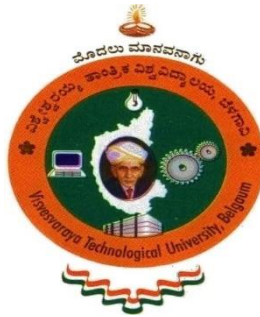


VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JNANA SANGAMA” Belagavi-590018, Karnataka



A PROJECT REPORT ON

“IMAGE STEGANOGRAPHY TOOL”

Submitted in partial fulfilment of the requirements for the award of the Degree of

BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING

Submitted by

RISHAV UPADHYAY
SAURABH SHARMA TIMILSINA
SHREYA KUMARI
SUVAM SHARMA

USN:1BH20CS029
USN:1BH20CS034
USN:1BH20CS040
USN:1BH20CS046

Under the guidance of

Mrs. Shylaja D N

Assistant Professor



BANGALORE TECHNOLOGICAL INSTITUTE

(An ISO 9001:2015 Certified Institute)

Department of Computer Science and Engineering

New Wipro Corporate Office, Off - Sarjapura Road, Kodathi, Bengaluru-560035

2023-2024



BANGALORE TECHNOLOGICAL INSTITUTE
(An ISO 9001:2015 Certified Institute)
Department of Computer Science and Engineering



CERTIFICATE

It is certified that the project work entitled “**IMAGE STEGANOGRAPHY TOOL**” carried out by **RISHAV UPADHYAY (1BH20CS029)**, **SAURABH SHARMA TIMILSINA (1BH20CS034)**, **SHREYA KUMARI (1BH20CS040)**, **SUVAM SHARMA (1BH20CS046)** are the bonafide students of **Bangalore Technological Institute, Bangalore** in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year **2023-2024** It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in the respect of the project work prescribed for the said degree.

Mrs. Shylaja D N

Guide & Asst. Prof.
Department of CSE

Dr. Sreeramareddy G.M.

Professor and HOD
Department of CSE

Dr. H S Nanda

Principal

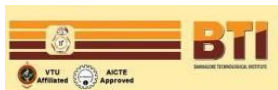
External Viva

Name of the examiners

1. _____

2. _____

Signature with date



BANGALORE TECHNOLOGICAL INSTITUTE
(An ISO 9001:2015 Certified Institute)
Department of Computer Science & Engineering



DECLARATION

We, the students of final year Computer Science and Engineering, Bangalore Technological Institute, Bengaluru, hereby declare that the project entitled “**IMAGE STEGANOGRAPHY TOOL**” has been independently carried out by us under the guidance of **Mrs. Shylaja D N**, Assistant Professor, Department of Computer Science and Engineering, Bangalore Technological Institute, Bengaluru and submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belagavi during the academic year **2023-24**.

We also declare that to the best of our knowledge and belief the work reported here does not form or part of any other dissertation on the basis of which a degree or award was conferred on an early occasion of this by any other.

PLACE:

DATE:

RISHAV UPADHYAY

1BH20CS029

SAURABH SHARMA TIMILSINA

1BH20CS034

SHREYA KUMARI

1BH20CS040

SUVAM SHARMA

1BH20CS046

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals and elders. We would like to take this opportunity to thank them all.

We heartily extend our words of gratitude to our guide, **Mrs. Shylaja D N** for her valuable advice, encouragement and suggestion given to us in the course of our project work. We convey our gratitude to her for having constantly monitored the development of the project and setting up precise deadlines.

We heartily extend our words of gratitude to our Project Coordinator **Dr. Sohan Kumar Gupta** for guiding and giving advices in project implementation.

We heartily extend our words of gratitude to **Mrs. Srujani J** for guiding and giving advices in project implementation.

We would like to express our immense gratitude to the respected Head of Department **Dr. Sreeramareddy G.M**, for his unfailing encouragement and suggestion given to us in course of our work.

We would like to take this opportunity to express our gratitude to the Principal, **Dr. H S Nanda**, for giving us this opportunity to enrich our knowledge.

We are also grateful to the President **Dr. A Prabhakara Reddy** and Secretary, **Sri. C L Gowda** for having provided us with a great infrastructure and well-furnished labs.

Finally, a note of thanks to the Department of Computer Science and Engineering, both teaching and non-teaching staff for their cooperation extended to us.

Last but not the least; we acknowledge the support and feedback of our parents, guardians and friends, for their indispensable help always.

RISHAV UPADHYAY [1BH20CS029]

SAURABH SHARMA TIMILSINA [1BH19CS034]

SHREYA KUMARI [1BH20CS040]

SUVAM SHARMA [1BH20CS046]

ABSTRACT

Steganography, the clandestine art of concealing communication, thrives on embedding information within other data, with digital images emerging as a prime carrier due to their ubiquity on the Internet. With a plethora of steganographic techniques available, each with distinct strengths and weaknesses, the choice of method depends on specific application requirements. Some prioritize the absolute invisibility of hidden data, while others emphasize the capacity to conceal larger messages. This paper provides an overview of image steganography, delving into its methodologies, applications, and the characteristics of effective steganographic algorithms. By examining the suitability of various techniques for different scenarios, it aims to elucidate the diverse landscape of image steganography and guide practitioners in selecting the most appropriate approach for their needs.

CONTENTS

TITLE	PAGE NO.
ACKNOWLEDGEMENT	i
ABSTRACT	ii
CONTENTS	iii
LIST OF FIGURES	vi
LIST OF TABLES	vii
CHAPTERS	
1. INTRODUCTION	1
1.1 Introduction to Project	1
1.2 Aim of the project	3
1.3 Scope of the project	3
2. LITERATURE SURVEY	4
2.1 Literature Survey on proposed system	4
3. SYSTEM ANALYSIS	9
3.1 Problem Statement	9
3.2 Existing System	9
3.3 Proposed System	11
3.4 Feasibility Study	12
3.4.1 Economic Feasibility	12
3.4.2 Technical Feasibility	12
3.4.3 Social Feasibility	12
4. REQUIREMENT SPECIFICATION	14
4.1 Hardware and Software Requirement	14
4.1.1 Software Requirement	14
4.1.2 Hardware Requirement	14
4.2 Resource Requirements	14
4.2.1 Python	14
4.2.1.1 Python Libraries	15
4.2.2 Integrated Development Environments (IDEs)	18
4.2.2.1 Visual Studio Code	18

5. METHODOLOGY	20
5.1 General Architecture	20
5.2 Design Phase	21
5.2.1 Context Diagram	21
5.2.2 Data flow Diagram	21
5.2.3 Use Case Diagram	23
5.2.4 Activity Diagram	25
5.2.5 Interface Structure Design	27
6. ALGORITHM	28
6.1 LSB Algorithm	28
7. MODULES	33
7.1 PIL (Python Imaging Library)	33
7.2 Fernet	34
7.3 Datetime	36
7.4 Base64	36
7.5 Hashlib	38
7.6 OS	40
8. IMPLEMENTATION AND TESTING	43
8.1 Input and Output	43
8.2 Implementation	43
8.2.1 Tools Used	43
8.2.2 Implementation Details of Modules	44
8.3 Testing	49
8.3.1 Unit Testing	51
8.3.2 Integration Testing	53
8.3.3 Acceptance Testing	53
8.3.4 Test Case for System Testing	54
9. WORKING	55
9.1 Efficiency of the proposed system	55
9.2 PSNR and MSE	55
9.3 Sample Code	56
10. RESULTS	65
SNAPSHOTS	69

11. CONCLUSION AND FUTURE WORK	74
11.1 Conclusion	74
11.2 Future Work	74
REFERENCES	76

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
1.1	Steganography method & retrieving of Secret Data	2
5.1	Architecture Diagram of hideEm	20
5.2	Context Diagram	21
5.3	Data Flow Diagram	21
5.4	Use Case Diagram for hideEm	23
5.5	Activity Diagram for Encryption	25
5.6	Activity Diagram for Decryption	26
5.7	Interface Structure Design of hideEm	27
6.1	Image Pixel Distribution in RGB Channel	29
6.2	Impact on changing the MSB and LSB	30
6.3	Pixel before and after insertion of the secret message	31
8.1	Menu Selector of hideEm	45
8.2	Image Selector of hideEm	46
8.3	Encryption Module	47
8.4	Decryption Module	48
8.5	Message Exporter Module	49
10.1	Main Menu	69
10.2	Image Menu	69
10.3	Encryption	70
10.4	Original Image	70
10.5	Message	71
10.6	About Menu	71
10.7	Decryption	72
10.8	Stego-Image	72
10.9	Message Decrypted	73
10.10	Exit Menu	73

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
4.1	Python Libraries & Tools	16
8.1	Test Case for Menu Selector	52
8.2	Test Case for Image Selector	52
8.3	Test Case for Message Decryption	52
8.4	Test Case for Message Exporter	53
8.5	System Testing for hideEm	54
10.1	Comparison of Source and Stego- Image	65

Chapter 1

INTRODUCTION

1.1 Introduction to Project

Project hideEm is created as a tool for secret communication among people where data privacy is denied by the government or any form. It can be very useful in the places where cryptography in internet is not allowed to be used. In such places, if you want to have a secure communication, you can use this tool that will take a message from users, encrypt it first and hide it into the provided image. The tool then prepares a new image with the encrypted message inside. The quality change in image, or the hidden message won't be visible to any naked eye nor will the hidden message will be detected by any tool or technology, used to filter the internet traffic.

In an age where digital communication pervades every aspect of our lives, ensuring the confidentiality and integrity of sensitive information has become paramount. Amidst this backdrop, image steganography emerges as a potent technique for covert communication, allowing individuals to conceal messages within seemingly innocuous images. Steganography, derived from the Greek words "steganos" (covered) and "graphia" (writing), encompasses the art of hiding information within other data, thereby obscuring the very existence of communication itself. The ubiquity of digital images on the internet makes them an ideal carrier for concealed messages, as they attract minimal suspicion during transmission. Image steganography tools leverage a diverse array of techniques to embed secret data within images, ranging from simple LSB (Least Significant Bit) insertion to more sophisticated algorithms that alter pixel values or frequencies. The allure of image steganography lies in its ability to evade detection by unauthorized parties, including automated monitoring systems and human observers. Unlike encryption, which encrypts data into an unreadable format, steganography hides the existence of communication altogether, making it an invaluable tool for scenarios where the mere act of encryption may arouse suspicion. In this context, the development of robust and user-friendly image steganography tools becomes imperative. These tools empower individuals to communicate securely in environments where traditional encryption methods may be restricted or monitored. Moreover, they provide a means for preserving privacy and confidentiality

in a digital landscape fraught with surveillance and censorship. The purpose of this paper is to introduce and explore one such image steganography tool: its functionality, capabilities, and potential applications. Through a comprehensive examination, we aim to elucidate the underlying principles of image steganography, highlight the strengths and limitations of existing techniques, and demonstrate the utility of these tools in real-world scenarios. By shedding light on the intricacies of image steganography and its role in contemporary communication, we hope to equip readers with the knowledge and insights necessary to navigate the evolving landscape of digital privacy and security. Whether for personal use, journalistic endeavors, or activism in restrictive environments, image steganography tools offer a clandestine yet indispensable means of preserving the confidentiality and integrity of sensitive information.

This tool is based on Python and has a text-based interface (TBI), which is super simple to use and to work upon by any people. This tool needs no deep knowledge or understanding of any technology. Users simply need to provide an image, message, and password than the tool does the task of encrypting and hiding the encrypted text in the image.

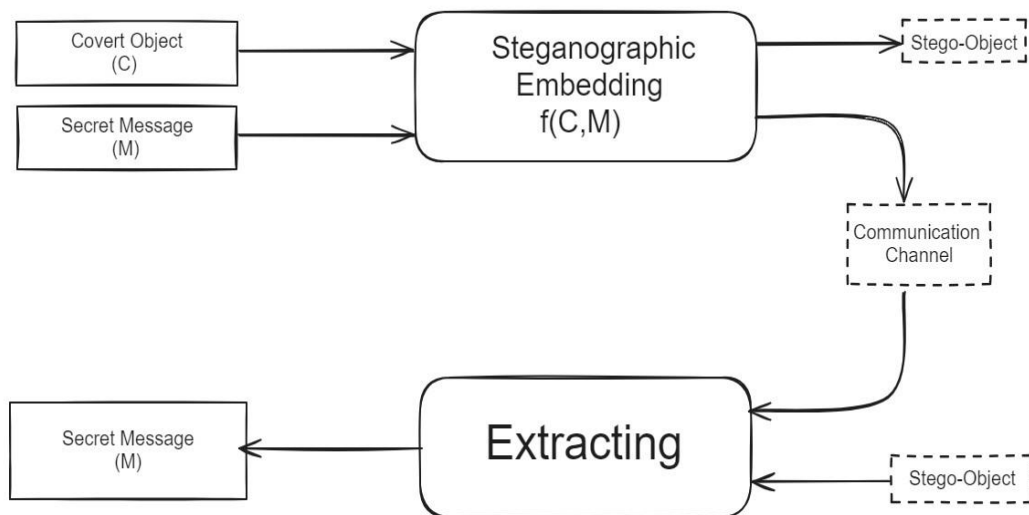


Figure 1.1: Steganography method and retrieving of Secret Data

1.2 Aim of the Project

The aim of the project is to develop a user-friendly and robust software solution that enables individuals to securely conceal messages within digital images. HideEm provides users to communicate covertly in environments where traditional encryption methods may be restricted or monitored. By leveraging steganographic techniques, the project seeks to provide a means for preserving privacy and confidentiality in digital communication channels. HideEm aims to enhance awareness and understanding of image steganography principles and applications, facilitating informed decision-making regarding privacy and security practices in the digital age. Ultimately, the project aims to contribute to the advancement of tools and methodologies for safeguarding sensitive information in an increasingly interconnected world.

1.3 Scope of the Project

The scope of our project is to encompass the development of a user-friendly tool that integrates steganographic algorithms and encryption techniques to enable secure communication by concealing messages within digital images. Key aspects include researching and implementing steganography algorithm such as LSB insertion, designing an intuitive user interface for easy message embedding, integrating encryption mechanisms for enhanced security, preserving image quality to avoid detection, ensuring compatibility with common image formats, conducting thorough security analysis, providing comprehensive documentation and support, and addressing ethical considerations surrounding the responsible use of steganography tools.

Chapter 2

LITERATURE SURVEY

2.1 Literature survey on proposed system

[1]. A Novel Steganography Technique for Digital Images Using the Least Significant Bit Substitution Method (Year: 24 Nov 2022).

This paper was proposed by Shahid Rahman, Student Member, IEEE, Jamal Uddin, Student Member, IEEE, Habib Ullah Khan, Hameed Hussain, Ayaz Ali Khan, and Muhammad Zakarya. This paper proposes a novel technique in steganography focusing on digital images like RGB, grayscale, texture, and aerial images. The approach utilizes the Least Significant Bit (LSB) substitution method using B-Channel (Blue Channel) and a novel algorithm based on value difference to achieve higher security, imperceptibility, capacity, and robustness compared to existing methods. The experimental results demonstrate the effectiveness of the proposed approach, showing improvements in various metrics such as PSNR (Peak Signal to Noise Ratio) correlation score, PSNR with variable measures of code insertion, and embedding different sizes of secret messages in different types of images. Through numerical simulations, the proposed strategy outperformed the next-best current methodology by 5.561 percent in terms of PSNR Correlation score. Additionally, the proposed approach achieved a 6.43 percent better score in PSNR with a variable measure of code inserted in similar images with distinct dimensions. Furthermore, encrypting the same amount of information in images of varying sizes resulted in approximately 6.77 percent improvements. Embedding different sizes of a particular secret message in a different image (such as Gray, Texture, Aerial, and RGB images) came out with about 5.466 percent better score. Overall, the findings suggest that the approach outperforms existing methodologies in terms of hiding capacity and robustness. This work uses the MLF (Maximum Likelihood Functional) technique to generate the DK (Digital Key). It uses GLE (Grey Level Encryption) for generating Stego-image. However, the continuous looping between the channels makes it fine-consuming. It may have high security due to encryption but is prone to tempering, and scaling attacks.

[2]. Carrier Image Rearrangement to Enhance the Security Level of LSB Method of Data Steganography (Year: Jan 2022).

This paper was proposed by Prof. Mohamad K. Abu Zalata, Dr. Mohamad T. Barakat, and

Prof. Ziad A. Alqadi. A method to enhance the security level of the Least Significant Bit (LSB) embedding technique has been proposed. This enhancement involves incorporating secret message protection by utilizing a complex Private Key (PK) to rearrange the carrier image and generate the Stego-image. The PK is created by the sender and kept confidential. This PK is crucial for both embedding and extracting the hidden message. It can be updated or modified based on factors such as block size and the specific carrier image chosen for embedding. By customizing the PK based on block division and rearrangement, the proposed method enhances the security of LSB steganography while providing flexibility and adaptability to meet the sender's preferences and requirements. Importantly, adding the protection stage to LSB does not compromise quality parameters such as Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR). The additional time required for rearrangement is minimal and does not adversely impact the efficiency of the LSB embedding process, ensuring that the proposed method remains efficient. Based on the experimental findings, it is recommended to utilize carrier images with larger sizes. This choice optimizes the quality parameters, specifically MSE and PSNR values, thereby enhancing the overall effectiveness of the steganographic process.

[3]. A Novel RGB Image Steganography Using Simulated Annealing and LCG via LSB (Year:20 Jan 2022).

