***“StegaSafe: Image-Based Message Encryption with Least Significant Bit”***

**A Major Project Report Submitted to**

**Rajiv Gandhi Proudyogiki Vishwavidyalaya**

****

**Towards Partial Fulfillment for the Award of**

**Bachelor of Engineering in Computer Science Engineering**

***Submitted by:***

***Guided by:***

**Aaditya Panwar (0827CS211001)**

**Anusha Nagar (0827CS211029)**

**Anushka Patel (0827CS211030)**

**Prof. Shraddha Sharma**

**Professor, CSE**

****

***Acropolis Institute of Technology & Research, Indore***

**July - Dec 2024**

EXAMINER APPROVAL

The Major Project entitled ***“StegaSafe: Image-Based Message Encryption”*** submitted by **Aaditya Panwar (0827CS211001), Anusha Nagar (0827CS211029), Anushka Patel (0827CS211030)** has been examined and is hereby approvedtowards partial fulfillment for the award of ***Bachelor of Technology*** ***degree in Computer Science Engineering*** discipline, for which it has beensubmitted. It understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed, or conclusion drawn therein, but approve the project only for the purpose for which it has been submitted.

**(Internal Examiner)** **(External Examiner)**

**Date:** **Date:**

RECOMMENDATION

This is to certify that the work embodied in this major project entitled ***“StegaSafe: Image-Based Message Encryption”*** submitted by

**Aaditya Panwar (0827CS211001), Anusha Nagar (0827CS211029), Anushka Patel (0827CS211030)** is asatisfactory account of the bonafide work done under the supervision of ***Prof. Shraddha Sharma***, is recommended towards partial fulfillment forthe award of the Bachelor of Technology (Computer Science Engineering) degree by Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal.

**(Project Guide)**

**(Project Coordinator)**

**(Dean Academics)**

STUDENTS UNDERTAKING

This is to certify that the major project entitled ***“StegaSafe: Image-Based Message Encryption”*** has developed by us under the supervision of ***Prof. Shraddha Sharma***. The whole responsibility of the work done in this project is ours.The sole intension of this work is only for practical learning and research.

We further declare that to the best of our knowledge; this report does not contain any part of any work which has been submitted for the award of any degree either in this University or in any other University / Deemed University without proper citation and if the same work found then we are liable for explanation to this.

**Aaditya Panwar (0827CS211001)**

**Anusha Nagar (0827CS211029)**

**Anushka Patel (0827CS211030)**

Acknowledgement



We thank the almighty Lord for giving me the strength and courage to sail out through the tough and reach on shore safely.

There are number of people without whom this project would not have been feasible. Their high academic standards and personal integrity provided me with continuous guidance and support.

We owe a debt of sincere gratitude, deep sense of reverence and respect to our guide and mentor **Prof. Shraddha Sharma,** Professor, AITR, Indore for his motivation, sagacious guidance, constant encouragement, vigilant supervision, and valuable critical appreciation throughout this project work, which helped us to successfully complete the project on time.

We express profound gratitude and heartfelt thanks to **Dr. Kamal** **Kumar Sethi**, Professor & Head CSE, AITR Indore for his support, suggestion, andinspiration for carrying out this project. I am very much thankful to other faculty and staff members of the department for providing me all support, help and advice during the project. We would be failing in our duty if do not acknowledge the support and guidance received from **Dr. S. C.** **Sharma**, Director, AITR, Indore whenever needed. We take opportunity toconvey my regards to the management of Acropolis Institute, Indore for extending academic and administrative support and providing me all necessary facilities for project to achieve our objectives.

We are grateful to **our parent** and **family members** who have always loved and supported us unconditionally. To all of them, we want to say “Thank you”, for being the best family that one could ever have and without whom none of this would have been possible.

**Aaditya Panwar (0827CS211001), Anusha Nagar (0827CS211029), Anushka Patel (0827CS211030)**

Executive Summary



***StegaSafe: Image-Based Message Encryption***

This project is submitted to Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal (MP), India for partial fulfillment of Bachelor of Engineering in Information Technology branch under the sagacious guidance and vigilant supervision of ***Prof. Shraddha Sharma***.

The *StegaSafe* project is based on steganography, a technique used to embed secret information within digital media to protect the confidentiality of messages. Unlike traditional encryption, steganography conceals the presence of a message altogether, enhancing privacy and security. This project employs the Python programming language, using the *Kivy* framework to develop an interactive GUI and the *Stegano* library to handle image-based encoding and decoding of hidden messages.

*StegaSafe* aims to offer a straightforward yet robust tool that enables users to hide messages within images, making it suitable for scenarios requiring discreet communication. By leveraging the ease of Python and the specialized functionalities of Kivy and Stegano, this project combines accessibility with effectiveness in secure information exchange.

**Key words**: Steganography, Image Processing, Python, Kivy, Stegano

*"In a world of open messages, true security lies in what cannot be seen.  
To protect what matters, let us not only encrypt—but conceal."*

List of Figures



Figure 1-1: Private Messaging using StegaSafe 4

Figure 1-2: Corporate Communication with StegaSafe 5

Figure 1-3: Personal Security with StegaSafe 5

Figure 3-1: Internal Process 15

Figure 3-2: Use-Case Diagram 18

Figure 3-3: Activity Diagram 19

Figure 3-4: Block Diagram 21

Figure 4-1: User Interface of StegaSafe 25

Figure 4-2: After loading the image 25

Figure 4-3: After we encoded the message into the image 26

Figure 4-4: When we decode the message from a encoded image 26

Figure 4-5: Home screen with the selected image loaded and message

entered in the text field 28

Figure 4-6: Confirmation screen 28

Figure 4-7: Home screen with the encoded image loaded 29

Figure 4-8: Decoded message displayed in the text field after clicking

"Decode" 30

Figure 4-9: Encoding screen with image loaded and message entered

before saving 31

Figure 4-10: Confirmation screen 31

List of Tables



Table 4.1: Test Case 1 27

Table 4.2: Test Case 2 29

Table 4.3: Test Case 3 30

Table of Contents



|  |  |
| --- | --- |
| [**CHAPTER 1.**](#page14) | [**INTRODUCTION .........................................................................**](#page14)**.**[**1**](#page14) |
| [1.1](#page14) | [Overview................................................................................](#page14).[1](#page14) |
| [1.2](#page15) | [Background and Motivation ................................................](#page15).[2](#page15) |
| [1.3](#page15) | [Problem Statement and Objectives .....................................](#page15)..[3](#page15) |
| [1.4](#page16) | [Scope of the Project ..............................................................](#page16).[4](#page16) |
| [1.5](#page18) | [Team Organization ................................................................](#page18).[6](#page18) |
| [1.6](#page18) | [Report Structure ..................................................................](#page18).[6](#page18) |

|  |  |  |  |
| --- | --- | --- | --- |
| [**CHAPTER 2.**](#page20) | [**REVIEW OF LITERATURE .......................................................**](#page20)**.**[**8**](#page20) | | |
| [2.1](#page20) | [Preliminary Investigation.......................................................](#page20).[8](#page20) | | |
|  | [2.1.1 Current System ................................................................](#page20).[8](#page20) | | |
| [2.2](#page21) | [Limitations of Current System ..............................................9](#page21) | | |
| [2.3](#page21) | [Requirement Identification and Analysis for Project ..........10](#page21) | | |
| [2.4](#page21) | [Existing Techniques in Image Steganography......................11](#page21) | | |
| [2.5](#page21) | [Conclusion............................................................................12](#page21) | | |
|  |  | |  |
| [**CHAPTER 3.**](#page28) | [**PROPOSED SYSTEM ............................................................**](#page28)**....**[**13**](#page28) | | |
| [3.1](#page28) | [The Proposal ...................................................................](#page28)..[13](#page28) | | |
| [3.2](#page28) | [Benefits of the Proposed System......................................](#page28).[14](#page28) | | |
| [3.3](#page29) | [Block Diagram ...................................................................15](#page29) | | |
| [3.4](#page29) | [Feasibility Study ................................................................16](#page29) | | |
|  | | [3.4.1](#page29) [Technical ........................................................................16](#page29) | |
|  | | [3.4.2](#page30) [Economical.....................................................................16](#page30) | |
|  | | [3.4.3](#page30) [Operational ....................................................................16](#page30) | |
| [3.5](#page31) | [Deployment Requirements................................................17](#page31) | | |
|  | [3.5.1](#page33) [Hardware………………........................................................17](#page33) | | |
|  | [3.5.2](#page34) [Software ………….............................................................17](#page34) | | |
| [3.6](#page31) | [Design Representation.......................................................18](#page31) | | |
|  | [3.6.1](#page33) [Use-Case Diagram...........................................................18](#page33) | | |
|  | [3.6.2](#page34) [Activity Diagram.............................................................19](#page34) | | |
|  | [3.6.3](#page34) [Block Diagram................................................................20](#page34) | | |
|  |  | | |

|  |  |
| --- | --- |
| [**CHAPTER 4.**](#page36) | [**IMPLEMENTATION .................................................................22**](#page36) |
| [4.1](#page36) | [Technique Used .................................................................22](#page36) |
|  | [4.1.1](#page36) [Steganography................................................................22](#page36) |
|  | [4.1.2](#page37) [Least Significant Bit (LSB) Technique ..............................22](#page37) |
| [4.2](#page38) | [Tools Used .........................................................................23](#page38) |
|  | [4.2.1](#page38) [Stegano ...........................................................................23](#page38) |
|  | [4.2.2 PIL (Python Imaging Library) ..........................................23](#page39) |
|  | [4.2.3](#page40) [Kivy….. ............................................................................23](#page40) |
| [4.3](#page44) | [Language Used ..................................................................23](#page44) |
| [4.4](#page45) | [Implementation Process ...................................................24](#page45) |
| [4.5](#page45) | [Screenshots .......................................... ............................25](#page45) |
| [4.6](#page46) | [Testing ...............................................................................27](#page46) |
|  | [4.6.1](#page46) [Strategy Used.................................................................27](#page46) |
|  | [4.6.2](#page46) [Test Case and Analysis ...................................................27](#page46) |
|  |  |

|  |  |
| --- | --- |
| [**CHAPTER 5.**](#page49) | [**CONCLUSION.............................................................................32**](#page49) |
| [5.1](#page49) | [Conclusion ...........................................................................32](#page49) |
| [5.2](#page49) | [Limitations of the Work ......................................................32](#page49) |
| [5.3](#page50) | [Suggestion and Recommendations for Future Work ........33](#page50) |
|  |  |
| [**BIBLIOGRAPHY..........................................................................................................35**](#page51) | |
| [**PROJECT**](#page51) **PLAN** [**….......................................................................................................37**](#page51) | |
| [**GUIDE**](#page51) **INTERACTION SHEET**[**……..........................................................................38**](#page51) | |
| [**APPENDIX**](#page51) **A-SOURCE CODE**[**...................................................................................39**](#page51) | |
| [**APPENDIX**](#page51) **B-STE**[**GASAFE USER MANUAL...........................................................45**](#page51) | |
| [**APPENDIX**](#page51) **C-RESEARCH PAPER WITH CERTIFICATES.**[**..............................49**](#page51) | |
|  | |

**Chapter 1: Introduction**

Introduction



In the era of rapid digital communication, securing sensitive information from unauthorized access is a growing concern. While encryption methods have been traditionally employed to secure data, there remains a significant need for additional layers of protection to prevent detection. Steganography, the art of concealing information within ordinary files, offers a novel solution to this issue. By embedding messages within images, audio files, or other media formats, steganography allows for secure communication without drawing attention to the existence of the message itself. This project, StegaSafe, focuses on image-based message encryption using steganography, which allows users to hide messages inside digital images, ensuring the security of sensitive information.

