IMPLEMENTING TEXTMESSAGE CONCEALMENT THROUGH DIGITALWATERMARKING

A MINI PROJECT REPORT

Submitted by

RANCHENI S R (312420104128)

SUSHMITHA C N (312420104167)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

St. JOSEPH'S INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

ANNA UNIVERSITY :: CHENNAI 600 025

MARCH 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"IMPLEMENTING TEXT MESSAGE CONCEALMENT THROUGH DIGITAL WATERMARKING"** is the bonafide work of **"RANCHENI S R (312420104128) and SUSHMITHA C N (312420104167)"** who carried out the project under my supervision.

SIGNATURE

Dr. J. DAFNI ROSE M.E., Ph.D.,

PROFESSOR & HEAD,

Computer Science and Engineering,

St. Joseph's Institute of Technology,

Old Mamallapuram Road,

Chennai - 600 119.

SIGNATURE

Ms. A.R. DARSHIKA KELIN M.E.,

SUPERVISOR,

Assistant Professor,

Computer Science and Engineering,

St. Joseph's Institute of Technology,

Old Mamallapuram Road,

Chennai - 600 119.

ACKNOWLEDGEMENT

We also take this opportunity to thank our respected and honorable Chairman **Dr. B. Babu Manoharan M.A., M.B.A., Ph.D.,** for the guidance he offered during our tenure in this institution.

We extend our heartfelt gratitude to our respected and honorable Managing Director **Mrs. S. Jessie Priya M.Com.,** for providing us with the required resources to carry out thisproject.

We express our deep gratitude to our honorable Executive Director **Mr. B. Sashi Sekar M.Sc.,** for the constant guidance and support for our project.

We are indebted to our Principal **Dr. P. Ravichandran M.Tech.,Ph.D.,** for granting us permission to undertake this project.

We would like to express our earnest gratitude to our Head of the Department **Dr. J. Dafni Rose M.E., Ph.D.,** for her commendable support and encouragement for the completion of the project with perfection.

We also take the opportunity to express our profound gratitude to our guide **Ms. A.R. Darshika Kelin M.E.,** for his guidance, constant Support, immense help and valuable advice for the completion of this project.

We wish to convey our sincere thanks to all the teaching and non- teaching staff of the department of **COMPUTER SCIENCE AND ENGINEERING** without whose cooperation this venture would not have been a success.

CERTIFICATE OF EVALUATION

College Name : St. JOSEPH'S INSTITUTE OF TECHNOLOGY

Branch : COMPUTER SCIENCE AND ENGINEERING

Semester : VI

Sl.No	Name of the Students	Title of the Project	Name of the Supervisor with designation
1	RANCHENI S R (312420104128)	Implementing Text Message Concealment Through Digital Watermarking	Ms. A.R. DARSHIKA KELIN M.E., Assistant Professor
2	SUSHMITHA C N (312420104167)		

The report of the project work submitted by the above students in partial fulfilment for the award of Bachelor of Engineering Degree in Computer Science and Engineering of Anna University were evaluated and confirmed to be report of the work done by above students.

Submitted for project review and viva voce exam held on _____

(INTERNAL EXAMINER) **(EXTERNAL EXAMINER)**

ABSTRACT

“Steganography” is the art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message. The word "Steganography" is of Greek origin and means "covered, or hidden writing". An encrypted file may still hide information using Steganography, so even if the encrypted file is deciphered, the hidden message is not seen. Steganography used in electronic communication include steganographic coding inside of a transport layer, such as an image file, or a protocol, such as UDP. The advantage of steganography over cryptography alone is that messages do not attract attention to themselves, to messengers, or to recipients. A steganographic message (the plaintext) is often first encrypted by some traditional means, and then a hiddentext is modified in some way to contain the encrypted message (ciphertext), resulting in stegotext... In this project, we are using Tkinter and PIL modules. The actual process of our project is that we will have a cover image file and the message. In that, we will embed each bit of secret text. This process will be continued until the last bit of secret text is revealed. After this step, the data is hidden under the image. The image file will be then sent to our client, who will reverse the process to recover the original text from the image.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the internet.

One of the reasons that intruders can be successful is the most of the information they acquire from a system is in a form that they can read and comprehend. Intruders may reveal the information to others, modify it to misrepresent an individual or organization, or use it to launch an attack. One solution to this problem is, through the use of steganography. Steganography is a technique of hiding information in digital media. In contrast to cryptography, it is not to keep others from knowing the hidden information but it is to keep others from thinking that the information even exists.