This paper was proposed by Mohammed J. Bawanwh, Emad Fawzi Al-Shalabi, Obaida M. Al-Hazaimh. This paper introduces a novel and resilient framework for color image steganography, amalgamating Linear Congruential Generator (LCG), simulated annealing (SA), Caesar cryptography, and LSB substitution method into a unified system. The aim is to mitigate the risks posed by steganalysis and securely transmit data to its intended destination. The framework utilizes Simulated Annealing (SA), with the assistance of a Linear Congruential Generator (LCG), to identify the optimal minimum path for concealing information within a cover color image (RGB). Subsequently, the confidential message undergoes encryption and is embedded within the RGB image path, serving as the host medium, through a combination of Caesar and LSB procedures. Both the embedding and extraction processes of the secret message necessitate a shared understanding between the sender and receiver, encompassing SA initialization parameters, LCG seed, Caesar key agreement, and secret message length. Without the correct knowledge of the manipulation process, intruders conducting steganalysis will be

unable to detect or decipher the secret message concealed within the host image. Simulated Annealing (SA), inspired by the heating and cooling of metals, serves as an optimization search method to tackle linear and non-linear problems. By mimicking the metal annealing process, SA begins with an initial worst-case scenario and gradually progresses toward the optimal solution while controlling temperature and fitness function values. It employs a random search process to maximize or minimize the solution, rejecting modifications that worsen the current solution. In this system, LSB is utilized to conceal digital messages within a digital medium by substituting the rightmost bit of the cover medium with a bit from the message. Simulated Annealing (SA) operates within the RGB color image, functioning as a searcher for an optimal solution vector to hide data within a set of random vectors generated via LCG. Each color channel (Red, Green, and Blue) of the host image contributes to constructing three optimal vectors, with each vector dedicated to concealing a specific part of the secret message. The secret message, treated as a binary file, is divided into three parts of bytes, each concealed within a color channel. Before embedding, the message bytes undergo encryption via the Caesar algorithm. LSB substitution is then applied to embed the bits of each secret message part into the computed vectors. LCG, a commonly used random generator, ensures pixel duplication prevention in the constructed SA vector by meeting prerequisite conditions before generating random data. During embedding and extraction processes, users input a set of keys for LCG (initial seed) and the Caesar algorithm (cryptography key). SA parameters, functioning as keys, remain fixed at sender and receiver sites to minimize key transfers, though they can be adjusted based on mutual agreement between parties. The combined secret message length and type are embedded within the host image, allowing authorized extractors to retrieve the data by employing the correct extraction knowledge. However, the image has a bit more noise as computed in terms of Mean Square Error (MSE). Due to the Simulated Annealing (SA) minimum path, some images even lacked similarities in visual appearance. The Peak Signal to Noise Ratio (PSNR) is also low with an average of 73.

[4]. Implementation and Comparative Analysis of Variet of LSB Steganographic Method (Year:25 Jan 2022).

This paper was proposed Temitayo Ejidokun, Olusegun O. Omitola, Kehinde Adeniji, Ifeoma Nnamah. This paper introduces the k-LSB-based method which hides an image within another by utilizing k least significant bits, with a local entropy filter aiding in identifying the concealed image bits while minimizing distortion and information loss. An enhanced LSB substitution technique adds a layer of security by encrypting the message before embedding, resulting in increased embedding capacity and good PSNR. Additionally, an automated dual-level security method encrypts secret messages using Character Bit Shuffler (CBS) before insertion into an image via LSB, prioritizing simplicity and image quality preservation. Another approach introduces an improved LSB steganography method employing the modulus function for data hiding within RGB true-color images, bolstering transmission capacity by reducing secret data streams from seven to five bits. These methods collectively represent advancements in steganographic techniques, offering heightened security, increased capacity, and improved image fidelity. This study conducted a comparative analysis of the LSB substitution method across three different image formats. The evaluation revealed a consistent trend: as the number of stego-bits increased, there was a corresponding decrease in PSNR alongside an increase in MSE, ultimately leading to image distortion. It was noted that the security of the embedded message is closely linked to the perceptibility of the image. Given that steganographic methods are not foolproof, caution is advised when concealing sensitive information. Based on the findings, it is recommended that the cover image should be sufficiently large to accommodate the secret data, ideally with the data occupying no more than 20% of the cover image on average. Additionally, it is advisable to minimize the amount of data to be hidden to avoid excessive pixel usage, which could compromise image quality.

[5]. A Visual Cryptography Based Data Hiding Technique for Secret Data Encryption and Decryption (Year: Dec 2021).

This paper was proposed by Sonal Karade, and Prof. Savita Chouhan. This paper introduces visual cryptography that is used to hide secret data within images, where an image is divided into multiple shares without the need for computation. These shares can then be superimposed to reveal the hidden key image or text through human perception. Initially, the model involved a page of cipher text and a transparency page (secret key),

where the original text could be obtained by superimposing the transparency over the cipher text. Later advancements introduced the (k,n) secret sharing scheme, where secret shares are distributed among participants, and the secret is revealed only when a sufficient number of shares are combined. Visual cryptography has since been extended to color images, with techniques developed based on color decomposition and the use of RGB/CMY images. Traditional approaches like Cryptography, Steganography, and Data Hiding can also be utilized in this field. Cryptography focuses on mathematical techniques for data security, while steganography hides messages within seemingly normal mediums, such as images, without arousing suspicion. Data hiding conceals the existence of secret information, while cryptography protects the content of messages. Reversible data hiding in encrypted images has gained attention, where data is embedded in the host media and can be recovered without loss. This hidden data can include authentication data or author information related to the image. Reversible data hiding ensures that both the secret data and the host media can be recovered at a lossless level. It produces a visual cryptographic image in the form of a master and slave. The Stego image is generated by the random generation method. The preprocessing tasks like image resizing and gray scale changing can be performed on Stego-image. It requires MATLAB R2012b software for simulation and also a high-processing system. The performance of the proposed algorithm is tested for different grey-scale images. The MATLAB-based simulation result shows good PSNR value for the Stego-image and better quality of the Stego-image. For calculation, the performance of the proposed method uses Peak Signal to Noise Ratio (PSNR) and Mean Square Error (MSE). The proposed method shows good results under the different attacks. The visual cryptography is a better method for data hiding as compared to steganography and cryptography. Visual cryptography-based data hiding methods provide a double layer of security and a better authentication process as compared to other methods.

Chapter 3

SYSTEM ANALYSIS

3.1 Problem Statement

The strict prohibition of Secure Socket Layer (SSL) usage in countries like Iran, Cuba, and Sudan poses significant risks to online communication security. Without SSL encryption, internet traffic remains vulnerable to attacks such as Network Sniffing and Man-in-the-Middle attacks. This leaves personal data exposed and susceptible to interception by government entities or malicious actors with sufficient resources. In such environments, sharing sensitive information like organizational server passwords or credentials carries inherent risks of exposure to hackers. However, utilizing image steganography tools provides a solution to this dilemma. By hiding secret messages within images, sensitive information remains encrypted and protected from interception. Even if third parties intercept the traffic, they will only see the image and will be unable to extract the hidden messages without access to the steganography tool. This approach helps mitigate the risks associated with unencrypted internet traffic in environments where SSL usage is restricted.

3.2 Existing System

- **SteganPEG** is a lightweight software application built specifically to help users hide sensitive data inside JPG images in only a few steps. It sports a clean and simple interface that offers only a few configuration settings to tinker with. The program gives you the possibility to embed photos into the working environment using the built-in browse function, so you cannot rely on “drag and drop” operations. SteganPEG allows you to encrypt data by setting up passwords, previewing pictures in the primary panel, as well as inserting multiple items to be hidden in the photos. What’s more, the utility can indicate how much space is occupied by the hidden files so you can calculate if there’s available space for adding more items. The photo, which embeds the sensitive data, can be exported to JPG file format, provided that you have specified the filename and saving directory. During our testing, we have noticed that SteganPEG accomplishes a task very quickly and without errors

throughout the entire process. As would be expected from such a small app, it remains light on the system resources, so it doesn't burden computer performance, nor interfere with other programs' functionality. To sum it up, SteganPEG seems to be the right choice in case you are looking for a simple-to-use encryption application that comes packed with only a few dedicated parameters, and is suitable especially for rookies.

- **Hide'N'Send** is a small utility that offers steganography. It lets you hide any kind of file behind a JPG image file. It supports hashing and encryption too. This means that you can hide your data by encrypting it. This adds an extra layer of security. The interface of the tool is simple and offers two tabs — one to hide data and the other to extract data. You can select the options accordingly. Just run the tool, select the image file, then select the file that you want to hide, select the encryption type, and then hide the data in the image. Use the same tool again to extract the hidden information in the image.
- **Camouflage** is another steganography tool that lets you hide any type of file inside of a file. There is no kind of restriction in the software for hiding the file. Use of the tool is simple and easy: you can just right-click on any file and select the Camouflage option. To extract your sensitive data from the file, right-click and select uncamouflaged. You can also set a password to encrypt the hidden data inside the file. The project is no longer in development, but you can use the old file for your work. It still performs well and you can use it to hide your confidential data inside an image.
- **OpenStego** can hide any data within an image file. It also does watermark image files with an individual signature. It can be used to detect unauthorized file copying. This tool has standard licensing on watermark technology and images with messages may be distinguished by any people.
- **RSteg** is yet another Steganography tool developed using Java. You need to have Java installed on your machine to run RSteg. Another striking advantage is its portable feature. Hence no need to install it, just run and the software windows pop up.

- **Our Secret** is yet another Image Steganography Tool that allows hiding files, text, and messages in photos. If these files are combined with images, then the final image will be of more size. However, this tool makes sure that the size of the final image is not abnormal.

3.3 Proposed System

- This proposed system is to create an Image Steganography Tool that allows users to embed the secret information within an image and maintain the integrity of the secret data with the Advanced Encryption Standard (AES) using a Pre-shared Key (PSK).
- With a focus on user-friendliness, the system will feature an intuitive interface and robust encryption mechanisms to ensure the confidentiality of embedded information.
- Users will be able to easily embed photos into images using a built-in browse function, preview the modifications, and export the modified images with specified filenames and saving directories.
- The tool will also display the occupied space by the hidden files, allowing users to optimize data embedding.
- Performance and efficiency will be prioritized to ensure quick execution of tasks without compromising system resources.
- Here, in the system, the LSB algorithm is slightly modified to gain the efficiency of the flow of the program for better time complexity. The program only utilizes the red channel of the RGB color and leaves others unchanged.
- The average Peak Signal to Noise Ratio (PSNR) of our tool generates the result of 101.454.

3.4 Feasibility Study

The feasibility study for the Image Steganography Tool aims to assess the practicality and viability of developing such a tool. This study evaluates various aspects, including technical, economic, operational, and scheduling considerations, to determine the project's feasibility. It consists of three parts:

3.4.1 Economic Feasibility

The economic feasibility of our project primarily revolves around the development costs involved. Fortunately, our program does not rely on proprietary software or hardware, which could otherwise inflate production expenses. Additionally, we plan to offer the product as a free tool to all users, eliminating any barriers to access. By leveraging open-source technology, we can implement the necessary features without incurring any costs. Looking ahead, we have the potential to integrate premium features into the program, which could provide additional revenue streams and support further development efforts, including the integration of proprietary technology. Consequently, our project is deemed economically feasible, offering a cost-effective solution to users while allowing for future growth and expansion opportunities.

3.4.2 Technical Feasibility

The system under development is built on Python and its modules, a technology widely supported by computing systems worldwide. The basic hardware requirements, such as ARM Cortex or Pentium 4 processors, 500 MB of RAM, and 5 GB of storage, are common place in today's computing landscape. As such, the technology used to develop the program is highly adaptable and applicable to current computing environments. This ensures compatibility across various systems and makes the program accessible to a wide range of users, regardless of their hardware specifications.

3.4.3 Social Feasibility

Social feasibility of the Image Steganography Tool involves assessing its acceptability and impact within society. This encompasses considerations such as ethical implications, user perceptions, and societal attitudes toward privacy and security. While steganography inherently raises concerns about the potential misuse of illicit activities,

such as covert communication for malicious purposes, its legitimate applications for safeguarding privacy and enabling secure communication are widely recognized. Therefore, social feasibility hinges on educating users about responsible usage, fostering awareness of privacy concerns and promoting ethical guidelines for employing steganography tools. By emphasizing the tool's role in preserving confidentiality and protecting sensitive information in an era of pervasive surveillance, its social feasibility can be enhanced, fostering greater acceptance and adoption within society.

Chapter 4

REQUIREMENT SPECIFICATION

4.1 Hardware and Software Requirement

4.1.1 Software Requirement

- Operating system (Example: Linux, Ubuntu)
- Code Editor (Example: VS code, Jupyter Notebook)
- Python and its Libraries
- Bash Shell

4.1.2 Hardware Requirement

- Processor: Any modern processor (ARM Cortex/Intel Pentium 4) or above
- Memory: 500 MB or above
- RAM: 1 GB or above
- Storage: 5 GB or above

4.2 Resource Requirements

4.2.1 Python

Python is a general-purpose language, which means it's designed to be used in a range of applications, including data science, software and web development, automation, and generally getting stuff done. Let's take a closer look at what Python is, what it can do, and how you can start learning it.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented, and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last

release of Python 2. Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member Steering Council to lead the project.

In 2022, Python 3.10.4 and 3.9.12 were expedited, and 3.8.13, and 3.7.13, because of many security issues. When Python 3.9.13 was released in May 2022, it was announced that the 3.9 series (joining the older series 3.8 and 3.7) would only receive security fixes in the future. On September 7, 2022, four new releases were made due to a potential denial-of-service attack: 3.10.7, 3.9.14, 3.8.14, and 3.7.14.

In 2024, Python 3.11.8 and Python 3.12.2 was released on February 6. Python 3.8.19, Python 3.9.19, and Python 3.10.14 were updated on March 19, 2024. Python version 3.11.9 was released on April 2, 2024. As of April 9, 2024, the latest version of Python 3 is Python 3.12.3. This recent release likely includes various improvements, bug fixes, and potentially new features to enhance the functionality and performance of the language. Users are encouraged to update to the latest version to take advantage of these advancements and ensure compatibility with the latest developments in the Python ecosystem. Python 3.13 will be released on October 1, 2024 and currently, the maintenance status is pre-release.

4.2.1.1 Python Libraries

Normally, a library is a collection of books or a room or place where many books are stored to be used later. Similarly, in the programming world, a library is a collection of precompiled codes that can be used later on in a program for some specific well-defined operations. Other than pre-compiled codes, a library may contain documentation, configuration data, message templates, classes, values, etc.

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and more convenient for the programmer. As we don't need to write the same code again and again for different programs. Python libraries play a very vital role in the fields of Machine Learning, Data Science, Data Visualization, etc.

As stated above, a Python library is simply a collection of codes or modules of codes that we can use in a program for specific operations. We use libraries so that we don't need to write the code again in our program that is already available. But how it works? Actually, in the MS Windows environment, the library files have a DLL extension(Dynamic Load Libraries). When we link a library with our program and run that program, the linker automatically searches for that library. It extracts the functionalities of that library and interprets the program accordingly. That's how we use the methods of a library in our program. We will see further, how we bring in the libraries in our Python programs.

Table 4.1: Python Libraries & Tools

Python Libraries & tools	
Name	Function/s
1. Fernet	It is an open-source library used for generation of the key, encryption of the plaintext into ciphertext, and decryption of the ciphertext into plaintext using encrypt and decrypt methods respectively. Fernet guarantees that data encrypted using it cannot be further manipulated or read without the key.
2. PIL (Python Imaging Library)	This library provides extensive file format support, an efficient internal representation, and fairly powerful Image Processing capabilities to the python interpreter. PIL is used for opening, manipulating and saving many different image file formats.

3. base64	The base64 library offers functions for encoding and decoding binary data to and from base64 strings, effectively converting any binary data to plain text. A common encoding method called base64 converts binary data into a string of printable ASCII characters.
4. hashlib	Python's hashlib provides a variety of hash functions, including md5, sha1, and sha256. These functions are used to create hash objects, which are then used to generate hashes of data.
5. datetime	The datetime library supplies classes for manipulating dates and times. While date and time arithmetic are supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation.
6. os	The OS library in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The "os" and "os.path" modules include many functions to interact with the file system.

4.2.2 Integrated Development Environments (IDEs)

An integrated development environment (IDE) is a software application that provides a comprehensive set of tools for software development. IDEs typically include features such as code editing, debugging, testing, and version control integration, as well as tools for building and deploying software.

Some common features of IDEs include:

- **Code editor:** A text editor with advanced features such as syntax highlighting, code completion, and refactoring tools.
- **Debugger:** A tool that allows developers to inspect the state of a program while it is running and find and fix errors.
- **Testing tools:** Features for creating and running automated tests for a software project.
- **Version control integration:** Support for working with version control systems such as Git, allowing developers to track changes to their code and collaborate with others.
- **Build and deployment tools:** Features for building and deploying software, such as support for compiling code and creating packages.

There are many IDEs available for different programming languages, each with its own set of features and target audience. Some IDEs are general-purpose, while others are specialized for a particular language or type of development. IDEs are commonly used by software developers to write, test, and debug code more efficiently.

4.2.2.1 Visual Studio Code

Visual Studio Code (VS Code) is a popular, open-source code editor developed by Microsoft. It is available for Windows, macOS, and Linux, and it supports a wide range of programming languages and platforms.

VS Code is designed to be lightweight and fast, with a focus on simplicity and productivity. It includes a range of features for code editing, such as syntax highlighting, code completion, and error highlighting, as well as support for debugging and testing. VS Code is known for its extensibility and customization options. It includes numerous

built-in extensions, as well as a marketplace where developers can find and install extensions created by the community. These extensions can add new features to VS Code, such as support for additional languages or tools, or they can customize the editor to better suit the needs of the developer.

Visual Studio Code is a source code editor that can be used with a variety of programming languages, including C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, and Rust. It is based on the Electron framework,[20] which is used to develop Node.js web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps(formerly called Visual Studio Online and Visual Studio Team Services).

Out of the box, Visual Studio Code includes basic support for most common programming languages. This basic support includes syntax highlighting, bracket matching, code folding, and configurable snippets. Visual Studio Code also ships with IntelliSense for JavaScript, TypeScript, JSON, CSS, and HTML, as well as debugging support for Node.js. Support for additional languages can be provided by freely available extensions on the VS Code Marketplace.

Instead of a project system, it allows users to open one or more directories, which can then be saved in workspaces for future reuse. This allows it to operate as a language-agnostic code editor for any language. It supports many programming languages and a set of features that differ per language. Unwanted files and folders can be excluded from the project tree via the settings.

Source control is a built-in feature of Visual Studio Code. It has a dedicated tab inside of the menu bar where users can access version control settings and view changes made to the current project. To use the feature, Visual Studio Code must be linked to any supported version control system (Git, Apache Subversion, Perforce, etc.). This allows users to create repositories as well as to make push and pull requests directly from the Visual Studio Code program.

Chapter 5

METHODOLOGY

5.1 General Architecture

The below architectural diagram shows how the system works and it demonstrates the role of users in the developed program.