The primary goal of *StegaSafe* is to offer a user-friendly platform for secure message embedding and extraction, utilizing Python libraries such as Stegano for encoding and Kivy for creating the graphical user interface (GUI). The project enables both novice and experienced users to easily encode messages within images and extract them without any noticeable alterations to the image quality. This adds a layer of stealth to communication, offering enhanced security.

**1.1 Overview**

The *StegaSafe* project revolves around the concept of image steganography, where information is hidden within an image, making the message undetectable to the human eye. This technique is advantageous in secure communication, especially in situations where the existence of the message must remain hidden.

In this project, we developed an application that can encode text messages within a selected image file and later extract the hidden messages using a decoding mechanism. The project uses Python as the primary

language for implementation, with libraries like Kivy to handle the graphical interface and Stegano to perform the steganographic encoding and decoding. The system allows users to select images, encode text messages, and save the modified images. The encoded image can be safely shared, and only the person who knows the method can extract the hidden message.

The project is built with the intention of providing both security and ease of use. It aims to simplify the process of securely communicating through images, making it accessible to users with minimal technical expertise. The hidden messages in the images are encoded in such a way that the changes to the image’s visual quality are imperceptible, ensuring that the encoded image appears just like any normal picture.

**1.2 Background and Motivation**

Steganography has a long history, with its origins dating back to ancient civilizations, where it was used to conceal messages in plain sight. Over time, as the need for data privacy has grown in the digital world, steganography has evolved alongside encryption techniques. Unlike traditional encryption methods, which make the existence of the message obvious to anyone who intercepts the communication, steganography seeks to hide the message itself. The primary motivation behind this project is to develop a practical and efficient tool that uses steganographic techniques to secure sensitive information in the form of messages hidden within images.

As the internet becomes increasingly interwoven into every facet of life, securing communication and data becomes even more important. With growing concerns over data breaches, surveillance, and unauthorized access, the need for tools that allow users to communicate securely without drawing attention to the information they are sharing is critical. *StegaSafe* was conceived to fill this gap by providing an easy-to-use and reliable tool for digital communication, making it a practical solution for individuals and organizations looking for additional layers of security in their

communications.

The motivation for this project stems from a need to address the challenge of hiding information effectively without compromising on image quality. Traditional methods often result in detectable alterations to the image, making the steganography vulnerable to detection. With *StegaSafe*, the goal is to provide a solution that can hide data within images without significantly altering their appearance, making it an ideal solution for secure communication.

**1.3 Problem Statement and Objectives**

In today’s digital age, the primary concern for many individuals and organizations is ensuring the confidentiality of their communications. With conventional encryption techniques, the existence of the message is often still visible to anyone who intercepts the data. Steganography offers a stealthier alternative by embedding messages within media files, which appear normal to anyone who views them. Despite the growing need for secure communication, there are few accessible tools that offer both ease of use and the ability to hide messages effectively without altering the file’s quality.

The primary objective of StegaSafe is to address this gap by creating an intuitive, secure, and efficient platform for encoding and decoding messages within images. The objectives of the project are:

1. **Objective 1:** To implement a method of hiding messages within digital images using steganography techniques, which ensures that the messages remain secure and undetectable to unauthorized users.
2. **Objective 2:** To develop a simple, user-friendly GUI that enables users to easily encode and decode messages without the need for technical expertise.
3. **Objective 3**: To maintain the integrity and quality of the image while encoding the message, ensuring that the image looks identical to the

original image without any visible distortions.

1. **Objective 4**: To provide a secure and reliable system for storing and retrieving encoded messages, offering peace of mind for users communicating sensitive information.

**1.4 Scope of the Project**

The *StegaSafe* project has a wide range of potential applications, particularly in fields where secure communication is critical. Some of the key areas where the project can be applied include:

* **Private Messaging**: *StegaSafe* can be used by individuals who wish to send private, confidential messages without relying on conventional encryption methods.



**Figure 1-1 : Private Messaging using StegaSafe**

* **Corporate Communication**: Organizations can use steganography to securely share sensitive documents or messages within their internal communication systems, ensuring that the data remains protected even if intercepted.



**Figure 1-2: Corporate Communication with StegaSafe**

* **Personal Security**: *StegaSafe* can also serve as a tool for protecting personal information, such as passwords or private notes, by hiding them within images and making them difficult to detect.



**Figure 1-3 : Personal security with StegaSafe**

**1.5 Team Organization**

* **Aaditya Panwar :**

Led the development of the GUI using Kivy and managed the overall project. Implemented the core functionality for encoding and decoding messages, along with integrating the Stegano library.

* **Anusha Nagar :**

Worked on the back-end logic for embedding messages within images, testing the encoding and decoding process for accuracy, and optimizing the system for efficiency.

* **Anushka Patel:**

Assisted with image processing techniques, ensuring that the changes to the image were imperceptible to the human eye. Conducted testing and performance evaluations of the system to ensure robustness and reliability.

**1.6 Report Structure**

This report is structured into five chapters, each detailing the different aspects of the *StegaSafe* project.

**Chapter 1**: Introduction- This chapter introduces the background and motivation for the project, the objectives, scope, and the roles of team members.

**Chapter 2:** Review of Literature- A review of existing research and projects in the field of steganography, highlighting the limitations and challenges of current systems.

**Chapter 3:** Proposed System - Describes the proposed solution, including the architecture, design, and feasibility studies, as well as the benefits of the system.

**Chapter 4:** Implementation - Provides an overview of the technologies and tools used, including Python libraries, the steganographic techniques employed, and the implementation details of the system.

**Chapter 5:** Conclusion - Summarizes the outcomes of the project, discusses the limitations, and suggests directions for future work.

**Chapter 2: Review of Literature**

Review of Literature



In the rapidly evolving field of secure digital communication, steganography has emerged as a crucial technique to hide information within multimedia files, providing a covert layer of security that complements traditional encryption. Unlike encryption, which renders the data unreadable but still visibly altered, steganography masks the presence of data entirely. In particular, image steganography has garnered extensive research attention, focusing on embedding data within images with minimal detection risk. This chapter provides a comprehensive overview of the literature on steganographic techniques, particularly those used in images, and evaluates the limitations, challenges, and recent advancements in the field.

**2.1 Preliminary Investigation**

Steganography, the art and science of concealing messages within non-suspicious media, has been explored across various digital mediums, including audio, video, and images. Image steganography, however, has proven particularly effective due to the complex structures within image files, which make data concealment less conspicuous. Early research in image steganography focused on straightforward methods, such as altering pixel values to store data within the image. Over time, more advanced techniques were developed to address the challenges of detection and robustness, incorporating methods from fields like signal processing, machine learning, and cryptography. Modern applications of image steganography include covert communication, digital watermarking for copyright protection, and secure data transmission in hostile environments.

**2.1.1 Current System**

Current image steganography systems typically utilize the least

significant bit (LSB) method, where data is embedded in the least noticeable parts of pixel data to reduce the likelihood of detection. More advanced methods involve using transformations and wavelet analysis to hide data within complex parts of the image, making extraction more secure. However, many existing systems face limitations, including potential image degradation and susceptibility to statistical steganalysis, which can detect the presence of hidden data.

**2.1.2 Recent Developments in Image Steganography**

As security concerns grow in the digital world, researchers have advanced image steganography beyond traditional LSB methods to include complex algorithms and machine learning approaches. One major advancement has been the integration of transform-domain techniques, where data is embedded in the frequency domain rather than in pixel values, enhancing robustness against image processing operations like compression, rotation, and scaling. Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT) have been widely applied, as they allow for higher security and improved image quality by embedding data in less noticeable parts of the image.

Deep learning-based steganography has recently gained momentum. Neural networks, particularly Convolutional Neural Networks (CNNs), are now used to automatically determine optimal embedding regions within an image, adapting based on image content to minimize detection risk. GANs (Generative Adversarial Networks) are another powerful tool being explored; they can generate images with hidden information that appears statistically indistinguishable from regular images, making detection through standard steganalysis techniques significantly more difficult.

**2.2 Limitations of Current System**

The primary limitations of current steganographic systems include:

* **Detection Vulnerability**: Basic methods like LSB modification are often susceptible to steganalysis techniques, which can reveal the presence of embedded data by detecting statistical anomalies in the image.
* **Image Quality**: Many traditional methods risk noticeable degradation in image quality when data is embedded, which can expose the existence of hidden messages.
* **Data Capacity Constraints**: The amount of data that can be hidden without altering the image noticeably is often limited. Systems that prioritize invisibility may offer minimal capacity, making them unsuitable for larger messages.
* **Accessibility**: Many steganographic systems require technical expertise to operate, creating barriers for general users who may benefit from secure, covert communication methods.

