user-friendly interfaces and straightforward APIs to minimize development time.

Support and Documentation: Opt for libraries and tools that have strong community support and thorough documentation.

6 IMPLEMENTATION

Implementation of an image steganography project using the least significant bit (LSB) technique in PythonThe code includes a graphical user interface (GUI) built with tkinter for easy user interaction. The Pillow library is used for image manipulation and stegano for the LSB steganography technique. The implementation covers hiding and revealing secret data within an image file.

Dependencies:

Python 3.x

Pillow (pip install Pillow)

Stegano (pip install stegano)

6.1 SAMPLE CODING

6.1.1 Importing Image Code:

It sets the selected file's path to the filename attribute of the class instance. The file dialog opens at the current working directory , displaying only files with specified image file extensions (.png, .jpg, .jpeg, .bmp).

```
from tkinter import *
from tkinter import filedialog
from PIL import Image, ImageTk
import os
from stegano import lsb

root = Tk()
root.title(" Major Project - Image Steganography")
root.geometry("700x500+150+180")
root.resizable(False, False)
root.configure(bg="#F7F700")


def showimage():
    global filename
    filename = filedialog.askopenfilename(initialdir=os.getcwd(),
                                          title='Select Image File',
                                          filetypes=(("PNG file", "*.png"),
                                                     ("JPG File", "*.jpg"),
                                                     ("All Files", "*.*")))
    img = Image.open(filename)
    img = ImageTk.PhotoImage(img)
    Ibl.configure(image=img)
    Ibl.image = img
```

Figure 6.1.1.1: *Importing Image Code*

show_image(): This function prompts the user to select an image file and displays the image in the GUI.

6.1.2 Encoding Code:

The encoding part, also known as hiding data, is implemented in the hide_data() function. This function performs the following steps:

Retrieves the secret data (text) from the text box in the GUI.

Checks if an image file is selected and the secret data is not empty.

Uses the lsb.hide() function from the stegano package to hide the secret data in the selected image file.

Stores the modified image (image with hidden data) as self.secret_data for later use.

6.1.3 Decoding Code:

The decoding part, also known as revealing hidden data, is implemented in the show_data() function. This function performs the following steps:

Checks if an image file is selected.

Uses the lsb.reveal() function from the stegano package to retrieve the hidden data from the selected image file.

Clears the text box and displays the retrieved data in the GUI.

```
def Hide():
    global secret
    message = text1.get(1.0, END)
    secret = lsb.hide(str(filename), message)
    return secret

def Show():
    clear_message = lsb.reveal(filename)
    text1.delete(1.0, END)
    text1.insert(END, clear_message)

def save():
    secret = Hide()
    secret.save("hidden.png")
```