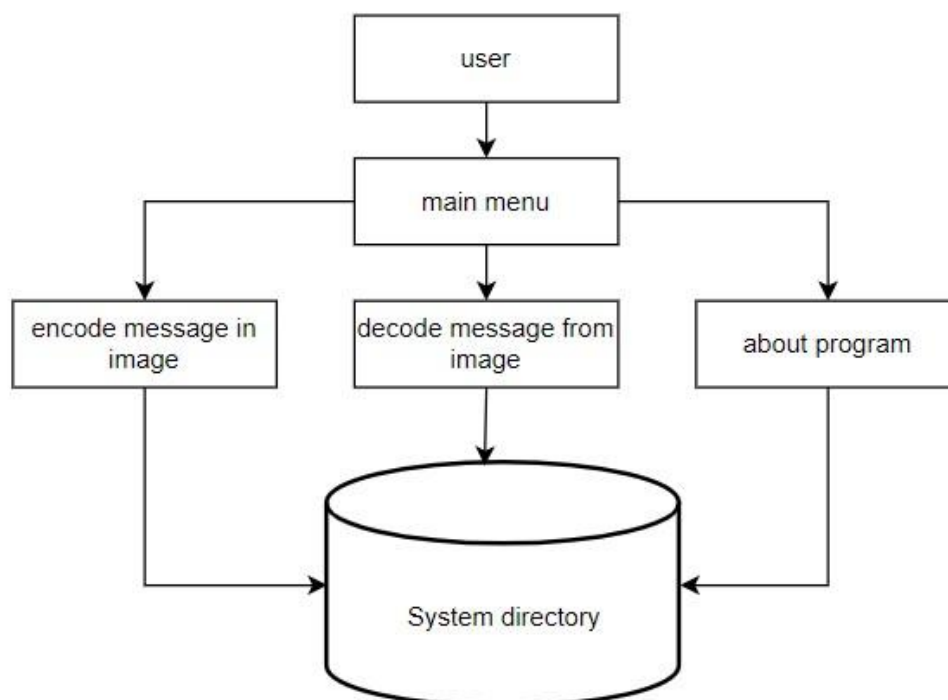


Figure 5.1: Architecture Diagram of hideEm

To get started, the user must access the main menu, which provides three distinct options. The first option is to encode a secret message within an image, which can only be decoded using this program. The second option is to decode a previously encoded message from an image. Finally, the third option provides detailed information about this program's features, capabilities, and usage instructions. If the user selects either the encoding or decoding option, the program will automatically navigate to the system directory, which serves as the default root directory. This directory is the central location that stores all the necessary files and data required for the program to perform its functions successfully. The program must locate this directory to operate correctly, and this is done automatically to ensure a seamless experience for the user.

5.2 Design Phase

5.2.1 Context Diagram

The below figure demonstrates the context diagram of the project.

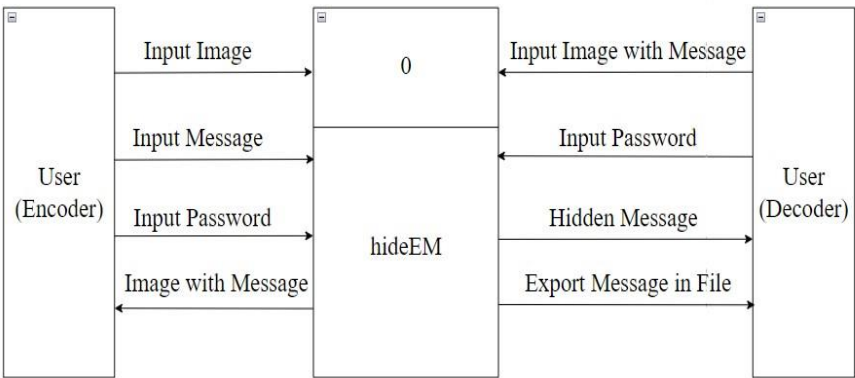


Figure 5.2: Context Diagram

5.2.2 Data Flow Diagram

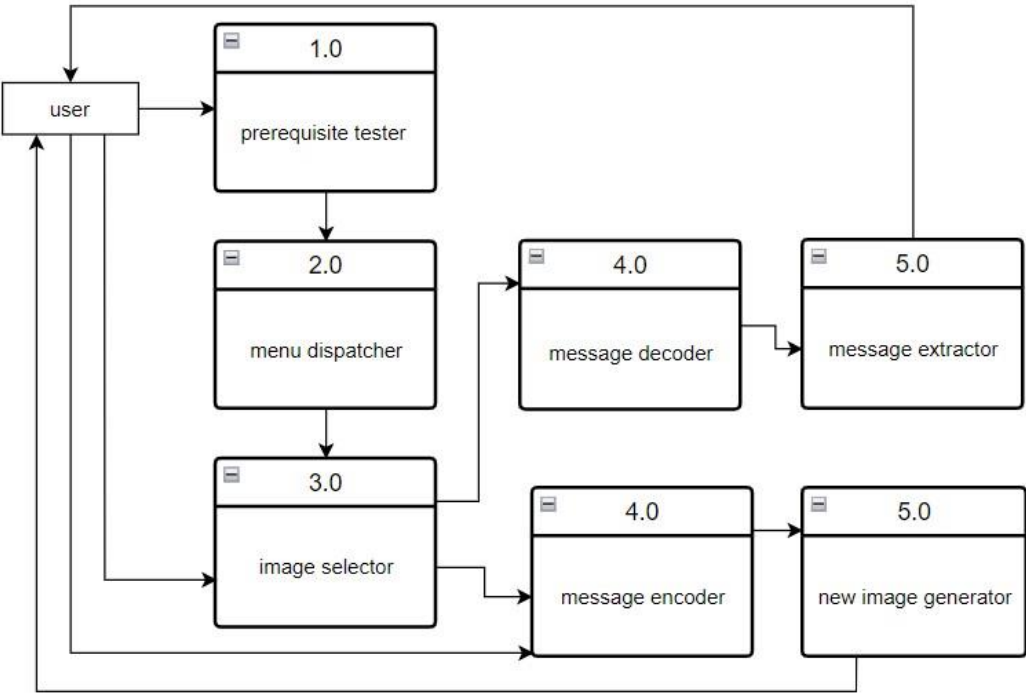


Figure 5.3: Data Flow Diagram

- **External Entities:**

1. **User:** Represents the external user interacting with the "hideEm" tool, providing input data and receiving the modified image as output.

- **Processes:**

1. **Prerequisite Tester:** This process verifies the prerequisites required for the tool to run, such as the availability of necessary libraries or dependencies.
2. **Menu Dispatcher:** Handles the user interface and menu navigation, guiding the user through the different functionalities of the tool.
3. **Image Selector:** Selects the original image file provided by the user for embedding the hidden message.
4. **Message Encoder:** Encrypts the message provided by the user using cryptographic algorithms to enhance security.
5. **Message Decoder:** Decrypts and extracts the hidden message from the modified image, if required by the user.
6. **Message Extractor:** Extracts the hidden message from the modified image.
7. **New Image Generator:** Generates the modified image with the hidden message embedded, or creates a new image file with the extracted message.

5.2.3 Use Case Diagram

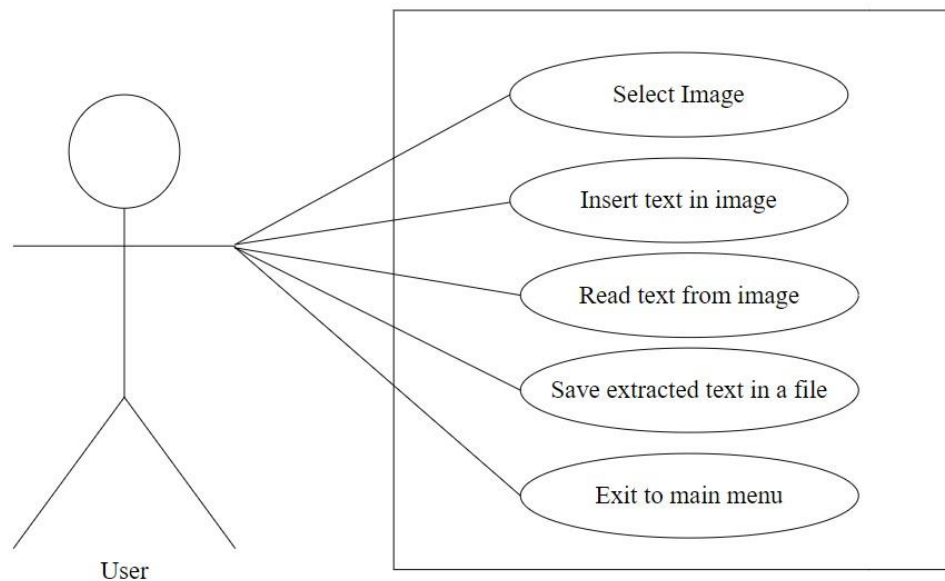


Figure 5.4: Use Case Diagram for hideEm

Select Image:

Actor: User

Description: The user selects an image file to be used as the carrier for hiding the text message.

Insert Text in an Image:

Actor: User

Description: The user provides a text message to be hidden within the selected image. The tool then encrypts the message and embeds it into the image using steganographic techniques.

Read Text from Image:

Actor: User

Description: The user selects an image file containing a hidden text message. The tool then extracts and decrypts the hidden message from the image, making it readable to the user.

Save Extracted Text from Image:

Actor: User

Description: After reading the extracted text from the image, the user has the option to save it to a file for future reference or further processing.

Exit to Main Menu:

Actor: User

Description: Allows the user to exit the current operation and return to the main menu of the "hideEm" tool, where they can choose other functionalities or perform additional tasks.

5.2.4 Activity Diagram

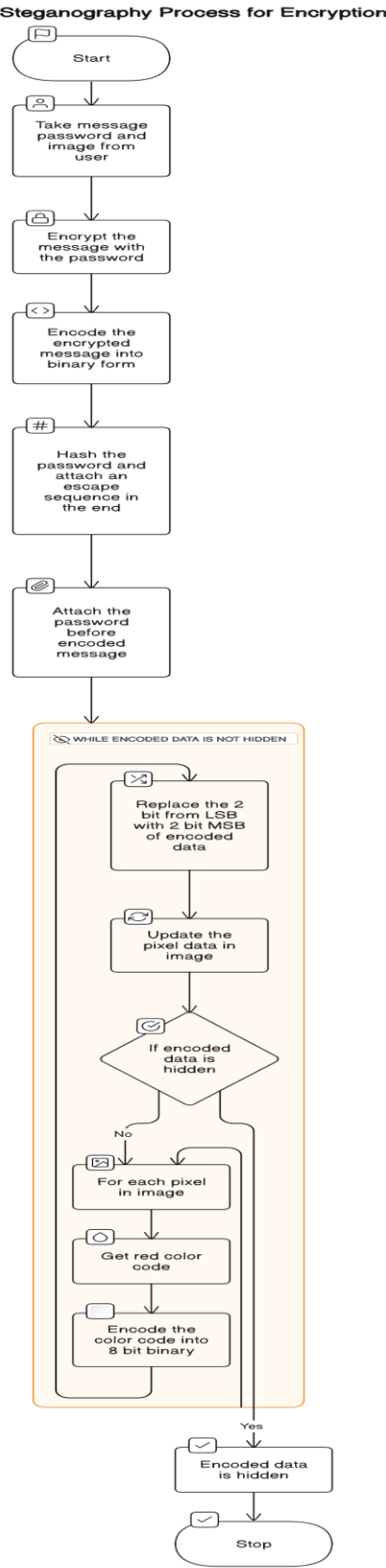


Figure 5.5: Activity Diagram for Encryption

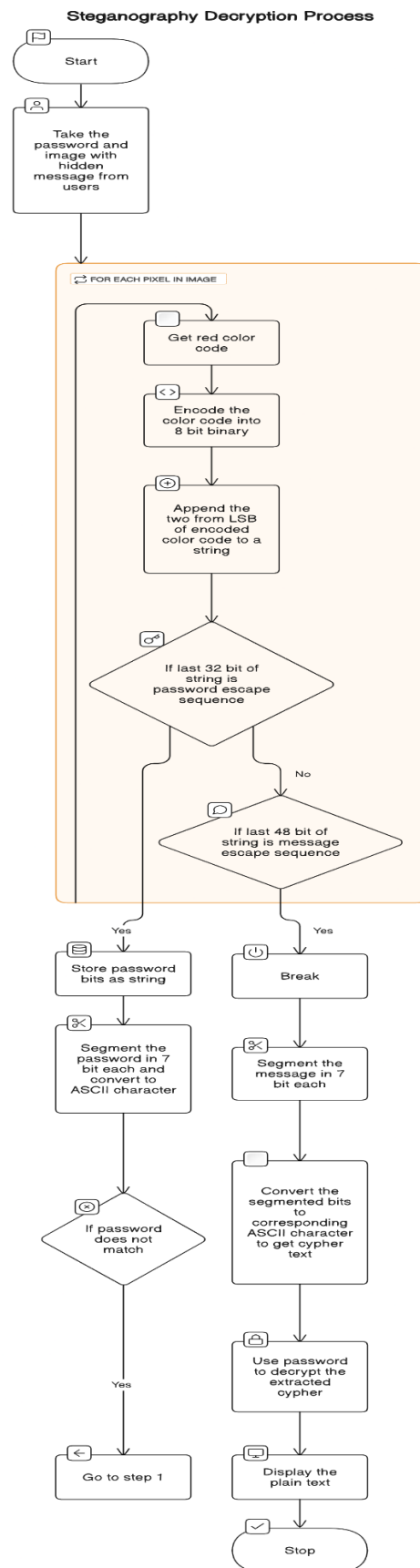


Figure 5.6: Activity Diagram for Decryption

The activity diagram describes how the image is loaded in the system; the encrypted text is then encoded into the image to develop a Stego-image for the encryption part as well as vice-versa for part of decryption.

5.2.5 Interface Structure Design

The program has a simplistic text-based interface that follows the following interface design.

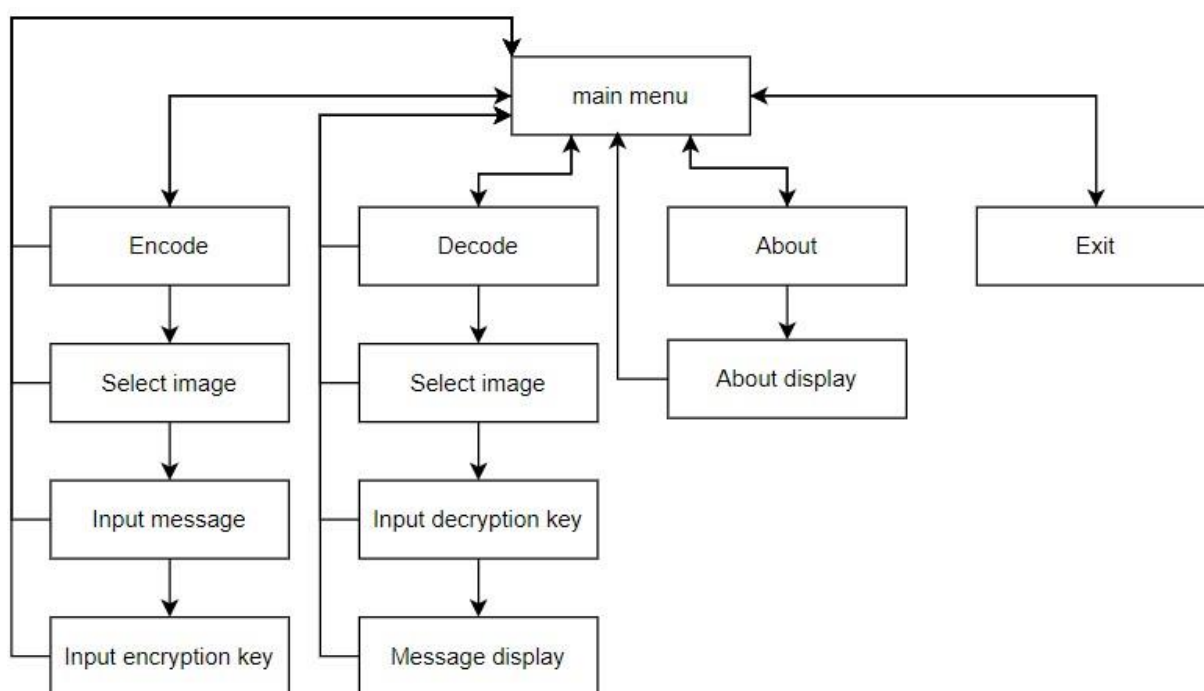


Figure 5.7: Interface Structure Design of hideEm

Chapter 6

ALGORITHM

6.1 LSB Algorithm

The LSB (Least Significant Bit) Algorithm, particularly when applied to the R-channel of an image, stands as one of the most prevalent methods in modern steganography. This technique capitalizes on the least noticeable bit of a pixel's information in an image. It proves most effective when the carrier file is longer than the message and when the image is grayscale. In a 24-bit image, LSB techniques can encode three bits into each pixel, facilitating the concealment of data within the image. Despite its simplicity in implementation, LSB insertion is susceptible to attacks. Steganography has found its way into various digital technologies, from embedding watermarks in currency to encoding metadata in MP3 files. It's even utilized for copyright protection, embedding ownership information within files. However, the fragility of LSB steganography is evident, as minor alterations to the image's color palette or simple manipulations can obliterate the hidden message, highlighting the need for robust security measures in digital concealment methods.

The R-channel LSB (Least Significant Bit) algorithm is a method used in steganography, which is the practice of hiding information within digital images. In this algorithm, a carrier image is chosen, and its red channel (one of the color channels in the image) is manipulated to conceal a secret message. Each pixel in the image is represented by three color values: red, green, and blue. By altering the least significant bit (the binary digit with the least impact on the pixel's color) of the red channel, the algorithm embeds bits of the secret message into the image. Since the changes made to the least significant bit are subtle, they are typically imperceptible to the human eye. To extract the hidden message, the process is reversed, and the altered least significant bits are read from the image. If encryption was used before embedding the message, decryption is required to reveal the original content. While the R-channel LSB algorithm is straightforward and effective, it's essential to recognize that it may not provide robust security against advanced detection methods. Therefore, additional security measures may be necessary depending on the application's requirements.