**2.3 Requirement Identification and Analysis for Project**

The literature review on image steganography highlights the importance of systems that balance security, data capacity, imperceptibility, and ease of use. Key requirements identified for the *StegaSafe* project include:

* **Imperceptible Data Embedding**: The project must prioritize embedding data in a way that does not alter the visual quality of the image. By maintaining a high level of image fidelity, *StegaSafe* can ensure that the hidden data remains undetectable under typical inspection.
* **Robustness Against Steganalysis**: The system should employ algorithms that are resistant to statistical detection methods, protecting the security of the hidden message even if the image is subjected to detailed analysis. This robustness is crucial for applications requiring high levels of confidentiality.
* **User-Friendly Interface**: To broaden accessibility, *StegaSafe* aims to incorporate a straightforward GUI that facilitates steganography for both novice and advanced users, allowing them to engage in secure, covert communication without technical complexities.
* **Efficient Data Extraction**: The system should enable easy and accurate extraction of hidden data for authorized users. This feature ensures that embedded messages can be retrieved without data loss or degradation, making *StegaSafe* reliable and user-friendly.
* These requirements aim to address the gaps in current systems and provide an accessible yet secure steganography solution that maintains the integrity of both the hidden data and the cover image.

**2.4 Existing Techniques in Image Steganography**

Several techniques have been developed to improve the quality, security, and capacity of image steganography:

* **Least Significant Bit (LSB) Method**: This method involves altering the least significant bits of pixel values to embed data. While simple and effective for small data, LSB-based methods are highly vulnerable to detection and may result in visible artifacts when too much data is embeddedTransform Domain Techniques\*\*: Methods like Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT) hide data in the transformed coefficients of images, which helps in making the data less detectable and provides robustness against compression .
* **P Differencing (PVD)**: PVD methods adjust pixel values slightly, based on the differences between adjacent pixels. This approach helps to reduce detection risk, as modifications blend naturally into complex regions of the image, making them harder to distinguish from noise .
* **Spread Speanography**: This technique spreads the hidden message across various frequencies within the image, improving the robustness of the steganography against various forms of detection and image manipulation .
* **Deep Learning-Baseds**: Recent advancements have explored the use of neural networks to create adaptive steganographic methods that can dynamically choose embedding positions and parameters to minimize

detection while maximizing capacity.

**2.5 Conclusion**

In this chapter, we reviewed various steganographic techniques and identified the limitations of current systems, specifically in the areas of detectability, data capacity, and usability. Traditional techniques, while effective in simple use cases, often lack the robustness and capacity required for secure, high-quality steganography. Recent advancements, particularly in machine learning, offer promising new avenues for secure and high-capacity embedding, though they introduce computational complexity that limits accessibility for general users.

The *StegaSafe* project aims to address these gaps by developing a steganographic solution that maintains data invisibility, enhances robustness against detection, and provides an intuitive interface. By integrating advanced methods for data embedding and extraction, *StegaSafe* aspires to deliver a secure, user-friendly steganography tool that enables covert communication while preserving the quality and usability of the cover image. Through this approach, *StegaSafe* hopes to advance the field of image steganography, making it accessible and reliable for both technical and non-technical users alike.

.

**Chapter 3: Proposed System**

Proposed System



**3.1 The Proposal**

The proposal aims to develop **StegaSafe**, a secure and user-friendly steganography tool that allows users to hide sensitive messages within digital images, ensuring data confidentiality and integrity. By leveraging the power of Python libraries and the Kivy framework, StegaSafe provides an intuitive interface where users can seamlessly encode and decode messages. This tool addresses the need for secure, covert communication by enabling the safe concealment of messages within images without visibly altering the original file, making it difficult for unauthorized viewers to detect the presence of hidden information.

**StegaSafe** prioritizes the following features:

* **User-Friendly Interface**: StegaSafe provides an intuitive graphical interface, allowing users to select an image file, input their confidential message, and seamlessly initiate the encoding process. The Kivy framework is used to create a visually appealing and accessible GUI, enabling users of varying technical expertise to engage with the tool effortlessly.
* **Secure Encoding Process**: StegaSafe employs robust steganographic algorithms to ensure the hidden message is securely embedded within the image, making detection by unauthorized parties highly challenging. The encoding process minimizes visible changes to the image, thereby preserving its original appearance and enhancing the security of the concealed data.
* **Efficient Message Retrieval**: For authorized users, StegaSafe provides a straightforward decoding process to extract the hidden message accurately. By entering the correct access parameters, users can retrieve the embedded information without affecting the integrity of the image or the message.
* **Maintaining Image Quality**: StegaSafe’s algorithms are designed to optimize image quality, ensuring that the embedded data does not cause any noticeable degradation. This is particularly important for use cases where high-quality visuals are essential, and alterations could lead to suspicion or detection.

In essence, **StegaSafe** is proposed as a comprehensive solution for secure message concealment within images, with applications in both personal data privacy and professional contexts where covert communication is essential. It combines the strengths of Python’s steganographic libraries with a versatile interface in Kivy, offering users a reliable, effective, and secure tool for embedding sensitive information within digital images.

* 1. **Benefits of the Proposed System**

The **StegaSafe** system provides several key advantages over traditional methods of data security and message concealment, enhancing both functionality and accessibility for users:

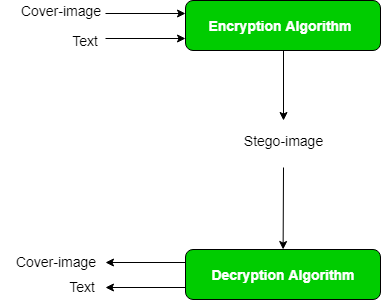
* **Enhanced Security**: StegaSafe leverages image steganography to hide messages within image files, reducing the risk of detection and interception. Unlike conventional encryption methods, which may draw attention due to the appearance of encoded text, StegaSafe’s approach conceals the very existence of a message, making unauthorized access significantly more challenging.
* **Efficiency and Usability**: Designed with a user-centric interface, StegaSafe makes the process of message embedding and retrieval straightforward, even for users without prior steganography experience. The Kivy framework’s clean, accessible GUI helps simplify the encoding and decoding processes, promoting wider adoption and ease of use for all users.
* **Data Integrity and Quality Preservation**: One of StegaSafe’s core features is the preservation of image quality after embedding a

message. The underlying algorithms ensure that data is inserted without causing visible distortion or degradation to the image, thus maintaining the original appearance and integrity of both the image and embedded message.

* **Cross-platform Compatibility**: Built using Python and Kivy, StegaSafe is compatible with multiple operating systems, including Windows, macOS, and Linux. This cross-platform functionality allows users on different systems to benefit from the tool’s features, making it versatile and accessible for a wider audience.
* **Scalability for Future Enhancements**: With its modular design and open-source Python libraries, StegaSafe is adaptable to future improvements. Potential enhancements could include additional encryption layers, support for other media types (e.g., audio and video), or expanded customization options for users.

These advantages position StegaSafe as a robust, accessible, and secure solution for individuals and organizations that require discreet message concealment, providing a blend of security, usability, and quality preservation not typically available in traditional data protection methods.

**3.3 Block Diagram**



**Figure 3-1 : Internal Process**

**3.4 Feasibility Study**

A feasibility study assesses the practicality of implementing StegaSafe by examining the technical, economic, and operational factors involved..

**3.4.1 Technical**

StegaSafe leverages several established Python libraries such as Stegano for steganography functions and the Python Imaging Library (PIL) for image processing. This technical foundation ensures a reliable and robust development environment suited to the needs of image steganography. The use of the Kivy framework allows for the creation of a cross-platform graphical user interface (GUI), enabling the application to function seamlessly across different operating systems (Windows, macOS, and Linux). Additionally, the encoding and decoding processes are optimized for moderate computing power, allowing StegaSafe to operate smoothly on standard hardware configurations without necessitating high-performance GPUs, making it broadly accessible to users.

**3.4.2 Economical**

From an economic perspective, StegaSafe is highly feasible due to its dependence on open-source resources. Python, Kivy, and the libraries used are all open-source, thus eliminating any licensing or subscription costs. This not only minimizes development expenses but also ensures that StegaSafe can be maintained and updated without financial barriers. The application’s low-to-moderate hardware requirements mean that users do not need to invest in specialized or costly equipment, reducing operational costs and making StegaSafe an affordable option for secure communication.

**3.4.3 Operational**

Operationally, StegaSafe is designed to streamline the process of secure data embedding and retrieval, making it feasible for a wide range of users, from beginners to more advanced individuals in the field of data security. The software’s user-friendly interface is intuitive, requiring minimal

training, which allows users to navigate and operate the application with ease. StegaSafe’s design incorporates automated error checks to validate messages and image files before processing, ensuring a smooth, error-free experience. This user-centric approach makes it feasible for everyday use, both for individual users seeking privacy and organizations aiming to protect sensitive data.

**3.5 Deployment Requirements**

To deploy StegaSafe successfully, we need to meet certain hardware, software, and system requirements as follows:

**3.5.1 Hardware**

* **Processor**: 32-bit or 64-bit system, with a basic processing capability (e.g., Intel i3 or equivalent). StegaSafe’s functions do not demand high-performance hardware, so it runs efficiently without the need for a dedicated GPU.
* **Operating System**: Compatibility with Windows, macOS, or Linux, allowing flexibility for users across different environments.
* **Memory and Storage**: Minimum of 4GB RAM and 500MB of available storage for software and image processing tasks.
* **Peripheral Devices**: Standard monitor, keyboard, and pointing device (mouse or touchpad) for user interaction with the application interface.

**3.5.2 Software**

* **Python (Version 3.6 or Higher)**: StegaSafe is developed in Python, leveraging the language’s extensive libraries and resources for image processing and steganography.
* **Kivy**: This library provides the cross-platform GUI functionality, enabling StegaSafe to deliver a consistent and user-friendly interface across different operating systems.
* **Stegano**: Used for core steganographic operations, allowing

efficient and secure message embedding and extraction from images.

* **PIL (Python Imaging Library)**: PIL, or its updated variant Pillow, is essential for handling image files, enabling StegaSafe to support different image formats and provide efficient processing.)

**3.6 Design Representation**

**3.6.1 Use-Case Diagram**

**A diagram of a diagram

Description automatically generated**

**Fig 3-2: Use-Case Diagram**

The Message Encryption and Decryption System use case diagram depicts the process through which a message is securely embedded within an image by one user and later decrypted by another. The diagram involves two primary actors: the User (who encrypts the message) and the Other User (who decrypts the message). The process begins with the User

uploading an image, entering the message they wish to encrypt, and setting a password. After clicking the Encrypt button, the system securely embeds the message within the image. If encryption is successful, the encrypted image is saved; otherwise, an error message is shown. Later, the Other User can attempt to decrypt the message by uploading the encrypted image and entering the password. The system checks if the entered password matches the one initially set; if it does, the message is successfully decrypted and displayed to the Other User. However, if the password is incorrect, an error is shown, preventing unauthorized access to the message.