Figure 6.1.3.1: *Encoding and Decoding data Code*

6.1.4 Graphical User Interface Code:

Initializes a graphical user interface (GUI) for a steganography application using Tkinter in Python. It sets icons for the window and logo for the application. The GUI is divided into four frames: the first frame displays the selected image, the second frame contains a text box for entering data, and the third and fourth frames hold buttons for various operations. Buttons in the third frame allow opening and saving images, while those in the fourth frame trigger hiding and showing data. Labels provide additional information, such as prompting users to select an image or indicating the purpose of the buttons. The main loop (root.mainloop()) keeps the GUI running. This layout facilitates user interaction and enables seamless execution of steganographic operations within the application.

```
# Icon
image_icon = PhotoImage(file="logo.jpg")
root.iconphoto(False, image_icon)
# Logo
logo = PhotoImage(file="computer.png")
Label(root, image=logo, bg="red").place(x=10, y=0)
Label(root, text="CYBER SECURITY", bg="#F7F700", fg="blue", font="arial 25 bold").place(x=110, y=14)
# First Frame
f = Frame(root, bd=3, bg="black", width=340, height=280, relief=GROOVE)
f.place(x=10, y=80)
Ibl = Label(f, bg="black")
Ibl.place(x=40, y=10)
# Second Frame
frame2 = Frame(root, bd=3, bg="white", width=340, height=280, relief=GROOVE)
frame2.place(x=350, y=80)

text1 = Text(frame2, font="Roboto 20", bg="white", fg="black", relief=GROOVE, wrap=WORD)
text1.place(x=0, y=0, width=320, height=295)

scrollbar1 = Scrollbar(frame2)
scrollbar1.place(x=320, y=0, height=300)
scrollbar1.configure(command=text1.yview)
text1.configure(yscrollcommand=scrollbar1.set)
# Third Frame
frame3 = Frame(root, bd=3, bg="#F7F700", width=330, height=100, relief=GROOVE)
frame3.place(x=10, y=370)

Button(frame3, text="Open Image", width=10, height=2, font="arial 14 bold", command=showimage).place(x=20, y=30)
Button(frame3, text="Save Image", width=10, height=2, font="arial 14 bold", command=save).place(x=180, y=30)
Label(frame3, text="Picture, Image, Photo, File", bg="black", fg="yellow").place(x=20, y=5)
# Fourth Frame
frame4 = Frame(root, bd=3, bg="#F7F700", width=330, height=100, relief=GROOVE)
frame4.place(x=360, y=370)

Button(frame4, text="Hide Data", width=10, height=2, font="arial 14 bold", command=Hide).place(x=20, y=30)
Button(frame4, text="Show Data", width=10, height=2, font="arial 14 bold", command=Show).place(x=180, y=30)
Label(frame4, text="Picture, Image, Photo, File", bg="black", fg="yellow").place(x=20, y=5)

root.mainloop()
```

Figure 6.1.4.1: *Graphical User Interface Code*

7 RESULTS

The Stegenographic schemes which were present for more than 1000 years were studied and analyzed in detail in this report. Various algorithms were analyzed, compared and implemented.

For designing the steganographic application, we worked on different phases like encryption, decryption and data transmission. An application for sending the personal data securely to the destination has been developed successfully.

The design phase is the primary phase, which gives a brief idea about the different levels used for developing an application with the help of block diagrams. The software is designed in a user-friendly manner. So, it is simple to use for developing a prototype of the application.

The most important phase in the project is the execution phase. The execution phase is developed with the help of the design phase. For executing the application, we worked on two sections: one is encryption and another is decryption. As we designed the program using the Python platform, the next part is debugging the program. We faced some problems when writing the code, but at last we were successful in executing the program without errors.

Figure 7.1: *Encoding and Decoding Process*

In this project we mainly concentrated on embedding the data into an image. We have designed the steganographic application which embedded the data into the image. Normally, after embedding the data into the image, the image may lose its resolution. In the proposed approach, the image remains unchanged in its resolution as well in size. The speed of embedding the data into the image is also high in the proposed approach such that the image is protected and the data to the destination is sent securely. For the decryption phase, we have used the same Python programming language for the purpose of designing. We have used security keys like personal passwords for protecting the image from unauthorized modification, which improved the security level.

Figure 7.2: *Original Image*

Figure 7.3: *Stego Image*

We have chosen image steganography because it is simple to use and its user friendly application. There are many applications for image hiding but the proposed approach is created using tkinter, pillow, stegano framework which is easier for coding and the performance is better compared to other languages.

8 CONCLUSION

The project utilizes image steganography to conceal personal data within images, bolstered by robust security measures and a secure password. Leveraging Python and libraries like tkinter, Pillow, and stegano, it ensures user-friendly interaction and efficient data concealment via the LSB technique. With encryption, decryption, and seamless data transmission, it provides a dependable solution for secure communication while maintaining image resolution integrity.

8.1 KEY TAKEAWAYS

Effective Data Hiding: The project effectively uses the LSB technique to embed data into images while maintaining the visual quality and integrity of the original images.

User-Friendly Interface: The graphical user interface (GUI) designed using tkinter allows users to easily navigate the application and perform steganographic operations without needing advanced technical knowledge.