R-channel LSB algorithm serves as a versatile tool in digital image processing, offering a range of applications in communication, security, and copyright protection. Its ability to conceal information within images while maintaining visual integrity makes it a valuable technique in various domains. The R-channel LSB (Least Significant Bit) algorithm finds numerous applications in digital image processing and steganography. Some of its common uses include:

- **Secret Communication:** One of the primary uses of the R-channel LSB algorithm is for secret communication. By embedding messages or data within the least significant bits of the red channel of an image, users can conceal information within images, making it difficult for unauthorized individuals to detect or intercept the hidden data.
- **Copyright Protection:** The algorithm can also be employed for copyright protection in digital media. Copyright information or ownership details can be embedded within images using LSB steganography, allowing creators to assert ownership and protect their intellectual property rights.
- **Digital Watermarking:** R-channel LSB can be utilized for digital watermarking, where imperceptible marks or identifiers are embedded into images to authenticate their origin or ownership. Watermarking helps to deter unauthorized use or distribution of copyrighted images and serves as a form of digital authentication.
- **Data Hiding in Multimedia:** In addition to images, the R-channel LSB algorithm can be extended to hide data in multimedia files such as audio and video. By manipulating the least significant bits of multimedia files, users can embed additional data or metadata, enhancing the file's functionality or security.
- **Steganalysis and Security:** While the R-channel LSB algorithm is commonly used for covert communication, it is also subject to steganalysis techniques aimed at detecting hidden information. As a result, the algorithm is utilized in security applications to analyze and detect the presence of hidden data in digital images, aiding in forensic investigations and digital forensics.

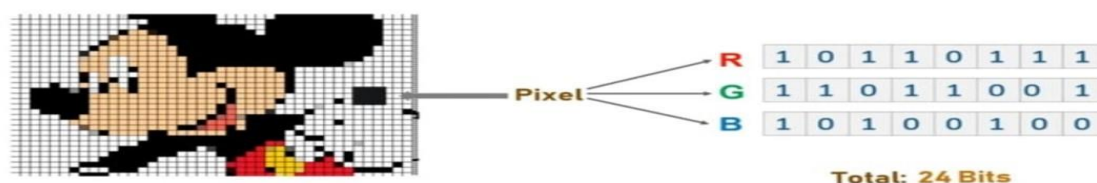


Figure 6.1: Image Pixel Distribution in RGB Channel

In Figure 6.1 above, the distribution of image pixels in the RGB (Red, Green, Blue) channel is depicted. The RGB channel comprises 24 bits in total, with each channel consisting of 8 bits. This configuration allows for the representation of colors in digital images by combining different intensities of red, green, and blue light. Each pixel in the image is represented by three sets of 8-bit values for red, green, and blue, respectively, determining the color and intensity of light emitted or reflected by that pixel. By manipulating these values, various colors can be produced, enabling the creation of vibrant and detailed images in digital media.

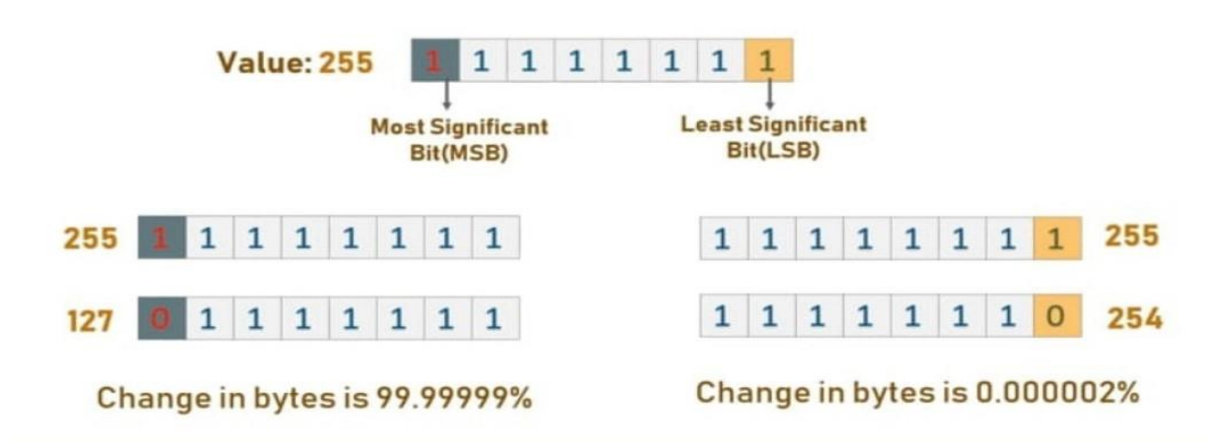


Figure 6.2: Impact on changing the MSB and LSB

In Figure 6.2 above, it's illustrated that modifying the Most Significant Bit (MSB) of a value will result in a more significant change to the final value compared to altering the Least Significant Bit (LSB). The MSB holds the highest weight in determining the value of a binary number, and thus, changing it can lead to substantial adjustments in the overall value. Conversely, the LSB carries the least weight and has minimal impact on the final value when altered. This observation underscores the principle that modifications to the MSB are more noticeable and impactful, while changes to the LSB are relatively subtle and have less influence on the outcome.

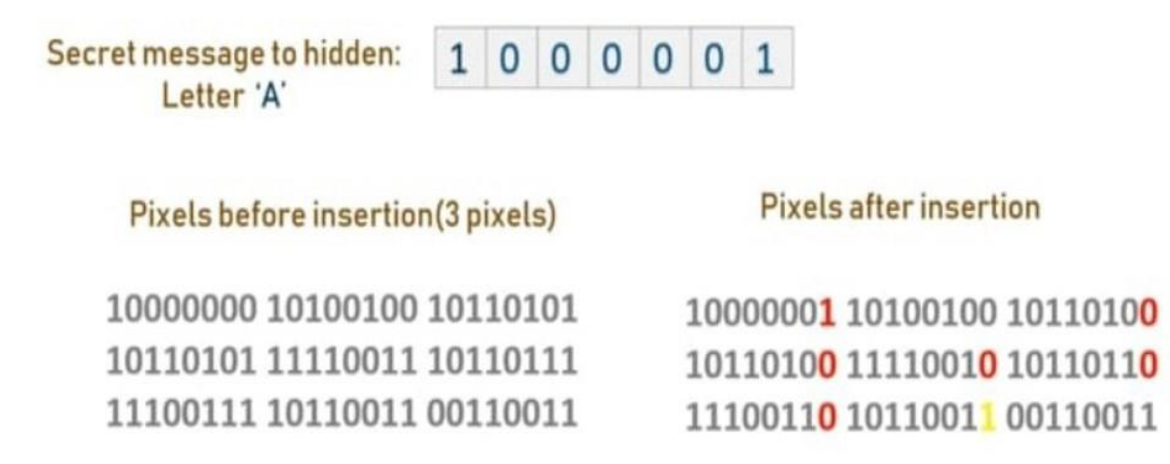


Figure 6.3: Pixel before and after insertion of the secret message

Encryption Algorithm:

- Step 1: Start
- Step 2: Take the message, password, and image from the user
- Step 3: Encrypt the message with the password
- Step 4: Encode the encrypted message into binary form
- Step 5: Hash the password and attach an escape sequence at the end
- Step 6: Attach the password before the encoded message
- Step 7: While encoded data is not hidden

For each pixel in the image

Get red color code

Encode the color code into an 8-bit binary

Replace the 2-bit from LSB with 2-bit MSB of encoded data

Update the pixel data in the image

If encoded data is hidden

Encoded data is hidden
- Step 8: Stop

Decryption Algorithm:

Step 1: Start

Step 2: Take the password and image with the hidden message from users

Step 3: For each pixel in the image

 Get red color code

 Encode the color code into an 8-bit binary

 Append the two from LSB of encoded color code to a string

 If the last 32 bits of the string is a password escape sequence

 Store password bits as a string

 Segment the password in 7-bit each and convert to ASCII character

 If the password does not match

 Go to step 1

 If the last 48 bits of the string is a message escape sequence

 Break

Step 4: Segment the message in 7-bit each

Step 5: Convert the segmented bits to corresponding ASCII characters to get cipher text

Step 6: Use a password to decrypt the extracted cipher

Step 7: Display the plain text

Step 8: Stop

Chapter 7

MODULES

7.1 PIL (Python Imaging Library)

The Python Imaging Library (PIL), now known as Pillow, stands as the foremost image-processing package for the Python programming language. It offers a comprehensive suite of lightweight tools designed to facilitate image editing, creation, and saving operations. Although support for the original Python Imaging Library ceased in 2011, the Pillow project emerged as a successor by forking the original PIL codebase and introducing Python 3.x compatibility. With its ongoing development and maintenance, Pillow has become the preferred choice for image-processing tasks in Python. Pillow boasts extensive support for a wide array of image file formats, including but not limited to BMP, PNG, JPEG, and TIFF. Its versatility is further demonstrated by its commitment to continuously expanding support for newer image formats through the addition of new file decoders. This proactive approach ensures that Pillow remains up-to-date with evolving industry standards and user requirements. In summary, Pillow serves as a powerful and user-friendly solution for image manipulation and processing in Python, offering a rich set of features and broad compatibility with various image file formats. Its active development community and commitment to ongoing improvement make it an indispensable tool for developers and enthusiasts working with images in the Python ecosystem.

Key features of the PIL library include:

- **Image Processing:** PIL provides a wide range of image processing capabilities, including resizing, cropping, rotating, flipping, and filtering images. These operations can be performed on individual images or applied to multiple images in batch processing.
- **Image Enhancement:** PIL offers tools for enhancing the quality and appearance of images. This includes adjusting brightness, contrast, saturation, and sharpness, as well as applying color corrections and gamma adjustments. **Image Filtering:** PIL supports various image filtering techniques, such as blurring, sharpening, edge detection, and noise reduction. These filters can be applied to images to achieve desired visual effects or to preprocess images before further analysis.

- **Image Drawing:** PIL allows users to draw shapes, text, and other graphics directly onto images. This feature is useful for annotating images, adding labels or captions, and creating visualizations.
- **Image File I/O:** PIL supports a wide range of image file formats for reading and writing images, including popular formats such as JPEG, PNG, GIF, BMP, and TIFF. It also provides functionality for converting between different image formats.
- **Image Manipulation:** PIL enables advanced image manipulation techniques, such as blending multiple images, compositing images with alpha transparency, and creating image mosaics or collages.

This module is not preloaded with Python. So, to install it execute the following command in the command line:

pip install pillow

7.2 Fernet

The Fernet library in Python is a part of the cryptography package and is specifically designed for secure encryption and decryption of data. Fernet is a symmetric encryption algorithm, meaning that it uses the same key for both encryption and decryption. It provides a simple and secure way to encrypt data, making it suitable for a wide range of applications, including securing sensitive information in web applications, storing encrypted data in databases, and transmitting encrypted messages over insecure channels. The fernet module of the cryptography package has inbuilt functions for the generation of the key, encryption of plaintext into ciphertext, and decryption of ciphertext into plaintext using the encrypt and decrypt methods respectively. The Fernet module guarantees that data encrypted using it cannot be further manipulated or read without the key. The Fernet library in Python offers a convenient and secure solution for encrypting and decrypting data. Its simplicity, robustness, and cross-platform compatibility make it a popular choice for securing sensitive information in Python applications.

Methods Used:

- **generate_key():** This method generates a new fernet key. The key must be kept safe as it is the most important component to decrypt the ciphertext. Also, if an intruder or hacker gets access to the key, they can not only read the data but also forge the data.

- **encrypt(data):** It encrypts data passed as a parameter to the method. The outcome of this encryption is known as a “Fernet token” which is the ciphertext. The encrypted token also contains the current timestamp when it was generated in plaintext. The encrypt method throws an exception if the data is not in bytes.

Key features of the Fernet library include:

- **Symmetric Encryption:** Fernet uses symmetric encryption, where the same secret key is used for both encryption and decryption. This simplifies the encryption process and makes it easier to manage keys securely.
- **Authenticated Encryption:** Fernet provides authenticated encryption, which ensures that the encrypted data cannot be tampered with or modified without detection. This protects against data integrity attacks and ensures that the decrypted data is authentic and unaltered.
- **Secure Key Generation:** Fernet generates secure random keys using cryptographic algorithms, ensuring that the keys are unpredictable and resistant to brute-force attacks.
- **Easy-to-Use API:** The Fernet library provides a simple and intuitive API for encrypting and decrypting data. Developers can easily integrate Fernet encryption into their Python applications with minimal effort.
- **Cross-Platform Compatibility:** Fernet encryption is compatible with different platforms and programming languages, making it easy to encrypt data in Python and decrypt it in other environments, or vice versa.

7.3 Datetime

The datetime module supplies classes for manipulating dates and times. While date and time arithmetic are supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation. The datetime module is a powerful and versatile tool for working with dates and times in Python, offering a wide range of functionalities for various date and time-related tasks. It offers functionalities for creating, manipulating, and formatting dates, times, and time intervals.

There are 6 main classes when it comes to datetime in Python.

- **Date:** By date, we mean the conventional concept of dates with effect on the Gregorian Calander. Very naturally, the date class in Python is explained with attributes like day, month, and year.
- **Time:** The next class that is discussed here is called time. This class is independent of any day. Here it is generally assumed that each day has $24*60*60$ seconds. The attributes of the time class in Python include minute, second, microsecond, hour, and tzinfo.
- **Datetime:** Datetime in Python is the combination between dates and times. The attributes of this class are similar to both date and separate classes. These attributes include day, month, year, minute, second, microsecond, hour, and tzinfo.
- **Timedelta:** You can think of timedelta as the variation between two different dates, time, or datetime examples to microsecond resolution.
- **Tzinfo:** tzinfo is nothing but an attribute. It can provide you different timezone-related information objects.
- **Timezone:** This class can deploy the tzinfo abstract base class in terms of a fixed offset from the UTC. Here, you are looking into the new UTC version 3.2.

7.4 Base64

A built-in module in Python, the base64 library offers functions for encoding and decoding binary data to and from base64 strings, effectively converting any binary data to plain text. A common encoding method called base64 converts binary data into a string of printable ASCII characters.

This module provides functions for encoding binary data to printable ASCII characters and decoding such encodings back to binary data. It provides encoding

and decoding functions for the encodings specified in RFC 4648, which defines the Base16, Base32, and Base64 algorithms, and for the de-facto standard Ascii85 and Base85 encodings.

The RFC 4648 encodings are suitable for encoding binary data so that they can be safely sent by email, used as parts of URLs, or included as part of an HTTP POST request. The encoding algorithm is not the same as the uuencode program.

There are two interfaces provided by this module. The modern interface supports encoding bytes-like objects to ASCII bytes and decoding bytes-like objects or strings containing ASCII to bytes. Both base-64 alphabets defined in RFC 4648 (normal, and URL- and filesystem-safe) are supported.

The legacy interface does not support decoding from strings, but it does provide functions for encoding and decoding to and from file objects. It only supports the Base64 standard alphabet, and it adds newlines every 76 characters as per RFC 2045. Note that if you are looking for RFC 2045 support you probably want to be looking at the email package instead.

Some key features of the base64 module:

- **Encoding:** The `base64.b64encode()` function takes binary data as input and returns a Base64-encoded bytes object. This function converts the binary data into a string of printable ASCII characters.
- **Decoding:** The `base64.b64decode()` function takes a Base64-encoded bytes object as input and returns the original binary data. This function reverses the encoding process, converting the Base64-encoded string back into its original binary form.
- **URL-safe Encoding:** The `base64.urlsafe_b64encode()` function performs Base64 encoding with URL-safe characters, suitable for use in URLs and filenames. This function replaces the standard Base64 characters `+` and `/` with `-` and `_`, respectively.
- **URL-safe Decoding:** The `base64.urlsafe_b64decode()` function decodes URL-safe Base64-encoded strings back into binary data. It accepts URL-safe Base64-encoded bytes objects as input and returns the original binary data.

- **Padding:** Base64-encoded strings may include padding characters at the end to ensure that the length of the encoded string is a multiple of four. The `base64.b64encode()` function automatically adds padding when necessary, and the `base64.b64decode()` function ignores padding during decoding.

7.5 Hashlib

The `hashlib` module in Python provides functions for hashing data using various cryptographic hash algorithms. Hashing is a process of converting input data (such as a string or a file) into a fixed-size string of bytes, typically to ensure data integrity, password storage, or digital signatures. Hashlib provides a convenient and efficient way to hash data for a wide range of applications, including data integrity verification, password storage, and digital signatures.

This module implements a common interface to many different secure hash and message digest algorithms. Included are the FIPS secure hash algorithms SHA1, SHA224, SHA256, SHA384, SHA512, (defined in the FIPS 180-4 standard), the SHA-3 series (defined in the FIPS 202 standard) as well as RSA's MD5 algorithm (defined in Internet RFC 1321). The terms "secure hash" and "message digest" are interchangeable. Older algorithms were called message digests. The modern term is secure hash.

There is one constructor method named for each type of hash. All return a hash object with the same simple interface. For example: use `sha256()` to create a SHA-256 hash object. You can now feed this object with bytes-like objects (normally bytes) using the `update` method. At any point, you can ask it for the digest of the concatenation of the data fed to it so far using the `digest()` or `hexdigest()` methods.

To allow multithreading, the Python GIL is released while computing a hash supplied more than 2047 bytes of data at once in its constructor or `.update` method.

Constructors for hash algorithms that are always present in this module are `sha1()`, `sha224()`, `sha256()`, `sha384()`, `sha512()`, `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()`, `shake_128()`, `shake_256()`, `blake2b()`, and `blake2s()`. `md5()` is normally available as well,

though it may be missing or blocked if you are using a rare “FIPS compliant” build of Python. These correspond to algorithms guaranteed.

Additional algorithms may also be available if your Python distribution’s hashlib was linked against a build of OpenSSL that provides others. Others are not guaranteed available on all installations and will only be accessible by name via `new()`.