A diagram of a computer process

Description automatically generated**3.6.2 Activity Diagram**

**Fig 3-3: Activity Diagram**

The **activity diagram** for the **Message Encryption and Decryption System** outlines the process flow for embedding and retrieving a hidden message within an image. It starts with the **User** uploading an image, entering the message, and setting a password, followed by clicking the Encrypt button to initiate encryption. If successful, the encrypted image is saved; otherwise, an error message is displayed. For decryption, the **Other User** uploads the encrypted image and enters the password. The system then checks if the entered password matches the original; if it does, the message is decrypted and displayed. If not, an error ("Wrong Password") is shown, ensuring unauthorized users cannot access the hidden message. This diagram effectively captures the sequence of actions and security checks involved in the encryption and decryption processes.

**3.6.3 Block Diagram**

The block diagram for the Message Encryption and Decryption System presents the main components and their interactions to achieve secure message hiding within an image. The system comprises several key modules: the User Interface, Encryption Module, Image Processor, Storage, Decryption Module, and Password Validator. The process begins at the User Interface, where the Interface to upload the encrypted image and enter a password. The Decryption Module interacts with the Password Validator to confirm if the entered password matches the original. If the password is valid, the Image Processor extracts the message, and the decrypted message is displayed on the User Interface. If the password is incorrect, an error message is shown. This block diagram illustrates how each module works together to ensure secure encryption and controlled decryption of the message in the image.User uploads an image, enters the message, and sets a password. This information flows to the Encryption Module, which instructs the Image

Processor to embed the message into the image. The encrypted image is then saved to Storage.

**A diagram of a process

Description automatically generated**

**Fig 3-4: Block Diagram**

**Chapter 4: Implementation**

Implementation



StegaSafe is designed as a steganography tool that enables secure message embedding within digital images. This system allows users to hide and retrieve messages in images using a user-friendly interface. Our implementation prioritizes security, efficiency, and cross-platform compatibility by leveraging Python libraries like Stegano and PIL for image processing, and Kivy for GUI development.

**4.1 Technique Used**

Our approach to steganography relies on image processing techniques that allow data to be embedded in images in a way that is invisible to the human eye. We use pixel manipulation to encode the message within the image, ensuring both data integrity and privacy.

**4.1.1 Steganography**

Steganography is the practice of concealing messages or information within other non-suspicious data, like images or audio files. In the context of StegaSafe, messages are embedded within images at the pixel level, manipulating the least significant bits (LSBs) of pixel data to store hidden information without altering the image’s visible quality.

By altering the LSBs, the information is hidden in a way that is undetectable to the human eye but easily retrievable by the software. Steganography provides a secure alternative to encryption for cases where users wish to avoid detection of any hidden data.

**4.1.2 Least Significant Bit (LSB) Technique**

The LSB technique is one of the most commonly used methods in steganography. This method involves modifying the least significant bit of each pixel in the image to store hidden data. Given that the human eye is not sensitive to small changes in color, this technique allows for effective message embedding without noticeable changes in the image.

The LSB technique is chosen for its simplicity and compatibility with various image formats, making it ideal for our project.

**4.2 Tools Used**

Our implementation utilizes several key tools and libraries:

**4.2.1 Stegano**

Stegano is a Python library for image-based steganography. It provides built-in functions to hide messages in images and retrieve them as needed. Stegano is well-suited for our project as it simplifies the LSB manipulation process, allowing us to focus on building a user interface and enhancing usability.

**4.2.2 PIL (Python Imaging Library)**

PIL, now part of the Pillow library, is a powerful Python tool for image manipulation. We use PIL for loading, saving, and processing images, which enables StegaSafe to read and write image files while applying transformations needed for steganographic embedding.

**4.2.3 Kivy**

Kivy is an open-source Python library for developing multi-platform applications. It supports input, gestures, and an intuitive GUI framework that is ideal for creating a smooth and user-friendly experience. Kivy enables StegaSafe to function on various platforms, including Windows, macOS, and Linux.

**4.3 Language Used**

Python is chosen as the primary language for StegaSafe due to its readability, extensive libraries, and ease of use for image processing and GUI development.

**Key Characteristics of Python for StegaSafe:**

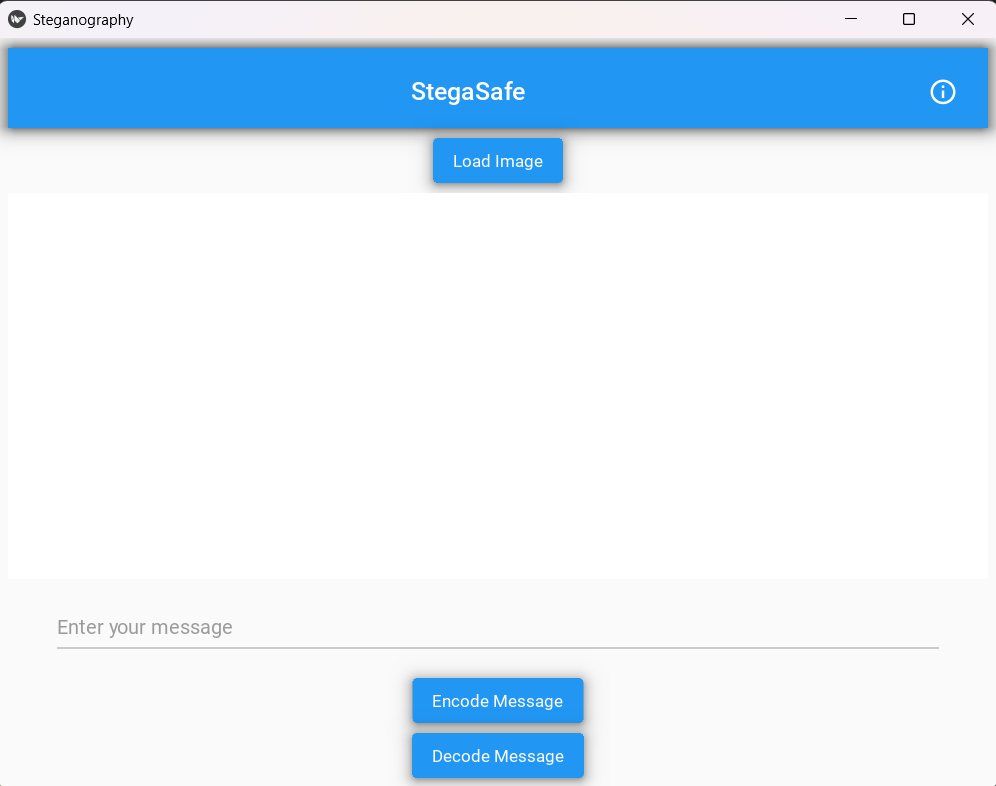
* **Simplicity**: Python’s syntax is simple and readable, making it easy to focus on implementing steganographic techniques without getting bogged down by complex syntax.
* **Open Source**: Python is free to use, and its vast ecosystem of libraries (e.g., Stegano, PIL, Kivy) supports a wide range of functionalities required for StegaSafe.
* **Object-Oriented**: Python’s support for object-oriented programming simplifies the creation of modular, reusable code that is easy to maintain and expand.
* **Extensive Libraries**: The Python Standard Library and third-party libraries provide a wealth of tools that streamline development, including image processing, GUI design, and cryptography.

**4.4 Implementation Process**

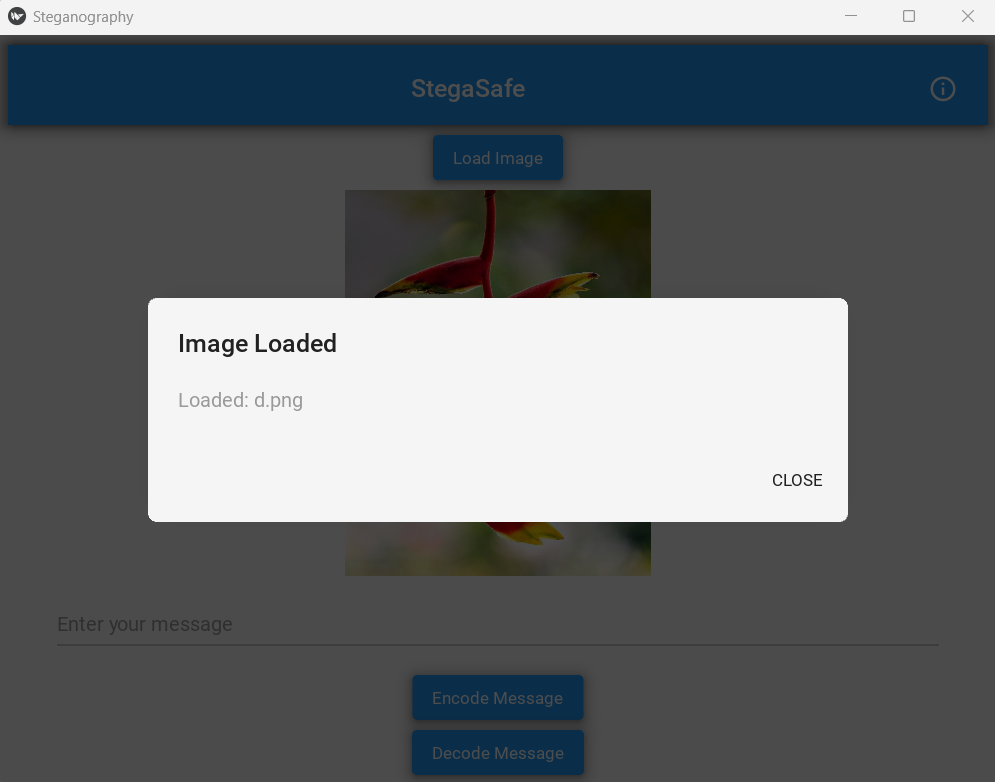
Our development process is structured into the following stages:

1. **User Interface Design**: Using Kivy, we designed an intuitive GUI that guides users through selecting an image, entering a message, and embedding the message into the image.
2. **Encoding and Decoding Functions**: The core steganographic functions are built using Stegano, allowing users to seamlessly hide and retrieve messages.
3. **Error Handling and Validation**: We implemented checks to ensure that only valid images and text inputs are accepted, enhancing user experience and minimizing errors.
4. **Testing and Optimization**: We performed tests to ensure that the image quality remains high after embedding, and optimized the software to run smoothly on a variety of systems.