Data Security: The application provides a secure method for hiding personal data within images and includes password protection to safeguard against unauthorized access.

Cross-Platform Compatibility: The project can be executed on various operating systems due to Python's cross-platform nature, allowing for broader accessibility and usability.

Performance and Efficiency: The project's choice of Python and associated libraries ensures efficient performance and execution, even when working with large images and significant volumes of data.

8.2 AREAS FOR FUTURE IMPROVEMENT

The image steganography project showcased effective data hiding within images using Python and libraries like tkinter, Pillow, and stegano, ensuring efficiency, security, and accessibility. To further elevate the application, research into advanced steganographic techniques, encryption methods, and machine learning can enhance security and robustness. Integration of blockchain for authentication and data integrity verification, along with support for different multimedia files and cloud services, may expand usability. While successful in its current state, future enhancements could propel the application to higher levels of security, performance, and usability, ensuring its relevance in evolving digital communication landscapes

9 FUTURE SCOPE

Image steganography, specifically using the least significant bit (LSB) technique, presents several opportunities for future development and improvement. As technology advances and digital communication becomes increasingly essential, there are multiple areas for expansion and enhancement of image steganography projects. Let's explore these areas in depth:

1.Enhanced Security and Encryption: Implement AES or RSA encryption for hidden data and dynamic key management systems.

2.Improved Steganographic Techniques: Research and implement advanced techniques like DCT, DWT, and adaptive steganography.

3.Cross-Platform and Mobile Compatibility: Extend support for Windows, macOS, Linux, and develop mobile apps for Android and iOS.

4. Steganalysis Resistance: Develop techniques to resist steganalysis detection and enhance robustness against attacks.

5.Optimization and Performance: Optimize algorithms, utilize GPU acceleration, and improve processing speed.

6.User-Friendly Enhancements: Develop a more intuitive GUI, integrate drag-and-drop functionality, and cloud storage services.

7.Integration with Other Technologies: Explore blockchain integration for data authentication and machine learning for steganalysis resistance.

8. Compliance and Ethics Ensure compliance with legal regulations, establish clear usage policies, and promote ethical usage.

10 REFERENCES

[1] Alfred J. M et al., 1996. Hand book of applied Cryptography. First edition.

[2] Bloom, J. A. et al., 2008. Digital watermarking and Steganography. 2nd edition.

[3] A. Westfeld. "F5-A Steganographic Algorithm: High Capacity Despite Better Steganalysis", Lecture Notes in Computer Science, vol. 2137, pp. 289-302, 2001.

[4] X. Yu, Y. Wang, and T. Tan, "On Estimation of Secret Message Length in Steganography", JSteglike Proceedings of the 17th International Conference on Pattern Recognition, vol. 4, pp. 673-676, 2004.

[5] Q. Weiwei, G. Yanging, and K. Xiangwei. "JPEG Quantization-Distribution Steganalytic Method Attacking JSteg". International Journal of Computer Science and Network Security, vol. 6, pp. 192-195.

[6] Bandyopadhyay, S.K., 2010. An Alternative Approach of Steganography Using Reference Image. International Journal of Advancements in Technology, 1(1), pp.05-11.

[7] https://www.ijcaonline.org/volume1/number15/pxc387502.pdf

[8] S. William, Cryptography and Network Security: Principles and Practice, 2 edition, Prentice-Hall, Inc., 1999 pp 23-50

[9] https://www.jjtc.com/pub/r2026.pdf

[10] Hide & Seek: An Introduction to Steganography: Niles Provos and Peter Honey man, IEEE Security & Privacy Magazine, May/June 2003.

[11] Image Compression and Discrete Cosine Transform Ken Cabin and Peter Gent, Math 45 College of the Redwoods, 1998

[12] Steganography Primer - Ruid, Computer Academic underground. 2004

[13] Artz, D., "Digital Steganography: Hiding Data within Data", IEEE Internet Computing Journal, June 2001

[14] Owens, M.. "A discussion of covert channels and steganography", SANS Institute, 2002