Applications of hashlib are:

- **Password Storage:** Hashing algorithms provided by hashlib are commonly used for storing passwords securely in databases. Instead of storing passwords in plaintext, they are hashed using algorithms like SHA-256 or SHA-512 before being stored. When a user attempts to log in, their password is hashed and compared to the stored hash value for authentication.
- **Digital Signatures:** Hash functions play a crucial role in digital signatures, where they are used to generate a unique fingerprint (hash) of a message or document. This hash value is then encrypted with the sender's private key to create a digital signature. Recipients can verify the integrity and authenticity of the message by decrypting the digital signature using the sender's public key and comparing it to the hash of the original message.
- **Data Integrity Verification:** Hashing is often used to verify the integrity of data during transmission or storage. By generating a hash of the data before transmission or storage, and then comparing it to the hash of the received or retrieved data, one can ensure that the data has not been altered or corrupted.
- **File Verification:** When downloading files from the internet, users can verify the integrity of the downloaded files by comparing their hash values to those provided by the source. This ensures that the files have not been tampered with during transmission and that they are identical to the original files.
- **Cryptographic Key Derivation:** Hash functions are used in key derivation functions (KDFs) to derive cryptographic keys from passwords or other input data. This ensures that the keys used for encryption and decryption are securely generated from a relatively weak source of randomness.

- **Message Authentication Codes (MACs):** Hash functions are used in combination with secret keys to generate message authentication codes (MACs). MACs are used to authenticate the integrity and origin of messages in network communication protocols, ensuring that messages have not been tampered with during transmission.

7.6 OS

Python has a built-in `os` module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc. The `os` module has the following set of methods and constants.

This module provides a portable way of using operating system-dependent functionality. If you just want to read or write a file see `open()`, if you want to manipulate paths, see the `os.path` module, and if you want to read all the lines in all the files on the command line see the file input module. For creating temporary files and directories see the `tempfile` module, and for high-level file and directory handling see the `shutil` module.

Notes on the availability of these functions:

- The design of all built-in operating system-dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function `os.stat(path)` returns stat information about the path in the same format (which happens to have originated with the POSIX interface).
- Extensions peculiar to a particular operating system are also available through the `os` module, but using them is of course a threat to portability.
- All functions accepting path or file names accept both bytes and string objects and result in an object of the same type if a path or file name is returned.
- On VxWorks, `os.popen`, `os.fork`, `os.execv` and `os.spawn*p*` are not supported.
- On WebAssembly platforms `wasm32-emscripten` and `wasm32-wasi`, large parts of the `os` module are not available or behave differently. API related to processes (e.g. `fork()`, `execve()`), signals (e.g. `kill()`, `wait()`), and resources (e.g. `nice()`) are not available. Others like `getuid()` and `getpid()` are emulated or stubs.

Some key features and functions provided by the os module are:

1. File and Directory Operations:

- **os.getcwd():** Get the current working directory.
- **os.chdir(path):** Change the current working directory to the specified path.
- **os.listdir(path='.'):** List the files and directories in the specified directory.
- **os.mkdir(path):** Create a new directory.
- **os.makedirs(path):** Create a new directory recursively, creating any missing intermediate directories.
- **os.remove(path):** Remove a file.
- **os.rmdir(path):** Remove an empty directory.
- **os.removedirs(path):** Remove a directory and any empty parent directories.
- **os.rename(src, dst):** Rename a file or directory.

2. Path Manipulation:

- **os.path.join(path1, path2, ...):** Join one or more path components into a single path.
- **os.path.abspath(path):** Return the absolute version of a path.
- **os.path.basename(path):** Return the base name of a path.
- **os.path.dirname(path):** Return the directory name of a path.
- **os.path.exists(path):** Check if a path exists.
- **os.path.isfile(path):** Check if a path is a regular file.
- **os.path.isdir(path):** Check if a path is a directory.
- **os.path.splitext(path):** Split the extension from a path.

3. Process Management:

- **os.system(command):** Execute the command in a subshell (not recommended for security reasons).
- **os.spawnv(mode, path, args):** Spawn a new process using the given executable file.
- **os.spawnve(mode, path, args, env):** Spawn a new process using the given executable file and environment variables.
- **os.execv(path, args):** Replace the current process with a new one using the given executable file.
- **os.execl(path, arg0, arg1, ...):** Replace the current process with a new one using the given executable file and arguments.

4. Environment Variables:

- **os.environ:** A dictionary containing the current environment variables.
- **os.getenv(key, default=None):** Get the value of an environment variable.
- **os.putenv(key, value):** Set the value of an environment variable.

5. Miscellaneous:

- **os.getlogin():** Get the name of the user logged in on the controlling terminal.
- **os.getpid():** Get the current process ID.
- **os.getppid():** Get the parent process ID.
- **os.urandom(n):** Generate a string of n random bytes suitable for cryptographic use.

Chapter 8

IMPLEMENTATION AND TESTING

8.1 Input and Output

Input is taken from the user as images and text. The image includes an original image to be encoded while the text can be secret text as well as a password for encryption. Output is generated in the form of a Stego-image from which the message can be decoded and decryption. The message can also be saved as a CSV file.

8.2 Implementation

8.2.1 Tools Used

- **Python:** Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming. This project is fully powered by Python and its various modules like Python Image Library (PIL), OS, time, cryptography, base64 and hashlib.
- **PyCharm:** PyCharm version 2023.2 is used as an Integrated Development Environment for the project. All the development part of the project is done with the IDE.
- **Bash shell:** Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. First released in 1989, it has been used as the default login shell for most Linux distributions. Bash was one of the first programs Linus Torvalds ported to Linux, alongside GCC. This tool is used to run the Python file in the terminal.
- **MS-Office:** The Microsoft Office package is used in the various phases of the project.

- **Draw.io online:** This online tool is used to draw various required diagrams during the project and documentation process. Draw.io is a free, online diagramming tool that allows you to create flowcharts, diagrams, mind maps, organization charts, and much more. A web-based application, Draw.io is fully integrated with Google Drive.
- **Excalidraw.io online:** Excalidraw.io is an online tool used for creating diagrams, including various types of diagrams such as flowcharts, wireframes, and sketches. Excalidraw is a collaborative whiteboard tool that allows users to create diagrams simply and intuitively. While it may offer features to assist with diagram creation, such as shape recognition or alignment tools, it does not primarily rely on artificial intelligence for its functionality.

8.2.2 Implementation Details of Modules

The project hideEm follows a modular architecture, where the application is built by integrating multiple modules. Each module serves a specific purpose or functionality within the program, and they are developed independently before being integrated to form the complete application. This modular approach allows for better organization, maintenance, and scalability of the codebase. Additionally, it facilitates easier testing and debugging of individual components, as well as the ability to replace or update modules without affecting the entire system. The modular design of hideEm enhances its flexibility and robustness, enabling efficient development and management of the software.

The menu Selector module in the program serves as the initial entry point for users and facilitates navigation among various options available in the application. When the program is initiated, the Menu Selector module is invoked, presenting users with a menu of options such as hiding a message, viewing a hidden message, exiting the program, accessing help, and more. Users can input their choice, and the Menu Selector module reads this input to determine the next action to be performed. Based on the user's selection, the module switches the program to execute the corresponding task or functionality, directing the flow of the program accordingly. As the index module, it plays a crucial role in orchestrating the overall functionality of the application, providing users with a seamless and intuitive interface to interact with the system.


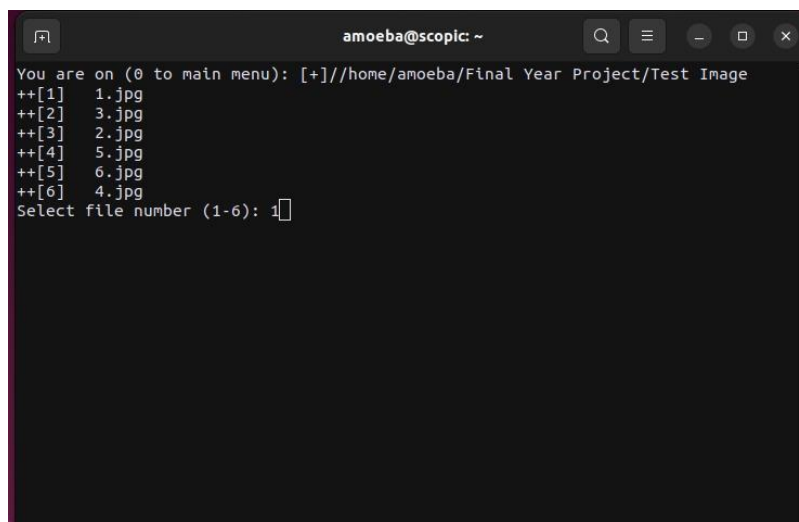
```
amoeba@scopic: ~  
----6EB46438C34E724A166CB10DC138A803564CD1----  
  
-----CD6B81C23EA06ABC14F8AB895A9BE4DC679158-----  
Main menu of hideEm  
++[0]   Exit  
++[1]   Hide message in image  
++[2]   Read message from image  
++[3]   About the application  
Select your option (0-3):
```

Figure 8.1: Menu Selector of hideEm

Image Selector module in the program enables users to select the desired image in which data is to be hidden. This module operates by first presenting users with a list of available files and folders within the current directory. Users can then input a numeric value corresponding to their choice. If a folder is selected, the module navigates the user to that directory for further exploration. However, if a file is chosen, the module verifies whether the selected file is an image. If the chosen file is indeed an image, the program proceeds to select that file. Additionally, the Image Selector module includes error-handling mechanisms to detect and handle invalid user input, ensuring a smooth and error-free selection process. The Image Selector module plays a crucial role in facilitating the selection of the appropriate image file for data hiding within the program.



```
amoeba@scopic: ~  
You are on (0 to main menu): [+]//home/amoeba/Final Year Project/Test Image  
++[1] 1.jpg  
++[2] 3.jpg  
++[3] 2.jpg  
++[4] 5.jpg  
++[5] 6.jpg  
++[6] 4.jpg  
Select file number (1-6): 1
```

Figure 8.2: Image Selector of hideEm

Message Encryption Module serves a critical role in the program by handling the encryption of user messages before they are hidden within images. This module prompts the user to input both an encryption key and the message to be encrypted. Using the AES algorithm, the module encrypts the message using the provided key, ensuring its confidentiality. Additionally, the module hashes the encryption key to further secure it and attaches an escape sequence at the end of both the password hash and the ciphertext. This additional step enhances security by preventing unauthorized access to the encrypted message, even if hackers manage to extract pixel data from the image. By encrypting the message and securing the encryption key, the Message Encryption Module effectively protects sensitive information from potential threats.

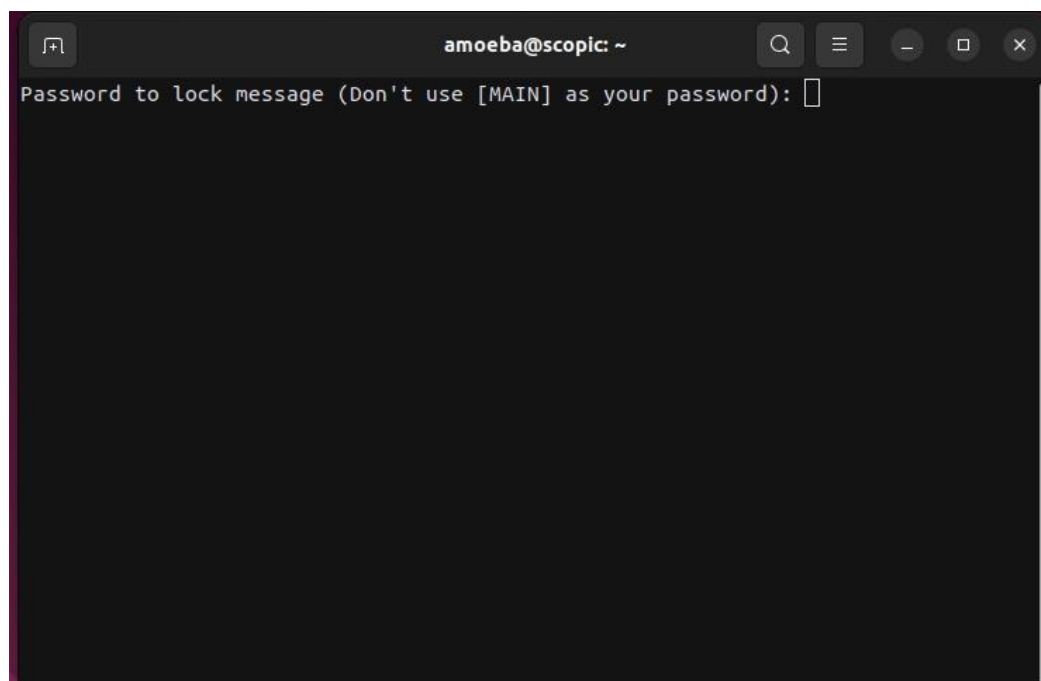


Figure 8.3: Encryption Module

The Image Generator Module plays a crucial role in the program by working in conjunction with the Message Encryption Module to conceal encrypted messages within images. This module utilizes the Image Selector Module to obtain the user-selected image and then employs the Python Imaging Library (PIL) to extract the pixel data of the image. Once the pixel data is extracted, the module manipulates the red pixel color code to embed the encrypted message within the image. Additionally, the module creates a copy of the entire color palette and generates a new image with the hidden message embedded in it. Finally, it saves a copy of the modified image in the user's home directory, ensuring that the original image remains intact. The Image Generator module is responsible for seamlessly integrating encrypted messages into images while preserving the integrity of the original image file.

Message Extractor Module serves as a crucial component in the program, allowing users to retrieve hidden data from images that contain encrypted messages. When invoked, this module prompts the user to provide the decryption key. If the provided decryption key matches the encryption key used to hide the message, the module proceeds to extract the cipher from the image. Once the cipher is extracted, the module removes any escape

sequences from the string, ensuring that only the genuine cipher is passed to the Message Decryption Module. This ensures the integrity of the extracted data and prepares it for decryption. The Message Extractor Module facilitates the retrieval of hidden messages from images while maintaining data integrity and security through the validation of decryption keys.

Message Decryption Module is a vital component of the program responsible for decrypting messages extracted from images. Upon receiving the cipher from the Message Extractor Module, this module utilizes the provided decryption key to decrypt the message successfully. To accomplish this task, the module makes use of the Cryptography module from the Python library, which provides robust encryption and decryption functionalities. By leveraging this module, the Message Decryption Module ensures the security and integrity of the decrypted message, allowing users to access the hidden data with confidence. The Message Decryption Module plays a crucial role in the process of retrieving and decrypting hidden messages from images, thereby enabling users to access concealed information securely and reliably.

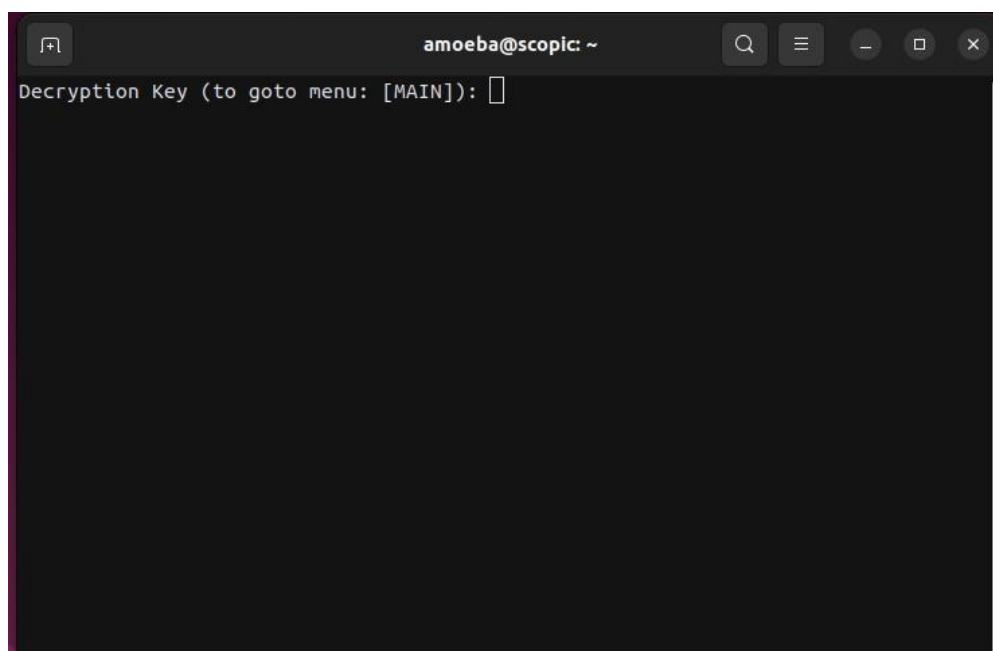


Figure 8.4: Decryption Module

Message Exporter module serves a dual purpose within the program. Firstly, it acts as a display for the extracted message, allowing users to view the decrypted content directly within the program interface. This feature enhances user convenience by providing immediate access to concealed information. Additionally, the module facilitates the export of the decrypted message by saving it into a file in the user's home directory. Users have the option to choose whether to export the message or simply navigate back to the main menu using the provided commands. This flexibility ensures that users can manage the extracted data according to their preferences, whether they choose to store it for future reference or simply view it temporarily within the program interface. The Message Exporter module enhances the usability of the program by offering convenient options for accessing and managing decrypted messages.