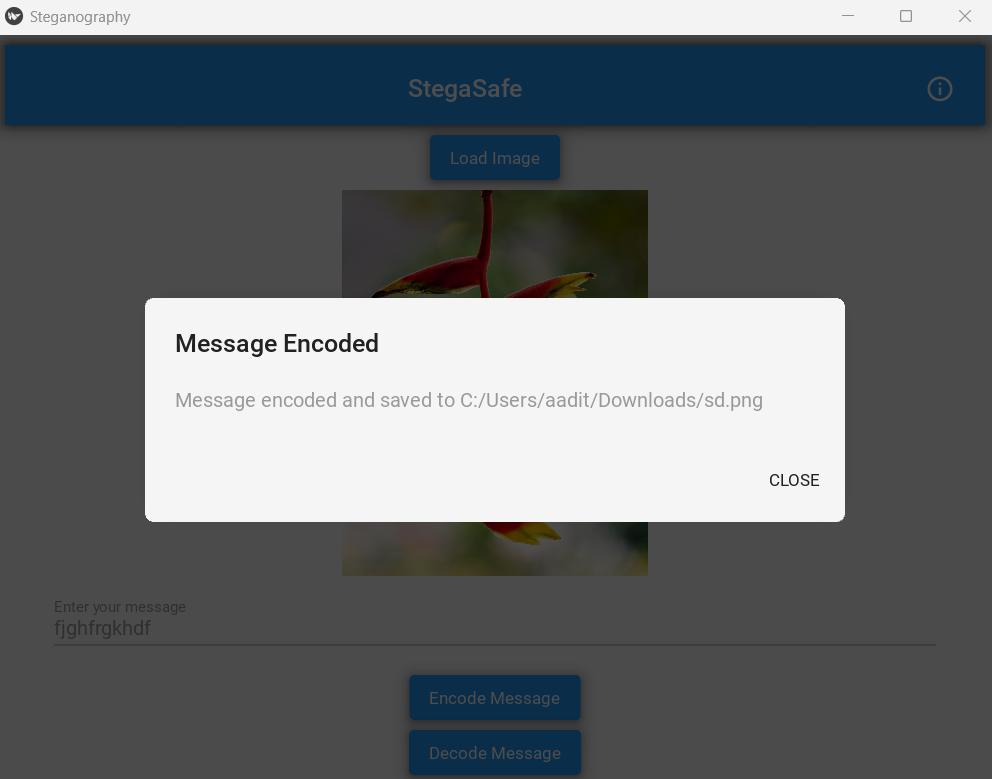
**4.5 Screenshots**



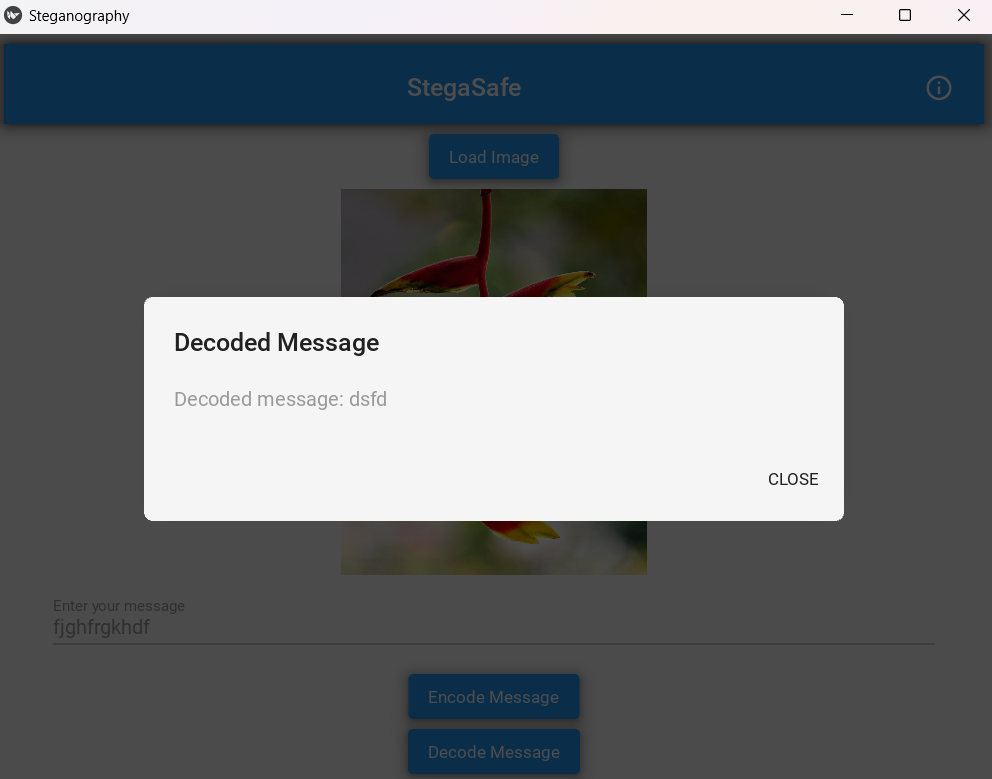
**Figure 4-1: User Interface of StegaSafe**



**Figure 4-2: After loading the image**



**Figure 4-3: After we encoded the message into the image**



**Figure 4-4: When we decode the message from a encoded image**

**4.6 Testing**

Testing ensures the accuracy and security of message encoding and decoding within *StegaSafe*.

**4.6.1 Strategy Used**

Our testing includes:

* **Functionality Testing**: To confirm encoding and decoding accuracy.
* **Security Testing**: Ensuring that hidden data cannot be easily detected by unauthorized users.

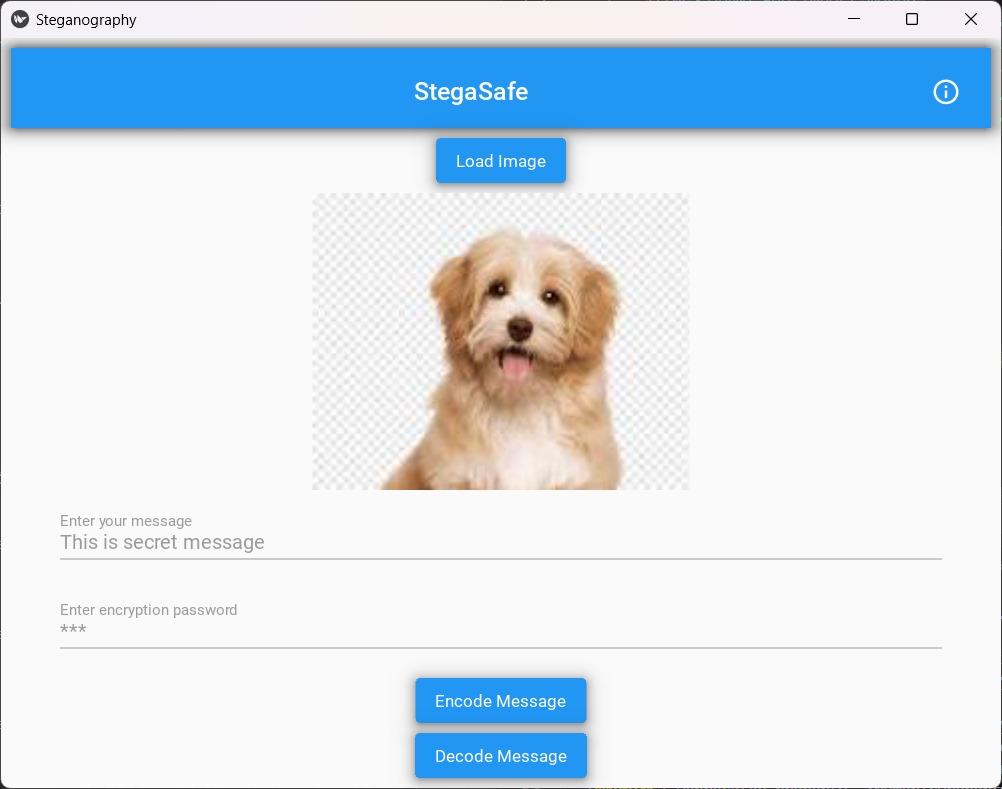
**4.6.2 Test Case and Analysis**

**Test Case ID: TC001 - Encode Message in Image**

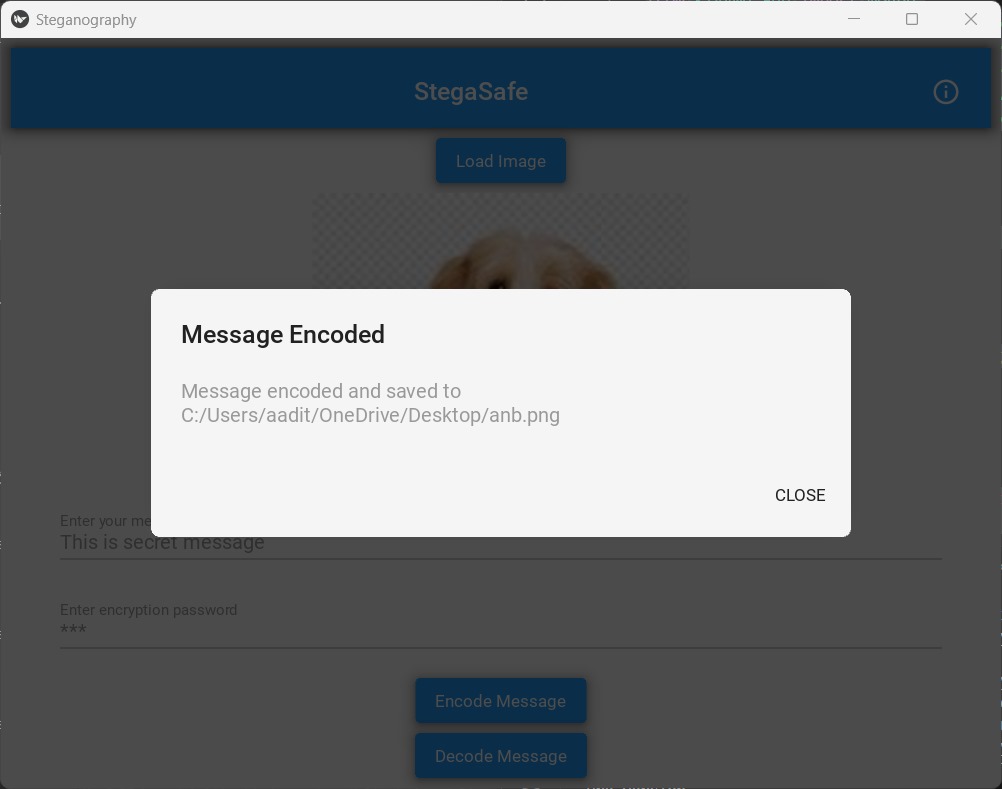
|  |  |
| --- | --- |
| **Test Case ID** | **TC001** |
| Test Case | Encode message in image |
| Steps | 1. Load a valid image.  2. Enter a message in the text field.  3. Click "Encode". |
| Expected Result | Message should be embedded successfully in the image. |
| Actual Result | Message encoded as expected. |
| Status | Pass |