Steganography become more important as more people join the cyberspace revolution. Steganography is the art of concealing information in ways that prevents the detection of hidden messages. Steganography include an array of secret communication methods that hide the message from being seen or discovered.

1.1. OVERVIEW

In this project, we propose to develop a system to hiding data by using "STEGANOGRAPHY" technique as I used many methods stands on some techniques to have at the back-end a software for hiding data based on hiding algorithms. After studying the data hiding algorithms we found many ways to hiding data by using the multimedia files and the main question for me was "Where hidden data hides?" as we found by our search to know where the data hides it's important to know what is the file type of the data that it shall be hidden and the cover file type so it is possible to alter graphic or sound files slightly without losing their overall viability for the viewer and listener. With audio, you

can use bits of file that contain sound not audible to the human ear. With graphic images, you can remove redundant bits of color from the image and still produce a picture that looks intact to human eye and is difficult to discern from its original. It is in those bits that stego hides its data. By the final of our research we developed a software uses an algorithm, to embed data in an image; The purposed system is called "Steganography", the aim of this project is to encrypt the data; the meaning of encrypt is to hide the data over an image using different steganographic algorithms, in this system DCT and AES is the algorithms that we use to hiding the data.

1.2 PROBLEM STATEMENT

This project aims to solve the issue of transmitting data securely over the internet using steganography. This technique involves hiding information in carrier files such as images, audio files, and videos to transmit a message without arousing suspicion. The proposed solution focuses on image steganography and utilizes specialized algorithms and techniques to embed a digital text of a secret message within an image file. The system provides a secure and reliable method of communication that is resistant to interception and eavesdropping. The embedded message is not readily visible, and the image appears unchanged to anyone who does not possess the key to extract the hidden information. The use of steganography allows for secure communication without raising alarms or drawing unwanted attention. The proposed image steganography system employs advanced encryption techniques to ensure that the hidden information remains secure and protected from unauthorized access. The system also includes measures to verify the authenticity of the image and the embedded message, preventing tampering or forgery. Overall, the project proposes a practical and effective solution for secure communication over the internet using steganography. It offers a secure and reliable method of transmitting data without the need for complex encryption methods that can attract unwanted attention.

1.3 EXISTING SYSTEM

Steganography is a system that hides information in an application cover carrier like image, text, audio, and video. Considerable amount of work has been carried out by different researchers on this subject. Least Significant Bit (LSB) insertion method was more suspicious and low robustness against attacks. The objectives of this study were to analyse various existing system and implement a dynamic substitution based Image Steganography (IS) with a secret key. In the existing system reversible data hiding technique the image is compressed and encrypted by using the encryption key and the data to hide is embedded in to the image by using the same encryption key. The user who knows the secret encryption key used can access the image and decrypt it after extracting or removing the data hidden in the image. After extracting the data hidden in the image then only can be the original image is retrieved.

1.3.1 DISADVANTAGES OF EXISTING SYSTEM

- Low security
- The accuracy of prediction is less.
- Anybody with an pen source code can decode.

1.4 PROPOSED SYSTEM

In this project, we present an idea for hiding data behind a file. This project proposes a highly secured data hiding technique in the spatial domain of image steganography. The proposed scheme takes the takes the message bit and performs the AES Encryption and it takes the source image and performs the DCT transformation and then produced output is embedded within the image. The embedding procedure is done in a way that there will be no sign of message hidden inside the cover object.

1.4.1 APPLICATIONS

a. SecretMessage+

SecretMessage+ is a mobile application that allows users to send and receive hidden text messages using digital watermarking. The app works by embedding the message into an image, which can then be shared with the intended recipient without arousing suspicion.

b. Watermark Messenger

Watermark Messenger is a messaging app that uses digital watermarking to conceal text messages within images. The app allows users to send and receive secret messages without arousing suspicion, by embedding them into everyday images such as memes, landscapes, or animal photos.

c. StegoChat

StegoChat is a secure messaging app that allows users to send text messages with hidden messages concealed within images using digital watermarking. The app is designed to provide a high level of security and privacy for its users by encrypting messages, embedding them into images, and securely transmitting them. Encryption: The app uses a strong encryption algorithm to protect the message from being read by unauthorized parties. Decryption: The recipient can then use StegoChat to extract the hidden message from the image and read it.

d. SecureNotes

SecureNotes is a note-taking app that allows users to hide secret messages within their notes using digital watermarking. The app is designed to provide an extra layer of security for users who need to keep their notes confidential and protect them from prying eyes. Encryption: The app uses a strong encryption algorithm to protect the message from being read by unauthorized parties.