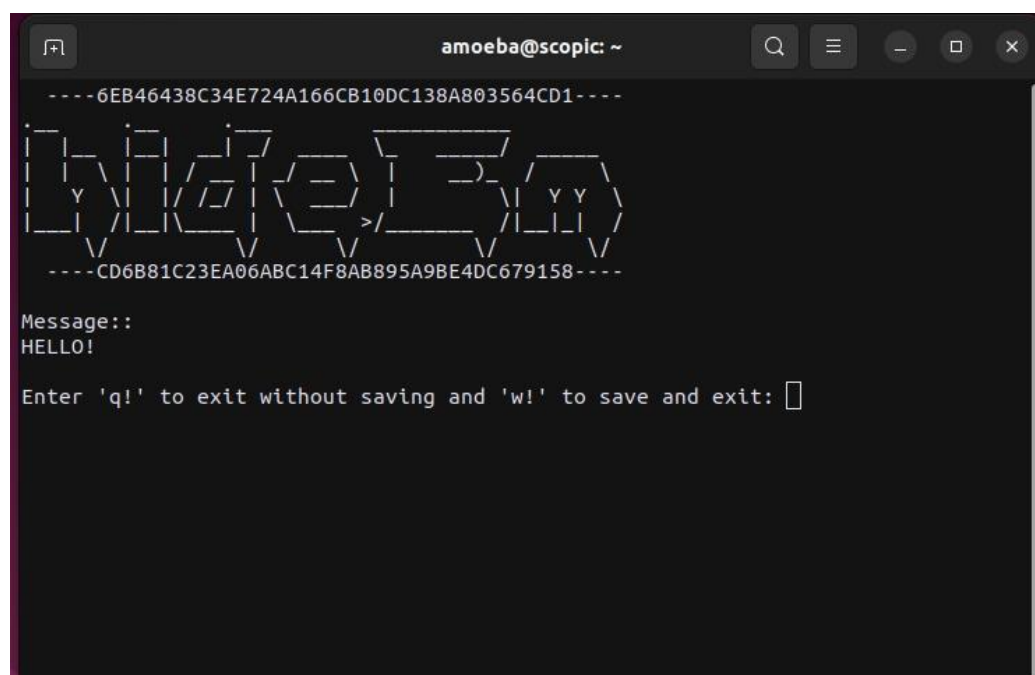


Figure 8.5: Message Exporter Module

8.3 Testing

System testing of the software is a testing conducted on a system that is a complete, integrated system that works as a whole. System testing is a critical testing procedure that must be conducted by a software developer before the system released. During system testing, it can evaluate the system's compliance with its activities in system testing

test not only the design of the system, but also the behavior and the believed expectations result from the customer. In addition, various complex test cases that used to test the system are according to the business process requirements which are collected from the user. Meanwhile, errors or bugs that detected during the testing is required software developer look into it from the initial step of the business process to the end of the process to ensure it have expected result in order to solve the errors or bugs to determine the degree of system stability.

Testing is a crucial phase in the development process of any software application, including the hideEm image steganography tool. The testing phase ensures that the application functions as intended, meets user requirements, and operates reliably under various scenarios. In the context of hideEm, testing encompasses several key aspects:

- **Unit Testing:** Unit testing involves testing individual components or modules of the application in isolation. Each module, such as the Message Encryption Module, Image Generator Module, Message Extractor Module, and others, is tested to verify its functionality according to specifications. This ensures that each component performs its intended task accurately and reliably.
- **Integration Testing:** Integration testing focuses on verifying the interaction between different modules or components of the application. This ensures that data flows correctly between modules and that they work together seamlessly to achieve the desired functionality. For hideEm, integration testing ensures that modules such as the Message Encryption Module and Image Generator Module interact correctly to embed messages within images.
- **Functional Testing:** Functional testing evaluates the application's behavior against specified functional requirements. Test cases are designed to validate various features and functionalities of hideEm, such as hiding messages within images, extracting hidden messages, decrypting messages, and exporting decrypted messages. Functional testing ensures that the application performs its intended functions accurately and reliably.
- **Security Testing:** Security testing is crucial for a steganography tool like hideEm, which deals with sensitive data encryption and hiding. This testing ensures that the application's encryption and hiding mechanisms are robust and secure against potential attacks or vulnerabilities. Security testing may involve validating encryption

algorithms, checking for vulnerabilities in input validation, and ensuring secure storage of sensitive information.

- **Regression Testing:** Regression testing ensures that recent code changes or updates do not adversely affect existing functionality. It involves retesting previously tested features and functionalities to ensure that they still work correctly after modifications. Regression testing is essential for maintaining the overall quality and stability of hideEm across different updates.

8.3.1 Unit Testing

Unit testing is a fundamental aspect of software development, focusing on testing individual units or components of code in isolation. It serves to detect and address defects early in the development cycle, ensuring the reliability and correctness of the software. By automating tests and isolating units from external dependencies, developers can verify that each component behaves as expected and meets its specifications. Unit tests typically include assertions to validate the behavior of the code, and they should execute quickly and produce consistent results. Unit testing not only helps maintain code quality but also facilitates rapid development cycles by providing fast feedback to developers. Additionally, it serves as a form of regression testing, guarding against unintended consequences of code changes. Unit testing is a crucial practice in software development for ensuring the robustness and stability of applications.

Unit Testing is a primary testing process for the developed hideEm system. It involves testing individual units or components of the software, such as modules, functions, or procedures, to verify their correctness and suitability for use. This testing method allows developers to identify any issues or defects within these units before integrating them into the larger system. Unit Testing is essential for ensuring the functional correctness of each independent module and detecting any errors early in the development cycle. It serves as the first level of testing before integration testing, helping to maintain the overall quality and reliability of the hideEm system.

Here, in the unit testing all the major functions being provided by each module of the project are tested with predefined conditions and outcomes are mentioned below.

Table 8.1: Test case for Menu Selector

Test case	Objective	Task	Expected Outcome	Obtained Outcome
Menu selection with invalid option	To select task from the menu	User inputs a random number	An error message is displayed	Failed
Provides option listed in menu	To select task from the menu	User inputs a number shown in the menu	Image selection option will be available	Passed

Table 8.2: Test Case for Image Selector

Test case	Objective	Task	Expected Outcome	Obtained Outcome
Image selection using invalid path	To provide image to the program	User inputs a random path to image	An error message is displayed	Failed
Selecting non image files	To provide image to the program	User selects text file	An error message is displayed	Failed
Providing non existing index of the files	To provide image to the program	User provide an alpha numeric value	An error message is displayed	Failed
Providing valid input	To provide image to the program	User provides available index value of file	User is asked to provide encryption key	Passed

Table 8.3: Test Case for Message Decryption

Test case	Objective	Task	Expected Outcome	Obtained Outcome
Decrypting message with false password	To decrypt the encrypted message	User inputs a random password	User is directed to main menu with error	Failed
Decrypting message with true password	To decrypt encrypted password	User inputs correct password	User is providing with decrypted text	Passed

Table 8.4: Test case for Message Exporter

Test case	Objective	Task	Expected Outcome	Obtained Outcome
Saving message with random input	To export message in text file	User provides random input	An error message is displayed	Failed
Saving message with quit command	To export message in text file	User provides the quit command	User is directed to main menu	Failed
Saving message with save and quit command	To export message in text file	User provides input for save and quit	The message file is saved and location is displayed	Passed

8.3.2 Integration Testing

Integration testing indeed plays a critical role in the software testing process, typically occurring as the second level following unit testing. Unlike unit testing, which focuses on testing individual components or units in isolation, integration testing involves verifying that these units work together harmoniously as designed. Integration testing specifically targets errors that may arise from the combination of these modules. Its primary objective is to ensure that all units collaborate effectively to perform their tasks correctly within the broader context of the software system. Through integration testing, potential defects and inconsistencies in the interaction between modules are identified and addressed, contributing to the overall quality and reliability of the software product.

8.3.3 Acceptance Testing

Acceptance Testing is a method of software testing where a system undergoes evaluation for acceptability, aiming to ensure compliance with business requirements and readiness for delivery. This formal testing process aligns with user needs, requirements, and business processes to assess whether the system meets acceptance criteria. Typically conducted by end users or clients, acceptance testing verifies the software's suitability before deployment to the production environment. It occurs after functional, integration, and system testing phases, focusing on validating end-to-end business flow rather than simple errors or system functionality. Carried out in a dedicated testing environment with production-like data, acceptance testing serves as a crucial step in determining whether the system is ready for acceptance and deployment.

8.3.4 Test Case for System Testing

The program is fully tested for its performance by providing various inputs. The test results are shown below.

Table 8.5: System Testing for hideEm

S.N	Test Case	Objectives	Expected Outcome	Remarks
1	Selecting menu options	To check whether the indexed function are accessible	The corresponding function to the index should run	Passed
2	Selecting image	To check if only images are being selected	Only files with JPG, PNG and JPEG should be selected	Passed
3	Generating new image	To check is new image with message is generated in user's home directory	An image, looking same as the processed image should be in user's home directory	Passed
4	Decrypting message	To check if the message is in human readable form	The displayed message should be in human readable form	Passed
5	Exporting message	To check if the decrypted message is saved in text file	The message should be saved in user's home directory	Passed
6	Accessing menu from other modules	To check if user can navigate to menu while using other features	User should be redirected to menu, discarding the current task	Passed
7	Exiting the program	To check if user can exit program without causing error	User should get exit message and exit the program	Passed
8	Setting up system	To check if alias for program is made in bashrc file	User must be able to access program from any directory using its name	Passed

Chapter 9

WORKING

9.1 Efficiency of the proposed system

The proposed system uses very less computational resources and is free of cost when it comes to its implementation. So, it is very efficient moreover it can be improved in performance by giving a stringer confidence interval for recognition.

9.2 PSNR and MSE

Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE) are widely used metrics in image processing to evaluate the quality of reconstructed or processed images compared to their originals.

PSNR measures the quality of the reconstruction by comparing the original and reconstructed images. It is expressed in decibels (dB) and represents the ratio between the maximum possible power of a signal (in this case, the original image) and the power of the noise (the difference between the original and reconstructed images). The formula for PSNR is:

$$\text{PSNR} = 10 \cdot \log_{10}(\text{MSE} / \text{MAX}^2)$$

where:

- MAX is the maximum possible pixel value of the image (e.g., 255 for an 8-bit grayscale image or 1 for a normalized image).

- MSE is the Mean Squared Error between the original and reconstructed images.

MSE quantifies the average squared difference between corresponding pixels of the original and reconstructed images. It provides a numerical measure of the average magnitude of the error between the two images. The formula for MSE is:

$$\text{MSE} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (I(i,j) - K(i,j))^2$$

where:

- $I(i, j)$ is the pixel value of the original image at position (i, j) .
- $K(i, j)$ is the pixel value of the reconstructed image at position (i, j) .
- M and N are the dimensions of the images.

In summary, PSNR provides a logarithmic measure of the ratio between the maximum possible signal power and the power of the noise, while MSE quantifies the average squared difference between corresponding pixels of the original and reconstructed images. Both metrics are commonly used to assess the fidelity and quality of image reconstructions in various image processing applications.

9.3 Sample Code

```
import os

import selectImage as si
import time
import intro
import about
import hideMessage as hm
import showMessgae as sm

def checkPython():
    check = os.system('python --version')
    check2 = os.system('python3 --version')
    if check != 0 and check2 != 0:
        print("Err:: This program needs python to run. Please install
python and try again...")
        quit()

def showErr():
    global error
    return error
    error = ""

def main(ack):
    os.chdir(f"{os.path.dirname(__file__)}")
    global error
    si.clear()
    intro.showBrand()

    if ack == "hidden":
```

```
print(f"\nYour message is hidden in {os.path.expanduser('~')}")
    if ack == "about":
        print(f"\nDeveloped by Suvam and team! ")

    if len(error) > 1:
        print(f'\nError::: {showErr()}\n')

    print("\nMain menu of hideEm\n")
    print("++[0]\tExit\n")
    print("++[1]\tHide message in image")
    print("++[2]\tRead message from image")
    print("++[3]\tAbout the application")
    print("\n")

    try:
        selectedNumber = int(input(f"Select your option (0-{optionRange}):
"))
        if selectedNumber == 0:
            si.clear()
            print(open("./txtFiles/smile.txt", "r").read())
            time.sleep(5)
            si.clear()
            quit()
        elif selectedNumber == 1:
            si.start()
            print(f"Full path of your image is {si.getName()}")
            time.sleep(0.5)
            if si.getName() == "[EXIT]":
                main("")
            else:
                hm.hide(si.getName())
                main("hidden")
        elif selectedNumber == 2:
            si.start()
            print(f"Full path of your image is {si.getName()}")
            time.sleep(0.5)
            if si.getName() == "[EXIT]":
                main("")
            else:
                sm.show(si.getName())
                main("shown")
        elif selectedNumber == 3:
            about.aboutApp()
            main("about")
        elif selectedNumber > 3 or selectedNumber < 0:
            error = "Number out of range"
            main("")
        else:
```

```
        error = "Unexpected error occurred..."
    except ValueError:
        error = "Only number accepted"
    main("")

os.chdir(f"{os.path.dirname(__file__)}")
error = ""
aspShowRun = False
optionRange = "3"
si.clear()
time.sleep(0.05)
si.clear()
checkPython()
main("")
```

```
import time
from datetime import datetime
import os
from PIL import Image
from cryptography.fernet import Fernet
import base64
import hashlib

def getText():
    password = input("Password to lock message (Don't use [MAIN] as your password): ")
    os.system("cls")
    remain = 32 - len(password)
    password = password + '0' * remain
    passByte = bytes(password, 'ascii')
    hashPass = hashlib.md5(passByte)
    basePass = base64.b64encode(hashPass.digest())
    hashPass = str(hashPass.digest())
    password = hashPass + "[^]"

    message = input("Enter the message: \n")
    fernet = Fernet(basePass)
    message = bytes(message, 'ascii')
    message = fernet.encrypt(message)
    message = str(message) + " [END]"

    encodedText = ""
    text = password + message
    for char in text:
        acsiiOfChar = ord(char)
        binOfChar = f"{acsiiOfChar:08b}"
```

```
        encodedText = encodedText + binOfChar

    return encodedText
def hide(imageName):
    os.system("cls")
    text = getText()
    os.system("cls")
    pixelToHide = len(text) / 2
    img = Image.open(imageName)
    loadImg = img.load()
    pixelCount = 0
    textStart = 0
    textEnd = textStart + 2
    for y in range(img.size[1]):
        if pixelCount == pixelToHide:
            break
        for x in range(img.size[0]):
            r, g, b = img.getpixel((x, y))
            binR = f"{r:08b}"
            textSection = text[textStart: textEnd]
            textStart = textStart + 2
            textEnd = textStart + 2
            encodedBinR = binR[:6] + textSection
            encodedDecR = int(encodedBinR, 2)
            loadImg[x, y] = (encodedDecR, g, b)
            pixelCount = pixelCount + 1

        if pixelCount == pixelToHide:
            break

    img.save(f"{os.path.expanduser('~')}/encoded-{datetime.now()}.{'jpg'}",
format="jpg")
    img.close()
    print("\nMessage hidden successfully...")
    time.sleep(1)

import os
import time

def clear():
    os.system('cls')
def showErr():
    global error
    return error
def start():
    global admin, userDir, error
    if len(error) > 0:
```

```
        print(f'\nError::: {showErr()}\n')
        error = ""
        time.sleep(0.5)
    userDir = input("Directory you want to begin with (/): ")
    if userDir != "" and os.path.exists(userDir):
        select()
    elif userDir == "":
        print("\nSelecting root (/) directory as default directory\n")
        userDir = "../Users/dell/Desktop/8th sem/final year project/PSNR
calc/test_image"
        time.sleep(0.5)
        select()
    elif not os.path.exists(userDir):
        error = "Given directory is not valid"
        time.sleep(0.5)
        start()
    else:
        print(f"\nError on selection: Please report to {admin}\n")

def getLenOfDir():
    global userDir
    return len(os.listdir(userDir))

def select():
    global userDir, error, imageName, admin

    cls()

    if len(error) > 0:
        print(f'\nError::: {showErr()}\n')
        error = ""

    print(f"You are on (0 to main menu): [+] {userDir}")
    for count, file in enumerate(os.listdir(userDir)):
        if file.endswith(".jpg") or file.endswith(".jpeg") or
file.endswith(".png") or not file.startswith("."):
            print(f"++[{count + 1}]\t{file}")

    try:
        fileNumber = int(input(f"Select file number (1-{getLenOfDir()}):
"))
        if getLenOfDir() >= fileNumber >= 1:
            selectedFile = os.listdir(userDir)[fileNumber - 1]
            newUserDir = f"{userDir}/{selectedFile}"
            if os.path.isdir(newUserDir) and '.' not in selectedFile:
                os.chdir(newUserDir)
                userDir = newUserDir
```

```

        select()
    else:
        if selectedFile.endswith('.png') or
selectedFile.endswith('.jpg') or selectedFile.endswith('.jpeg'):
            imageName = f'{userDir}/{selectedFile}'
        else:
            error = "Only .jpeg and .jpg file format are supported
for now"
            select()

    elif fileNumber == 0:
        end()

    else:
        error = "Number out of range"
        select()
except ValueError:
    error = "Only numbers accepted"
    select()

def getName():
    global imageName
    return imageName

def end():
    global imageName
    imageName = "[EXIT]"
admin = ""
error = ""
imageName = None
userDir = None

```

```

import base64
import hashlib
from cryptography.fernet import Fernet
import time
import os
from datetime import datetime

from intro import showBrand
import selectImage as si
from PIL import Image

escSeqForMessage = "001000000101101101000101010011100100010001011101"
escSeqForPassword = "01011011011110110101111001011101"
passwordFlag = 0
wrongCount = 0