**Table 4.1: Test Case 1**

**Output:**

****

**Figure 4-5: Home screen with the selected image loaded and message entered in the text field**

****

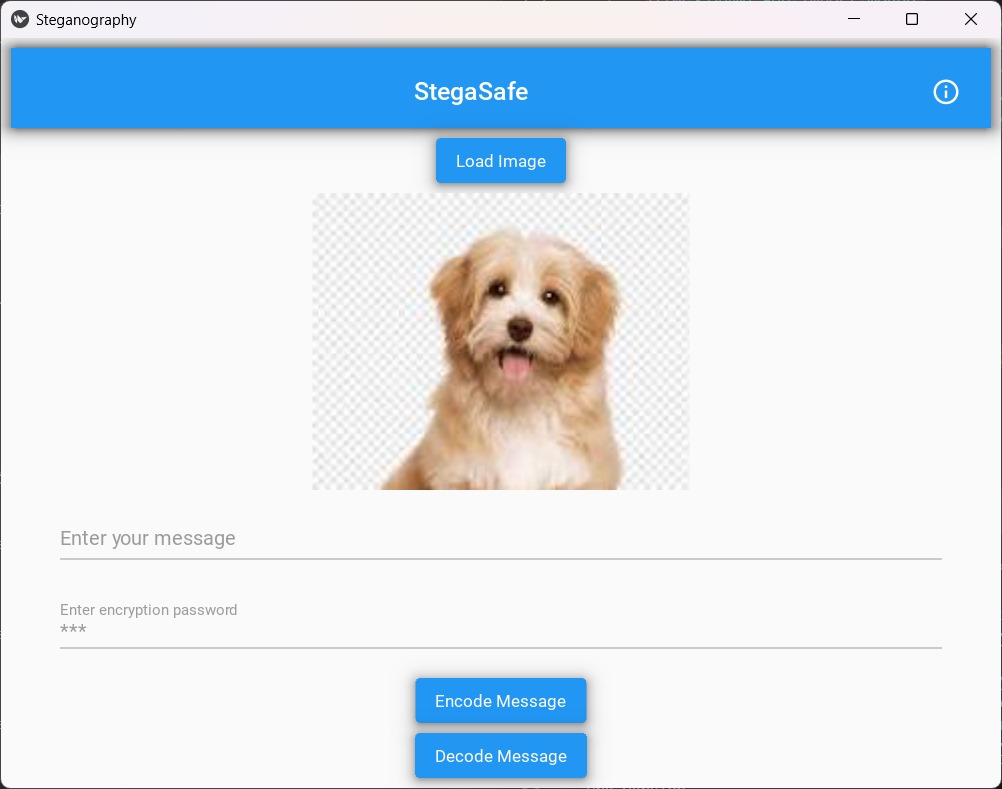
**Figure 4-6** **Confirmation screen**

**Test Case ID: TC002 - Decode Message from Image**

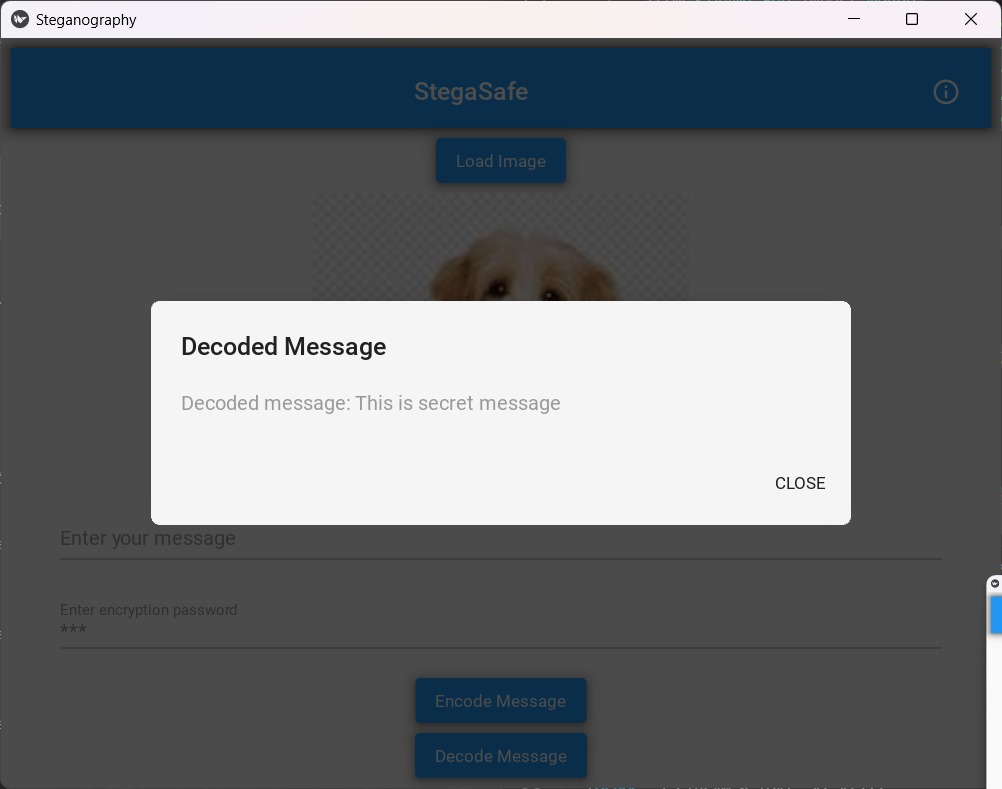
|  |  |
| --- | --- |
| **Test Case ID** | **TC002** |
| Test Case | Decode message from image |
| Steps | 1. Load an image that contains a hidden message.  2. Click "Decode". |
| Expected Result | Hidden message should be accurately retrieved. |
| Actual Result | Message decoded accurately. |
| Status | Pass |

**Table 4.2: Test Case 2**

**Output:**

****

**Figure 4-7** **Home screen with the encoded image loaded**

****

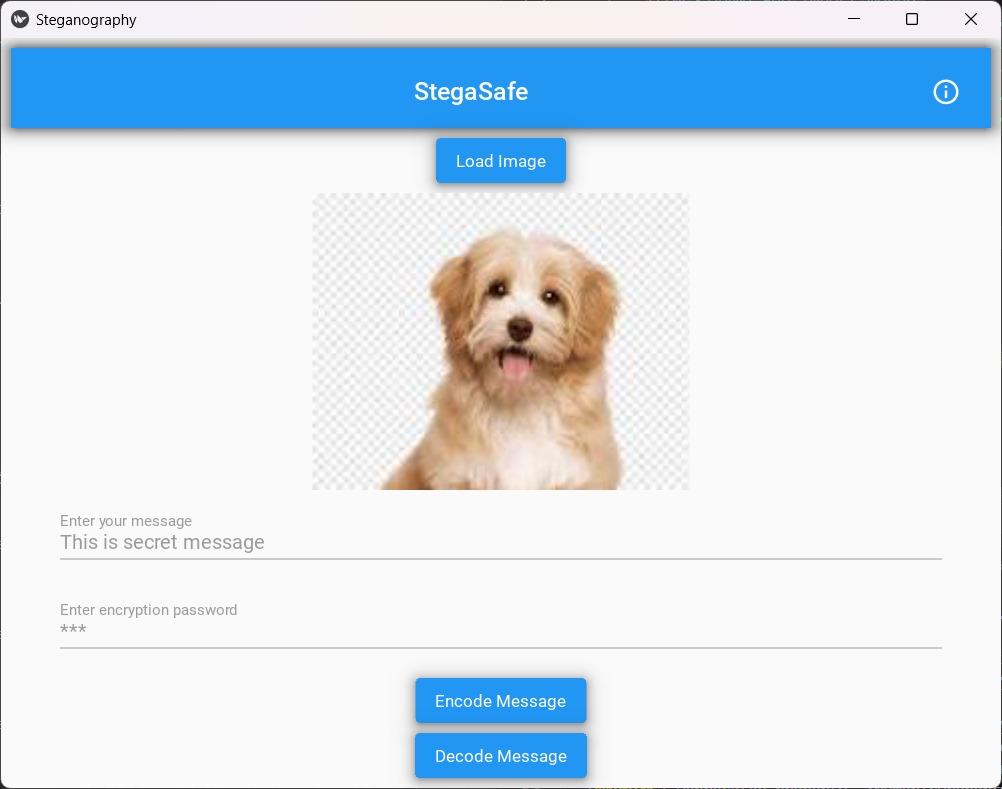
**Figure 4-8** **Decoded message displayed in the text field after clicking "Decode"**