```

```
fileExp = ""
passkey = ""
passToDecrypt = ""

def writeTextToFile(text):
    global imgName, fileExp
    fileName = f"{os.path.expanduser('~')}/{datetime.now()}.txt"
    fileExp = fileName
    openFile = open(fileName, 'w')
    openFile.write(text)

def refineText(text):
    global passkey
    refinedText = text.replace(f"{passkey}b", "")
    refinedText = refinedText.replace("' [END]", "")
    return refinedText

def hold(text):
    checkToExit = input("\nEnter 'q!' to exit without saving and 'w!' to
save and exit: ")
    if checkToExit == 'q!':
        return
    elif checkToExit == 'w!':
        writeTextToFile(text)
        si.clear()
        print(f"Your message is saved in {fileExp}")
        holding = input("Enter to continue...")
    else:
        hold(text)

def display(text):
    global passToDecrypt
    fernet = Fernet(passToDecrypt)
    text = fernet.decrypt(bytes(text, 'ascii'))
    text = str(text)
    remove = text[:2]
    text = text.replace(remove, "")
    remove = text[-1]
    text = text.replace(remove, "")
    return text

def checkPassword():
    global escSeqForPassword, imgName, passwordFlag, wrongCount, passkey,
passToDecrypt
    password = input("Decryption Key (to goto menu: [MAIN]): ")
    si.clear()

    if password == '[MAIN]':
        returnFlag = '[MAIN]'
        return returnFlag
    else:
        remain = 32 - len(password)
```



```
password = password + "0" * remain
passByte = bytes(password, 'ascii')
hashPass = hashlib.md5(passByte)
basePass = base64.b64encode(passByte)
password = str(hashPass.digest()) + '[{}^]'

passkey = password
passToDecrypt = basePass
encodedPassword = ""

passwordFromImage = ""
for char in password:
    acsiiOfChar = ord(char)
    binOfChar = f"{acsiiOfChar:08b}"
    encodedPassword = encodedPassword + binOfChar

img = Image.open(imgName)
for y in range(img.size[1]):
    if len(passwordFromImage) >= 32:
        if passwordFromImage[-32:] == escSeqForPassword:
            break
    for x in range(img.size[0]):
        r, g, b = img.getpixel((x, y))
        binR = f'{r:08b}'
        passwordFromImage = passwordFromImage + binR[6:]
    if len(passwordFromImage) >= 32:
        if passwordFromImage[-32:] == escSeqForPassword:
            break

if passwordFromImage == encodedPassword:
    return '[YES]'
else:
    return "[NO]"

def show(imageName):
    global escSeqForMessage, passwordFlag, wrongCount, imgName
    imgName = imageName
    passwordFlag = 0
    wrongCount = 0
    si.clear()

    status = checkPassword()
    if status == '[YES]':
        print(f"Selecting image {imageName}\n")
        binText = ""
        img = Image.open(imageName)

        for y in range(img.size[1]):
```

```
        if len(binText) >= 48:
            if binText[-48:] == escSeqForMessage:
                break
        for x in range(img.size[0]):
            r, g, b = img.getpixel((x, y))
            binR = f"{r:08b}"
            binText = binText + binR[6:]
            if len(binText) >= 48:
                if binText[-48:] == escSeqForMessage:
                    break

    startText = 0
    endText = startText + 8
    decodedMessage = ""

    loopBinText = len(binText) / 8
    loopCount = 0
    while loopCount < loopBinText:
        slicedBits = binText[startText: endText]
        startText = startText + 8
        endText = startText + 8
        desSlicedBits = int(str(slicedBits), 2)
        decodedMessage = decodedMessage + chr(desSlicedBits)
        loopCount = loopCount + 1

    print("Message extracted successfully...")
    time.sleep(1)
    si.clear()
    os.chdir(f"{os.path.dirname(__file__)}")
    showBrand()
    print("\nMessage::")
    refinedText = refineText(decodedMessage)
    decryptedText = display(refinedText)
    print(decryptedText)
    hold(decryptedText)

    elif status == '[NO]':
        input("\nYou entered wrong password...")
    else:
        print("\nUnknown error occurred...")

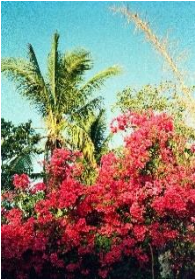

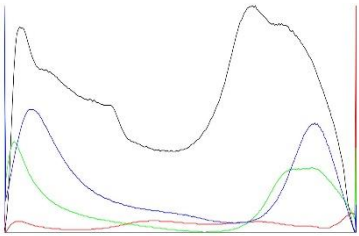
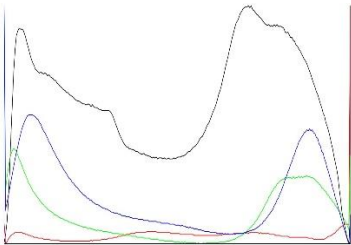


message = ""
imgName = ""
```

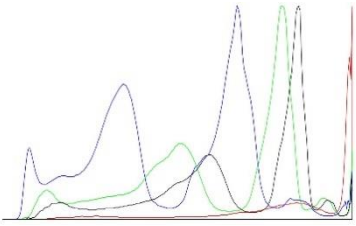
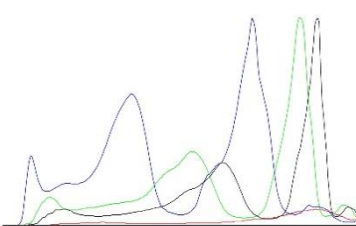


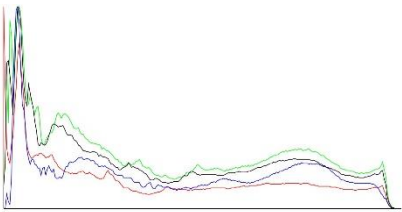
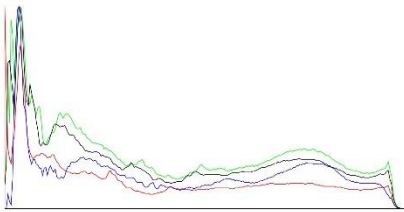


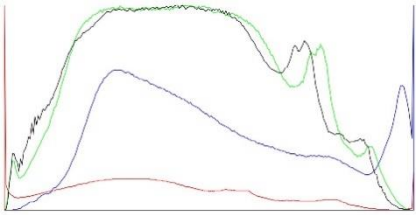
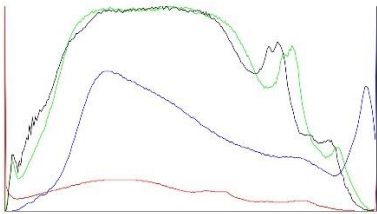
Chapter 10



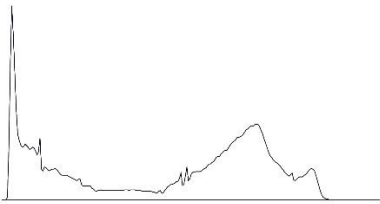
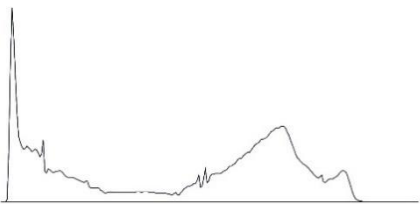


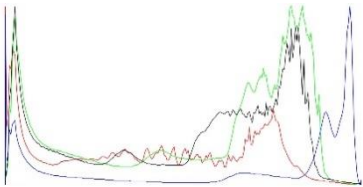
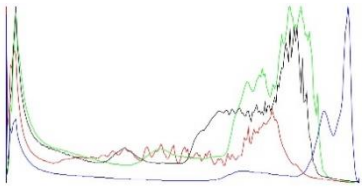


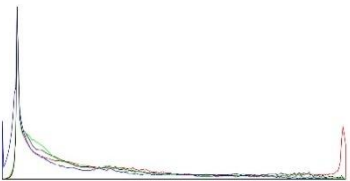
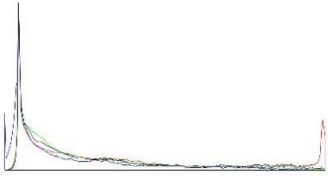
RESULTS



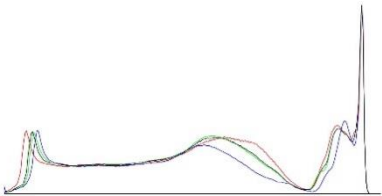
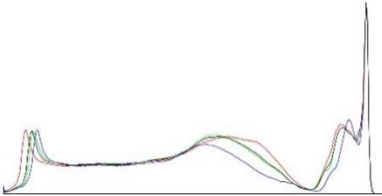
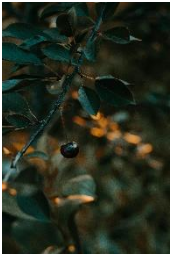
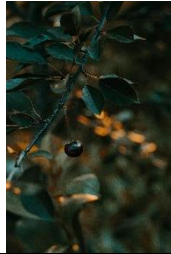
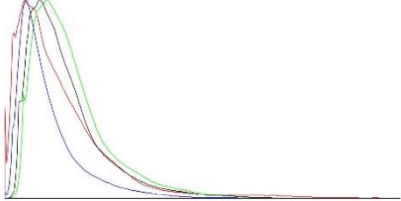
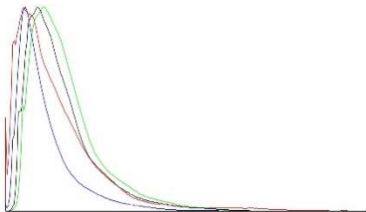


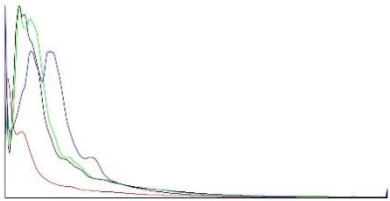
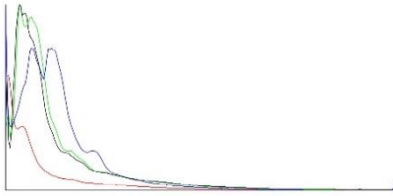
PSNR-Value represents the Peak Signal to Noise Ratio which is the peak-error comparison between two images. It is used to show the efficiency of our tool. Also to mention that using an original Image of low quality will result in a low PSNR value it is already low quality. The image we used are of 3341 X 5011 resolution with 72 Dpi and 24-bit depth. The message encoded is “Hello World” and the password used is “test123”.

Table 10.1: Comparison of Source and Stego-Image

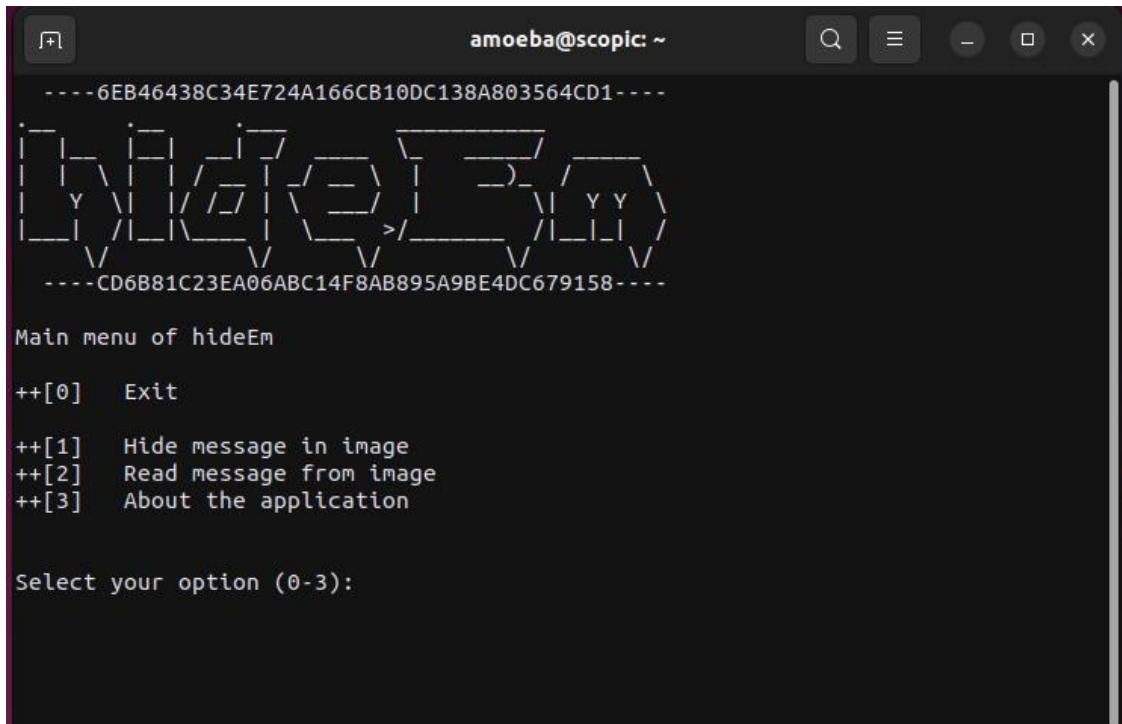
S.N .	Source Image & their histogram	Stego-Image & their histogram	PSNR -Value
1.			100.02
			
2.			102.75

			
3.			99.30
			
4.			95.55
			

5.			102.73
			
6.			104.80
			
7.			103.1
			

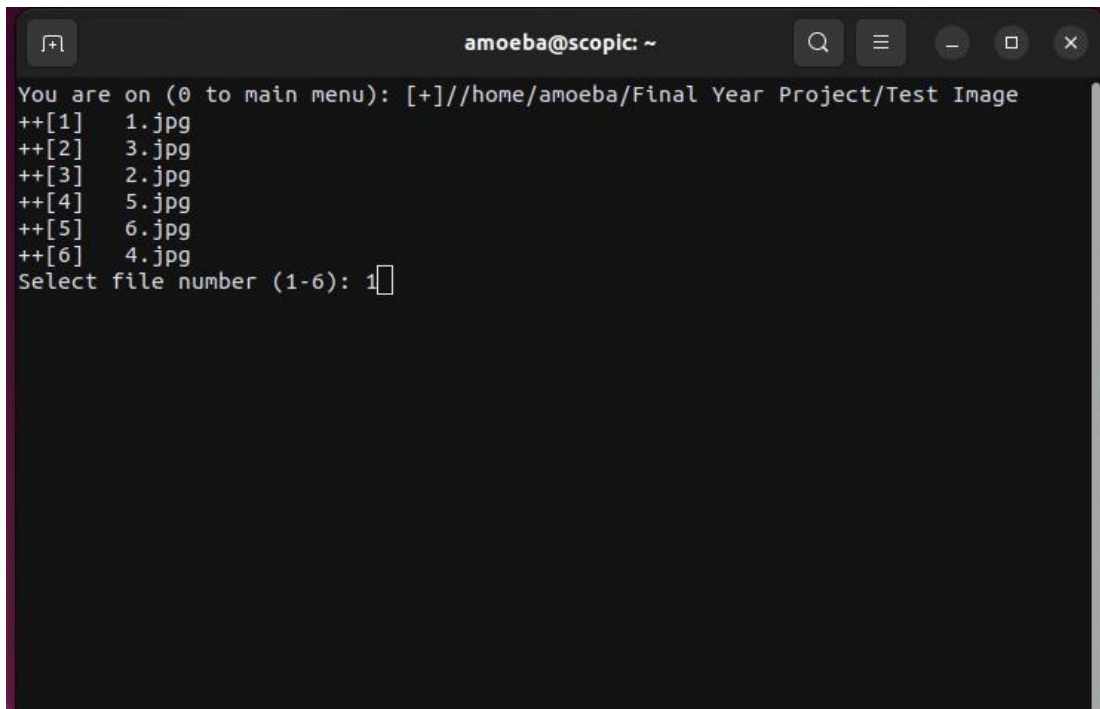
8.			99.21
			
9.			105.3
			
10.			101.78
			

SNAPSHOTS



```
amoeba@scopic: ~  
----6EB46438C34E724A166CB10DC138A803564CD1----  
[ASCII art logo]  
----CD6B81C23EA06ABC14F8AB895A9BE4DC679158----  
  
Main menu of hideEm  
  
++[0]   Exit  
++[1]   Hide message in image  
++[2]   Read message from image  
++[3]   About the application  
  
Select your option (0-3):
```

Figure 10.1: Main Menu



```
amoeba@scopic: ~  
You are on (0 to main menu): [+]//home/amoeba/Final Year Project/Test Image  
++[1]   1.jpg  
++[2]   3.jpg  
++[3]   2.jpg  
++[4]   5.jpg  
++[5]   6.jpg  
++[6]   4.jpg  
Select file number (1-6): 1
```

Figure 10.2: Image Menu

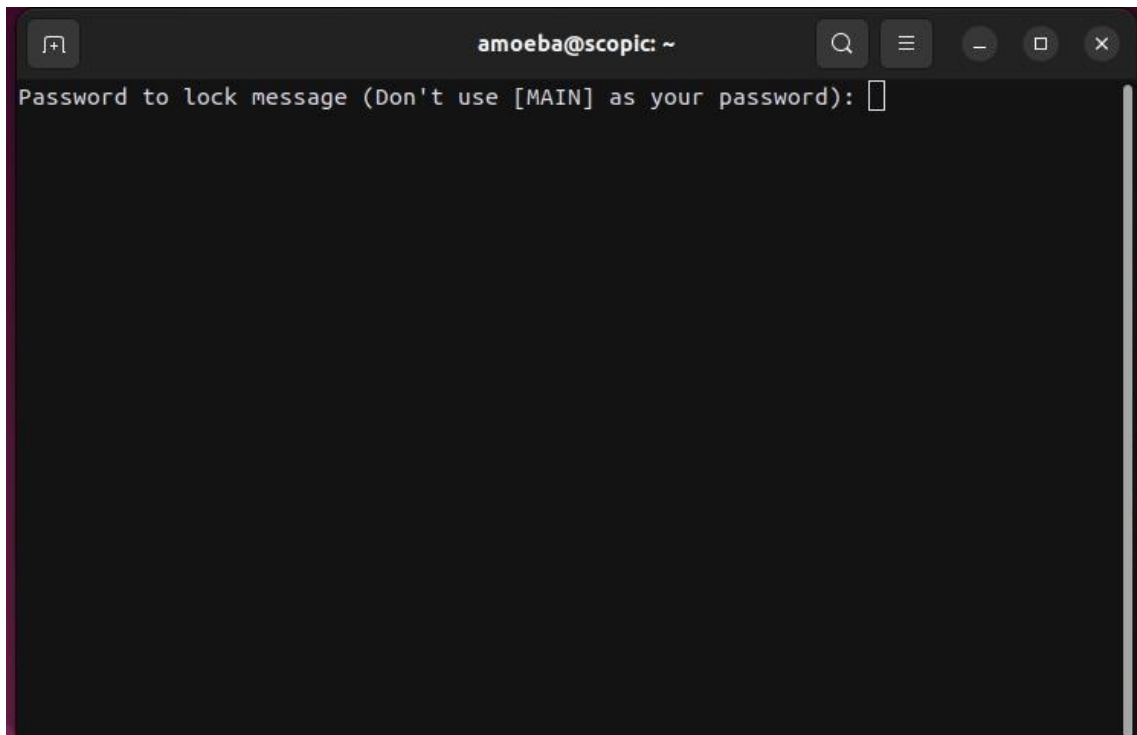


Figure 10.3: Encryption



Figure 10.4: Original Image

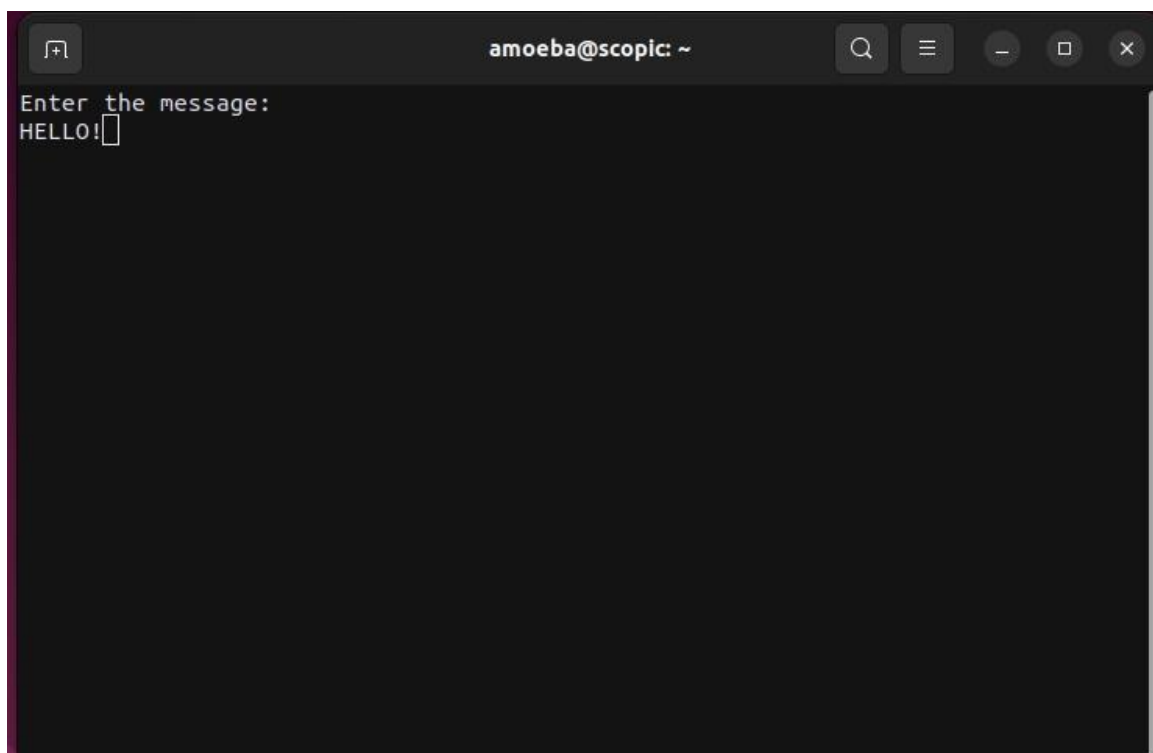


Figure 10.5: Message

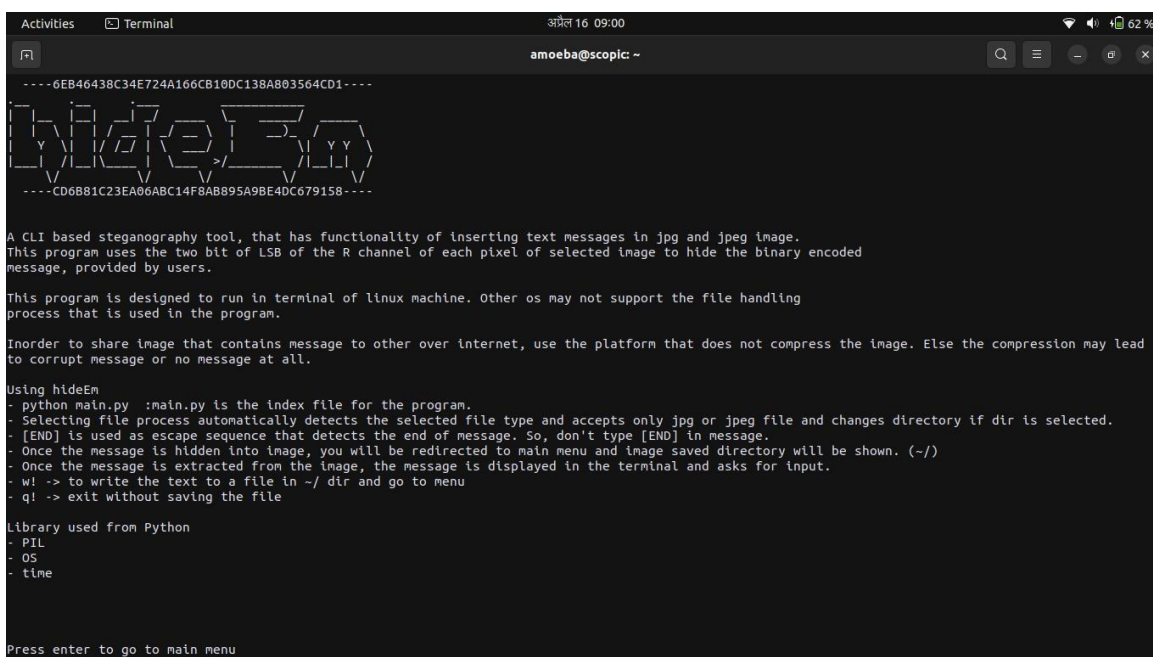


Figure 10.6: About Menu

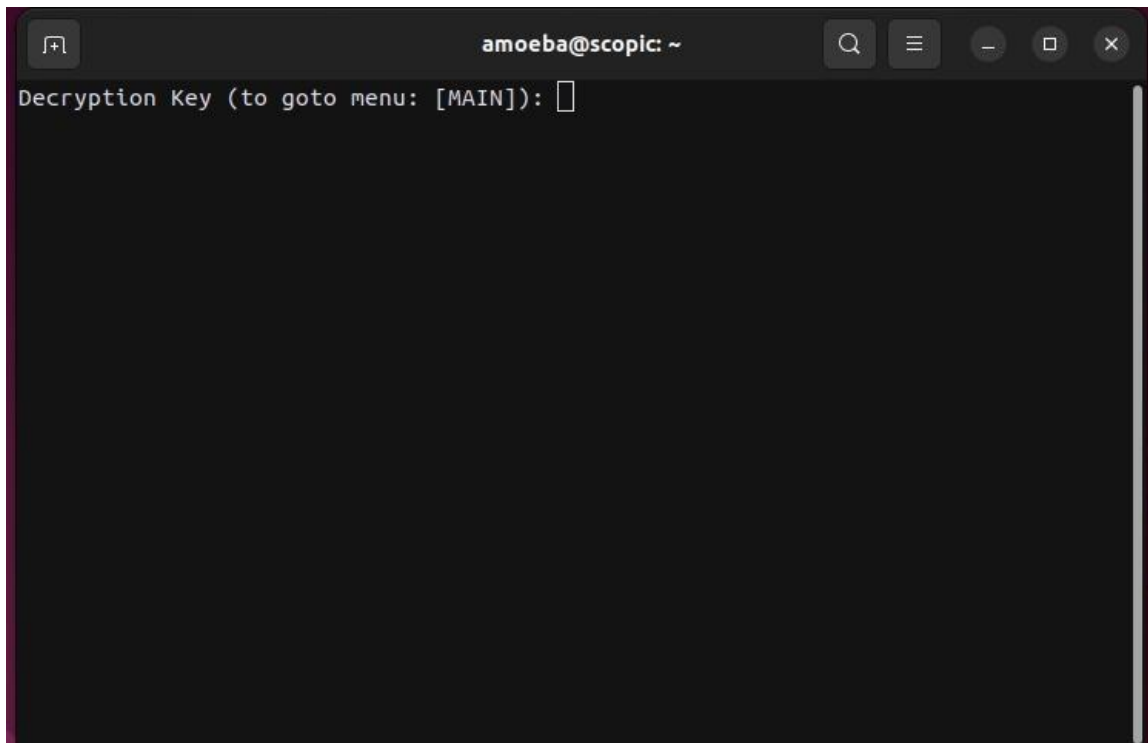


Figure 10.7: Decryption

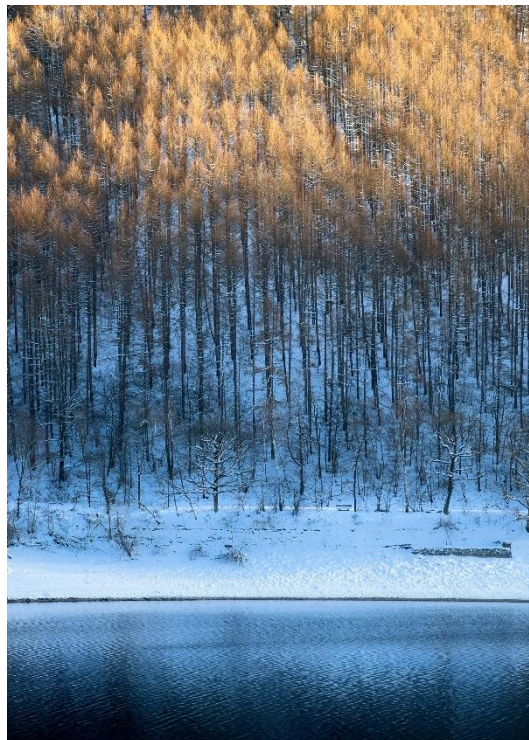


Figure 10.8: Stego-Image

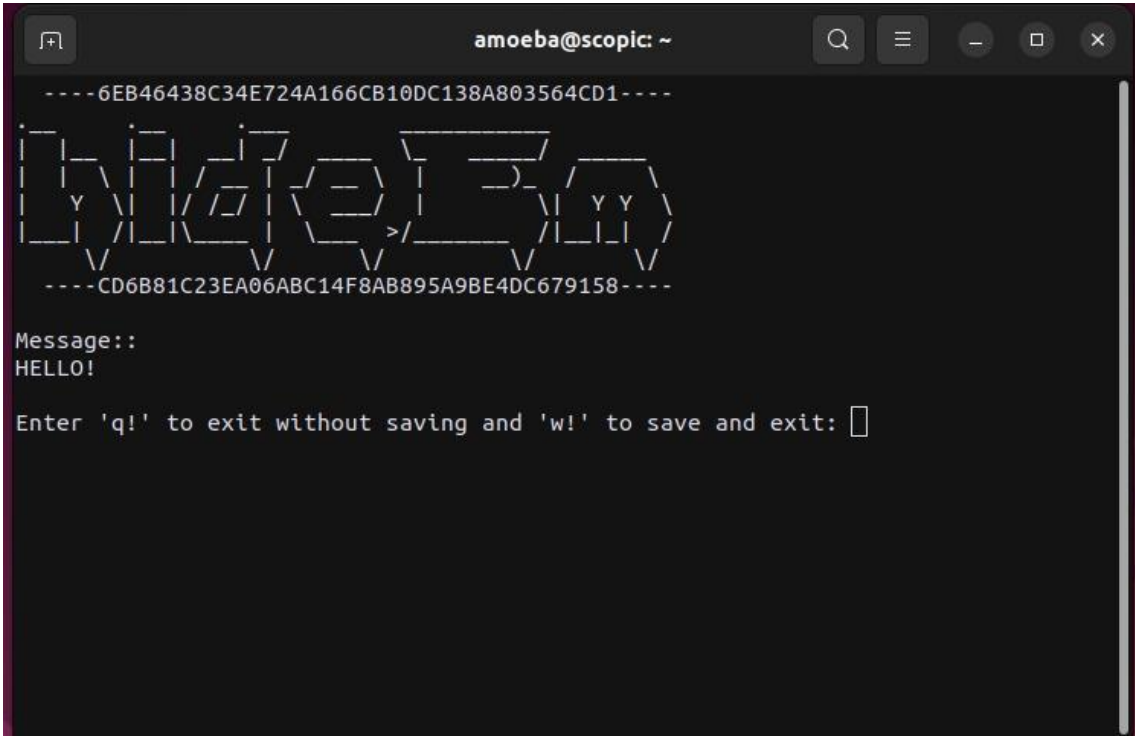


Figure 10.9: Message Decrypted

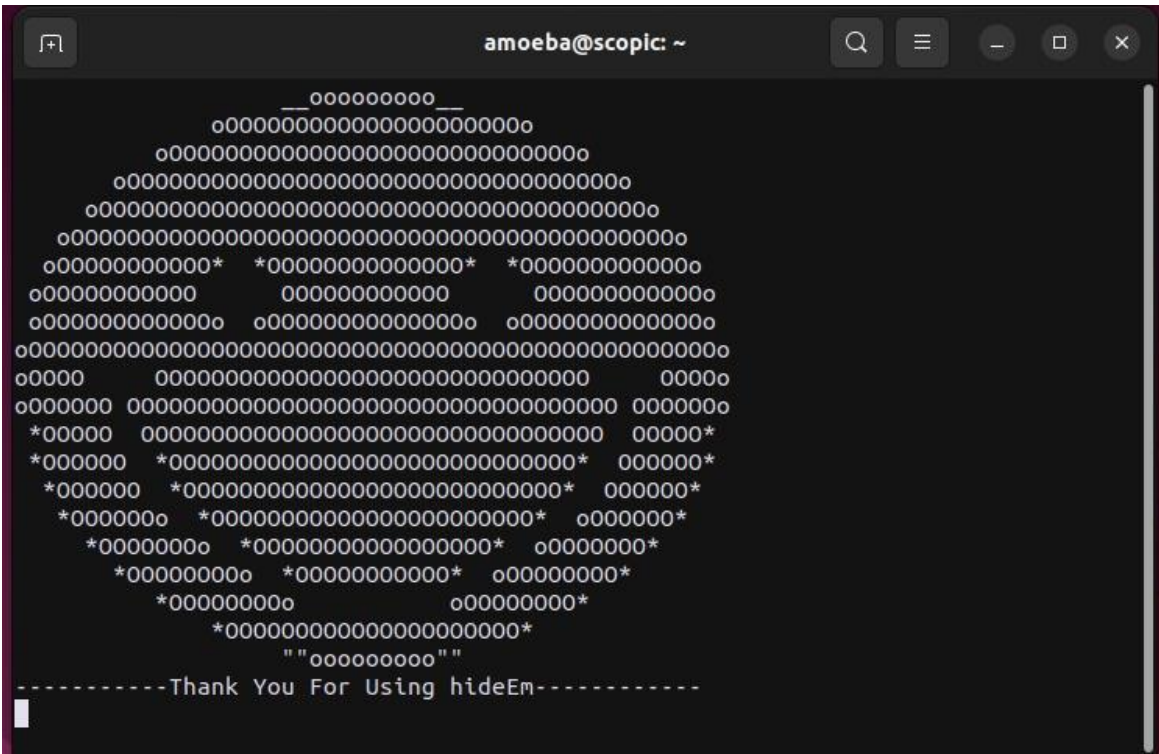


Figure 10.10: Exit Menu

Chapter 11

CONCLUSION AND FUTURE WORK

11.1 Conclusion

In this era of technological advancement, “hideEm” emerges as a valuable asset, facilitating the secure sharing of confidential data among users. Particularly beneficial for organizations and individuals unable to invest in robust security measures like GRE or IPSec, hideEm offers a practical solution to safeguard sensitive information when transmitting over public networks. However, the program's effectiveness ultimately depends on user diligence, with factors such as encryption key length and complexity directly impacting its security. While hideEm currently offers essential steganography features, ongoing updates, and enhancements will ensure it remains aligned with evolving user needs and security standards, reinforcing its position as a reliable tool for protecting sensitive data in the digital age.

The project entitled “hideEm” represents a robust and efficient solution for image steganography, offering users a reliable method to conceal sensitive data within image files securely. Through its modular architecture and seamless integration of encryption algorithms, users can encrypt messages and hide them within images, ensuring confidentiality and protection against unauthorized access. The program's intuitive interface and user-friendly design make it accessible to a wide range of users, while its comprehensive testing ensures reliability and performance. hideEm's ability to effectively hide data within images, coupled with its encryption capabilities, makes it a valuable tool for protecting sensitive information in various applications. As technology continues to advance, hideEm stands as a testament to the ongoing innovation in data security and privacy.

11.2 Future Work

For the long-term improvement of the project, several recommendations are proposed:

- Conduct regular usability testing sessions to gather user feedback and insights for further improving the user interface.

- Explore emerging encryption algorithms and protocols to assess their suitability for integration into the project, focusing on enhancing data security.
- Enhance cross-platform compatibility by optimizing the program's codebase and addressing dependencies that may hinder its performance on different systems.
- Implement robust encryption mechanisms for exported messages, such as password protection or encryption keys, to prevent unauthorized access to sensitive data.
- Develop comprehensive documentation and tutorials to assist users in effectively utilizing the program's features and understanding its security protocols.
- Establish a dedicated support system or community forum where users can seek assistance, report issues, and share insights for ongoing improvement.
- Collaborate with cybersecurity experts and professionals to conduct security audits and penetration testing to identify and address potential vulnerabilities.
- Continuously monitor and track industry trends and regulatory requirements related to data security and privacy to ensure compliance and alignment with best practices.
- Foster an open-source community around the project to encourage collaboration, contributions, and peer review, enhancing its robustness and reliability over time.
- Consider implementing additional features or functionalities based on user demands and evolving security needs, such as support for different file formats or integration with other security tools.
- Address compatibility issues with the command prompt to ensure seamless functionality across different platforms and environments.
- Implement measures to safeguard exported messages, enhancing overall data protection and confidentiality.

REFERENCES

- [1] Shahid Rahman, Jamal Uddin, Habib Ullah Khan, Hameed Hussain, Ayaz Ali khan, Muhammad Zakarya (24 November 2022). A Novel Steganography Technique for Digital Images Using the Least Significant Bit Substitution Method. IEEE Journal, 101109.
- [2] Prof. Mohamad K. Abu Zalata, Dr. Mohamad T. Barakat, Prof. Ziad A. Alqadi (January 2022). Carrier Image Rearrangement to Enhance the Security Level of LSB Method of Data Steganography. International Journal of Computer Science and Mobile Computing (IJCSMC), Volume 11, 1047760.
- [3] Mohammed J. Bawanwh, Emad Fawzi Al-Shalabi, Obaida M. Al-Hazaimah (20 January 2022). A Novel RGB Image Steganography Using Simulated Annealing and LCG via LSB. International Journal of Computer Science and Network Security (IICSNS), Volume 21, 1022937.
- [4] Temitayo Ejidokun, Olusegun O. Omitola, Kehinde Adeniji, Ifeoma Nnamah (25 January 2022). Implementation and Comparative Analysis of Variet of LSB Steganographic Method. SAUPEC Conference (2022), 101109
- [5] Sonal Karade, Prof. Savita Chouhan (December 2021). A Visual Cryptography Based Data Hiding Technique for Secret Data Encryption and Decryption. International Journal of Engineering Associates (IJEa), Volume 10, 2320-0804.