**Test Case ID: TC003 - Save Encoded Image Successfully**

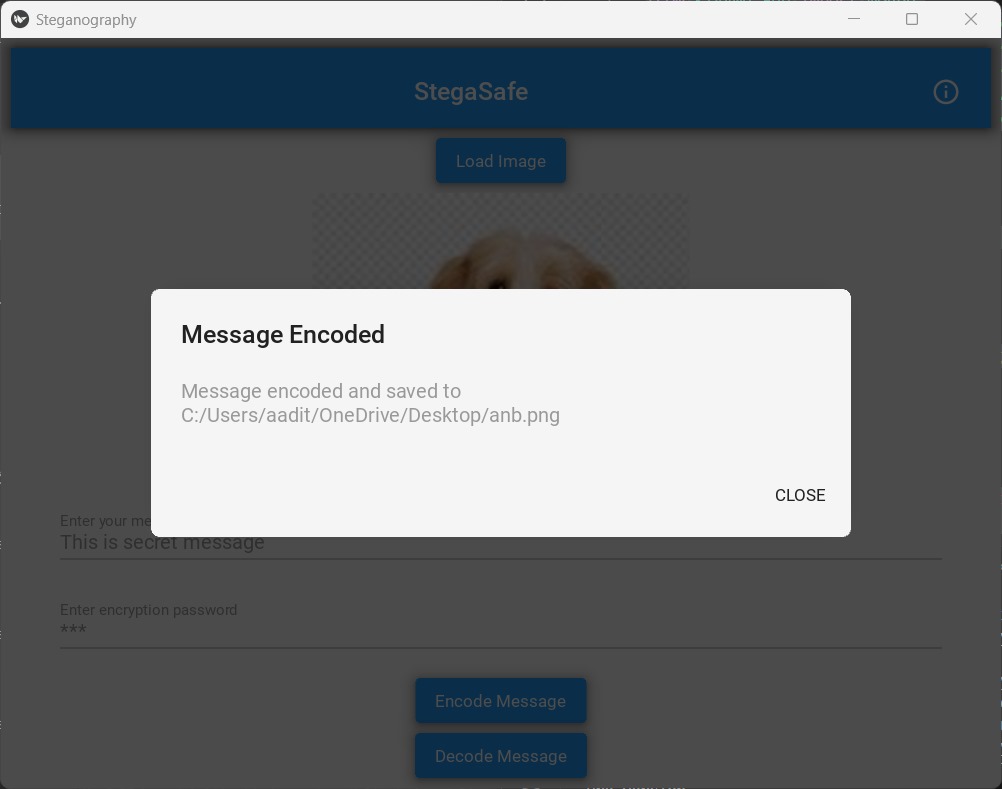
|  |  |
| --- | --- |
| **Test Case ID** | **TC003** |
| Test Case | Save the image after encoding a message |
| Steps | 1. Load an image and encode a message.  2. Attempt to save the encoded image. |
| Expected Result | Encoded image should be saved successfully to the specified location. |
| Actual Result | Encoded image saved as expected. |
| Status | Pass |

**Table 4.3: Test Case 3**

**Output:**

****

**Figure 4-9** **Encoding screen with image loaded and message entered before saving**

****

**Figure 4-10** **Confirmation screen**

**Chapter 5: Conclusion**

Conclusion



**5.1 Conclusion**

The primary objective of *StegaSafe* was to develop an efficient and user-friendly system for securely hiding and retrieving messages within digital images using steganographic techniques. This project successfully achieves this goal by implementing the **Least Significant Bit (LSB)** method, which allows message embedding without creating visible changes in the image, thus maintaining its original appearance. By leveraging **Python** libraries like **Pillow** for image manipulation, **Stegano** for encoding, and **Kivy** for creating an intuitive user interface, *StegaSafe* offers an effective solution for secure, covert communication.

Our project demonstrates the power and versatility of steganography, particularly in scenarios where privacy and discreet message transmission are essential. *StegaSafe* has practical applications in secure data exchange, intellectual property protection, and confidential messaging. With its current capabilities, *StegaSafe* showcases how secure image-based message concealment can be made accessible, accurate, and practical, serving as a foundation for future developments in steganographic methods.

**5.2 Limitations of the Work**

Despite the successful completion of core functionalities, StegaSafe has a few limitations:

* **Limited Data Capacity**: The LSB technique allows for limited data embedding based on the image size and format. Larger messages may require larger images, which could be inconvenient for users or result in noticeable visual distortions if the image is heavily altered.
* **Image Quality Trade-offs**: While the LSB method is designed to be minimally intrusive, some quality degradation may be noticeable in

high-detail images, especially when the message size approaches the image’s capacity. This may limit the practicality of *StegaSafe* for high-resolution images where quality is paramount.

* **Basic Security Measures**: The LSB technique is relatively straightforward, making it potentially vulnerable to detection by advanced steganalysis techniques. This means that *StegaSafe* currently provides a level of security suitable for casual, everyday communication but may not be adequate for high-stakes applications where data security is a top priority.
* **Platform and Performance Constraints**: *StegaSafe* currently operates on desktop platforms with a focus on simplicity and ease of use. However, expanding the application to mobile platforms and optimizing it for larger image files or complex computations remains a challenge.

**5.3 Suggestion and Recommendations for Future Work**

To enhance the effectiveness, security, and versatility of StegaSafe, we recommend exploring the following enhancements in future iterations:

* Advanced Steganographic Techniques: Incorporating advanced embedding methods like Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT) could increase security and data capacity without sacrificing image quality. These techniques are widely used in professional steganography applications and can provide greater resilience against steganalysis.
* Encryption Layer Integration: Adding an encryption layer prior to embedding the message would significantly improve the security of hidden information. This extra step would ensure that, even if the message is detected, it remains unreadable without the proper decryption key.
* Adaptive Embedding Techniques: Implementing adaptive steganography could allow the software to intelligently adjust the embedding technique based on the characteristics of each image, maximizing data capacity and minimizing quality loss. Adaptive techniques analyze aspects such as color distribution and pixel density to identify optimal regions for data hiding.
* Multi-Image Embedding: Another possible enhancement is the ability to split the message across multiple images, allowing larger messages to be embedded while minimizing impact on any single image. This approach would be especially useful for users needing to send more extensive data while maintaining high image quality.
* Cross-Platform Compatibility and Mobile Integration: Extending StegaSafe to mobile platforms would make it more accessible, allowing users to send and retrieve hidden messages directly from their smartphones. This would require additional optimization for performance, considering mobile hardware constraints, but could vastly broaden the project’s applicability.
* User Authentication and Cloud Sync: Integrating user authentication and cloud-based sync options would add an extra layer of data security and convenience. Users could securely back up images with embedded messages, and only authenticated users would have access to these hidden communications, enhancing usability while maintaining security.

These future enhancements could enable StegaSafe to reach a broader audience, offer higher security standards, and support a wider range of use cases. With these upgrades, StegaSafe could serve not only as a powerful tool for secure communication but also as a platform for further innovation in the field of steganography.

**Bibliography**



[1] R. Chandramouli and N. Memon, "Analysis of LSB based image steganography techniques," in *Proceedings of IEEE International Conference on Image Processing (ICIP)*, 2001, pp. 1022-1025.

[2] N. Provos and P. Honeyman, "Hide and seek: An introduction to steganography," *IEEE Security & Privacy*, vol. 1, no. 3, pp. 32-44, 2003.

[3] J. Fridrich, "Steganography in digital media: Principles, algorithms, and applications," Cambridge University Press, 2009.

[4] D. Kahn, "The history of steganography," in *Proceedings of the First International Workshop on Information Hiding*, Springer, 1996, pp. 1-5.

[5] K. Bailey and K. Curran, "An evaluation of image-based steganography methods," *Multimedia Tools and Applications*, vol. 30, no. 1, pp. 55-88, 2006.

[6] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Systems Journal*, vol. 35, no. 3.4, pp. 313-336, 1996.

[7] T. F. Chan and L. Shen, "Image processing and pattern recognition techniques for image-based steganography," *Journal of Computer Science and Technology*, vol. 13, no. 4, pp. 635-650, 1998.

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2017.

[9] P. Agarwal, "Steganographic approach for data hiding using LSB technique," in *Proceedings of International Conference on Computing, Communication & Automation (ICCCA)*, 2015, pp. 227-232.

[10] A. Kaur and A. Gupta, "Improved LSB image steganography using secret key," *International Journal of Computer Applications*, vol. 123, no. 1, pp. 20-25, 2015.

[11] S. M. Raja, C. P. Sumathi, and M. I. Udhayakumar, "Steganography for privacy and security in computer networks," *Journal of Computer Science*, vol. 8, no. 8, pp. 1341-1347, 2012.

[12] E. T. Lin and E. J. Delp, "A review of data hiding in digital images," *Proceedings of the Image Processing, Image Quality, Image Capture Systems Conference (PICS)*, 1999, pp. 274-278.

[13] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, "Digital image steganography: Survey and analysis of current methods," *Signal Processing*, vol. 90, no. 3, pp. 727-752, 2010.

[14] M. Naor and A. Shamir, "Visual cryptography," in *Proceedings of Advances in Cryptology – EUROCRYPT*, Springer, 1994, pp. 1-12.

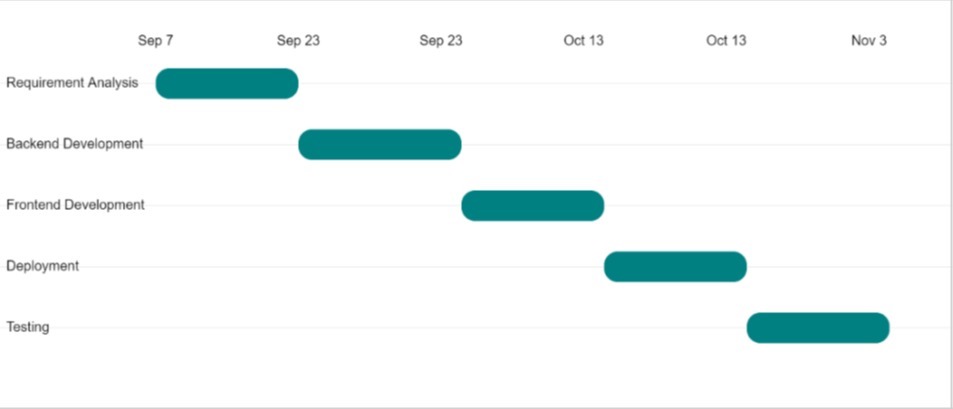
[15] The Python Software Foundation, "Pillow Documentation," Available: <https://pillow.readthedocs.io/>.

[16] Kivy Development Team, "Kivy: Cross-platform Python framework for NUI development," Available: <https://kivy.org/>.

[17] Stegano Developers, "Stegano: Steganography module for Python," Available: <https://stegano.readthedocs.io/>.

**Project Plan**

**Gannt Chart**



**G****uide Interaction Sheet**

|  |  |  |
| --- | --- | --- |
| **Date** | **Discussion** | **Action Plan** |
| 06/09/2024 | Discussed project title selection. | Agreed upon "StegaSafe: Image-Based Message Encryption" as the project title. |
| 13/09/2024 | Discussed the approach for project development. | Selected Python for backend, Kivy for frontend, and Steganography. |
| 20/09/2024 | Presented synopsis and diagrams for review. | Reviewed and discussed  necessary modifications for the synopsis. |
| 1/10/2024 | Presented PowerPoint on the project. | Conducted presentation on StegaSafe:Image Based Message Encryption. |
| 15/10/2024 | Reviewed project implementation progress. | Discussed frontend and backend progress, identified necessary features. |
| 25/10/2024 | Presented and reviewed project report. | Reviewed the project report and outlined required revisions. |
| 05/11/2024 | Reviewed updated project  implementation. | Reviewed frontend, backend, and discussed any remaining  adjustments. |
| 09/11/2024 | Submission of Project synopsis, research paper, and report for review. | Submitted project documents for review and feedback. |



**Appendix A- Source Code**



from kivymd.app import MDApp

from kivymd.uix.boxlayout import MDBoxLayout

from kivymd.uix.button import MDRaisedButton, MDFlatButton

from kivymd.uix.textfield import MDTextField

from kivymd.uix.toolbar import MDTopAppBar as MDToolbar

from kivymd.uix.dialog import MDDialog

from kivy.uix.image import Image

from tkinter import filedialog

from stegano import lsb

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

import base64

import hashlib

import tkinter as tk

# Initialize Tkinter without creating a window

root = tk.Tk()

root.withdraw()

class SteganographyApp(MDApp):

    def build(self):

        self.theme\_cls.primary\_palette = "Blue"

        self.theme\_cls.theme\_style = "Light"

        # Root layout

        main\_layout = MDBoxLayout(orientation='vertical', spacing=10, padding=10)

        # Toolbar

        toolbar = MDToolbar(title="StegaSafe")

        toolbar.right\_action\_items = [["information-outline", lambda x: self.show\_info\_dialog()]]

        main\_layout.add\_widget(toolbar)

        # Load Image Button

        load\_button = MDRaisedButton(text="Load Image", pos\_hint={"center\_x": 0.5}, on\_release=self.load\_image)

        # Image Display

        self.image\_display = Image(size\_hint=(1, 0.4), allow\_stretch=True)

        self.image\_path = False

        # Text Field for Message Input

        self.message\_input = MDTextField(

            hint\_text="Enter your message",

            helper\_text="This message will be encoded into the image",

            helper\_text\_mode="on\_focus",

            size\_hint\_x=0.9,

            pos\_hint={"center\_x": 0.5}

        )

        # Password Field for Encryption

        self.password\_input = MDTextField(

            hint\_text="Enter encryption password",

            helper\_text="Password is required for encoding and decoding",

            helper\_text\_mode="on\_focus",

            size\_hint\_x=0.9,

            pos\_hint={"center\_x": 0.5},

            password=True  # Hide the password text

        )

        # Buttons for Encoding and Decoding

        self.encode\_button = MDRaisedButton(text="Encode Message", pos\_hint={"center\_x": 0.5}, on\_release=self.encode\_message)

        self.decode\_button = MDRaisedButton(text="Decode Message", pos\_hint={"center\_x": 0.5}, on\_release=self.decode\_message)

        # Add widgets to layout

        main\_layout.add\_widget(load\_button)

        main\_layout.add\_widget(self.image\_display)

        main\_layout.add\_widget(self.message\_input)

        main\_layout.add\_widget(self.password\_input)

        main\_layout.add\_widget(self.encode\_button)

        main\_layout.add\_widget(self.decode\_button)

        return main\_layout

    def show\_info\_dialog(self):

        info\_dialog = MDDialog(

            title="About StegaSafe",

            text="This app allows you to securely hide messages within images.",

            buttons=[MDFlatButton(text="CLOSE", on\_release=lambda x: info\_dialog.dismiss())]

        )

        info\_dialog.open()

    def load\_image(self, instance):

        self.image\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.png;\*.jpg")])

        if self.image\_path:

            self.image\_display.source = self.image\_path

            self.image\_display.reload()

            self.show\_dialog("Image Loaded", f"Loaded: {self.image\_path.split('/')[-1]}")

        else:

            self.show\_dialog("No Image Selected", "You did not select an image.")

    def encode\_message(self, instance):

        if not self.image\_path:

            self.show\_dialog("No Image", "Please load an image first.")

            return

        message = self.message\_input.text

        password = self.password\_input.text

        if not message:

            self.show\_dialog("No Message", "Please enter a message to encode.")

            return

        if not password:

            self.show\_dialog("No Password", "Please enter a password for encryption.")

            return

        # Encrypt the message

        encrypted\_message = self.encrypt\_message(message, password)

        # Tkinter file dialog for save location

        save\_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "\*.png")])

        if not save\_path:

            self.show\_dialog("Encoding Canceled", "Encoding canceled.")

            return

        try:

            encoded\_image = lsb.hide(self.image\_path, encrypted\_message)

            encoded\_image.save(save\_path)

            self.show\_dialog("Message Encoded", f"Message encoded and saved to {save\_path}")

        except Exception as e:

            self.show\_dialog("Error", f"Error: {e}")

    def decode\_message(self, instance):

        if not self.image\_path:

            self.show\_dialog("No Image", "Please load an image first.")

            return

        password = self.password\_input.text

        if not password:

            self.show\_dialog("No Password", "Please enter the password used for encryption.")

            return

        try:

            encrypted\_message = lsb.reveal(self.image\_path)

            if encrypted\_message:

                # Decrypt the message

                hidden\_message = self.decrypt\_message(encrypted\_message, password)

                self.show\_dialog("Decoded Message", f"Decoded message: {hidden\_message}")

            else:

                self.show\_dialog("No Hidden Message", "No hidden message found.")

        except Exception as e:

            self.show\_dialog("Error", f"Error: Wrong Password")

    def encrypt\_message(self, message, password):

        key = hashlib.sha256(password.encode()).digest()

        cipher = AES.new(key, AES.MODE\_CBC)

        ct\_bytes = cipher.encrypt(pad(message.encode(), AES.block\_size))

        iv = base64.b64encode(cipher.iv).decode('utf-8')

        ct = base64.b64encode(ct\_bytes).decode('utf-8')

        return iv + ct

    def decrypt\_message(self, encrypted\_message, password):

        key = hashlib.sha256(password.encode()).digest()

        iv = base64.b64decode(encrypted\_message[:24])  # Extract IV

        ct = base64.b64decode(encrypted\_message[24:])

        cipher = AES.new(key, AES.MODE\_CBC, iv)

        message = unpad(cipher.decrypt(ct), AES.block\_size).decode('utf-8')

        return message

    def show\_dialog(self, title, text):

        dialog = MDDialog(

            title=title,

            text=text,

            buttons=[MDFlatButton(text="CLOSE", on\_release=lambda x: dialog.dismiss())]

        )

        dialog.open()

if \_\_name\_\_ == "\_\_main\_\_":

    SteganographyApp().run()

**Appendix B- StegaSafe User Manual**



**1. Introduction**

Welcome to **StegaSafe**, a cutting-edge steganography application designed for secure and private communication. StegaSafe allows users to hide messages within images, ensuring data privacy and confidentiality. This manual will guide you through the features, installation process, and detailed usage instructions.

**1.1 System Requirements**

To run StegaSafe, ensure your system meets the following requirements:

* **Operating System**: Windows, macOS, or Linux
* **Python Version**: Python 3.x
* **Dependencies**: KivyMD, Stegano (installation instructions provided below)

**2. Installation**

To get started with StegaSafe, follow these steps:

1. **Download the StegaSafe package** from the official repository or source provided.
2. **Install the required dependencies** by running:

**pip install kivymd stegano**

1. **Run the main application file** by opening your terminal or command prompt and executing:

**python main.py**

If you encounter any issues, consult the **Troubleshooting** section in this manual.

**3. Features Overview**

**3.1 Image Loading**

Users can load an image to act as a “carrier file” for the hidden message. StegaSafe supports most common image formats, such as JPEG and PNG.

**3.2 Encoding Messages**

Easily hide a message within an image by entering your text and selecting the “Encode” option. The message will be securely embedded in the image, invisible to the naked eye.

**3.3 Decoding Messages**

Retrieve hidden messages from encoded images by loading the image and selecting “Decode.” The original hidden message will appear in the message field.

**4. User Interface Walkthrough**

**4.1 Home Screen**

The StegaSafe Home Screen provides quick access to all features. Key buttons include:

* **Load Image**: To select an image as the carrier.
* **Encode**: To start the process of hiding a message.
* **Decode**: To retrieve a hidden message from an image.

**4.2 Encoding Section**

Enter the message text in the provided field and press “Encode” after loading your chosen image. Once encoded, you can save the newly generated image to your system.

**4.3 Decoding Section**

In this section, upload an encoded image, then press “Decode” to view the concealed message.

**5. How to Encode a Message**

1. From the **Home Screen**, click on **Load Image** to select the carrier image.
2. In the message field, type the text you want to hide.
3. Click **Encode**. Once the process completes, save the encoded image by choosing a file location.

**6. How to Decode a Message**

1. From the **Home Screen**, click **Load Image** to upload the encoded image.
2. Click **Decode** to retrieve and display the hidden message in the message field.

**7. Best Practices and Tips**

* **Use High-Resolution Images**: Higher-resolution images provide better hiding capacity and security.
* **Avoid Image Compression**: Compression can degrade image quality and potentially disrupt the hidden message.

**8. Frequently Asked Questions (FAQ)**

**Q**: What types of files are supported for encoding and decoding? **A**: StegaSafe supports JPEG and PNG formats for image encoding and decoding.

**Q**: Why can’t I decode a message? **A**: Ensure you’re using the original encoded image, as the message may not decode correctly with altered files.

**9. Troubleshooting**

**9.1 Encoding Issues**

If encoding fails, ensure the image file format is supported and that your message size is appropriate for the selected image resolution.

**9.2 Decoding Problems**

If the message does not decode correctly, check that the image hasn’t been modified after encoding.

**10. Contact and Support**

If you need additional support, please contact the StegaSafe development team:

* **Aaditya Panwar** - Team Lead
* **Anusha Nagar** - Developer
* **Anushka Patel** - Developer

**Appendices**

**Glossary**

* **Encoding**: The process of hiding a message within an image.
* **Decoding**: Retrieving a hidden message from an image.
* **Carrier File**: The image used to hide a message.