Decryption: The user can then use SecureNotes to extract the hidden message from the image and read it.

CHAPTER 2

LITERATURE SURVEY

Akhtar, et.al [2013][1] implemented a variation of plain LSB (Least Significant Bit) algorithm. The stego-image quality has been improved by using bit-inversion technique. LSB method improving the PSNR of stegoimage. Through storing the bit patterns for which LSBs are inverted, image may be obtained correctly. For the improving the robustness of steganography, RC4 algorithm had been implemented to achieve the randomization in hiding message image bits into cover image pixels instead of storing them sequentially. This method randomly disperses the bits of the message in the cover image and thus, harder for unauthorized people to extract the original message. The presented method shows good enhancement to Least Significant Bit technique in consideration to security as well as image quality.

Andrew D. Ker [2010][2] Detecting LSB matching steganography is quiet difficult compared to the LSB replacement steganography. In this paper Histogram characteristic function (HCF) is used for the detection of steganography in color images, but it cannot be used for gray scale images.

Bingwen Feng et.al [2015][3] purposed a state-of-the-art approach of binary image steganography. This technique is proposed to minimize the distortion on the texture. In this method of steganography firstly the rotation, complement and mirroring invariant texture patterns are extracted from the binary image. They also proposed a measurement and based on this proposed measurement this approach is practically implemented. Practical results show that proposed steganographic approach has high statistical security with high stego image quality and high embedding capacity.

C.-C.Chang et al. [2007][4] this paper proposes a novel steganographic technique, which modifies the pixel values. This method does not replace the LSBs of pixel value directly, but changes the pixel value into another similar value. In a word, this steganographic method provides a large embedding capacity

with little perceptual distortion.

Da-Chun Wu et al. [2013][5] proposed a differencing steganographic method that uses the difference between two consecutive pixels in the same block to determine the number of secret bits to be stuffed. In this method a range table is used which ranges from 0-255. The difference value is subsequently adjusted to the difference in the same range to embed the secret bits, and the difference between the original difference value and the new one is shared between the two pixels. Extraction scheme in this method is quite simple and it do not require cover image.

Das, R. et.al [2012][6] proposed a novel technique for image steganography based on Huffman Encoding. Two 8 bit gray level image of size M X N and P X Q are used as cover image and secret imagerespectively. Huffman Encoding is performed over the secret image/message before embedding and each bit of Huffman code of secret image/message is embedded inside the cover image by altering the least significant bit (LSB) of each of the pixel's intensities of cover image. The size of Huffman encoded bit stream and Huffman Table are also Parmar Ajit Kumar Maganbhai et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (1) , 2015, 685-688 www.ijcsit.com 686 embedded inside the cover image, in order that the StegoImage becomes standalone information to the receiver. Results show that the algorithm has a high capacity and agood invisibility. Moreover Peak Signal to Noise Ratio (PSNR) of stego image with cover image shows better result in comparison with other existing steganography approaches. The satisfactory security is maintained in this research.

Della Baby et al. [2015][7] proposed a “Novel DWT based Image Securing method using Steganography”. In their work new steganography technique is prosed in which multiple RGB images are embedded into single RGB image using DWT steganographic technique. The cover image is divided into 3 colors i.e. Red, Green and Blue color space. These three color spaces are utilized to hide secret

information. Experimental results obtained using this system has good robustness. Value of PSNR and SSIM index have been used by authors to compare the quality of stego image and original cover image. Proposed method has good level of PSNR and SSIM index values. Authors have found that their experimental results are better than existing approaches and have increased embedding capacity because of data compression. So overall security of their approach is high with less perceptible changes in stego image.

E. Dagar and S. Dagar [2013][8] present the steganography technique for color RGB images to improve the security level of data transfer through the internet. 24-bit RGB image is utilized as cover image to embed secret data in red, green and blue pixels. X-Box mapping is used and several boxes contain 16 different values. Here "X" represents any integer number from 0 to 9. After these values saved in X-Boxes are mapped with LSBs of carrier image. It is very difficult for the attacker to extract the secret information because they make use of mapping. Thus, this mapping provides high level of security to hidden information. PSNR value is also calculated and it has high PSNR value which leads to greater stego image quality.

G. Prashanti et.al [2015][9] have done survey on recent achievements of LSB based image steganography. In this survey authors discuss the improvements that enhance the steganographic results such as high robustness, high embedding capacity and un-detectability of hidden information. Along with this survey two new techniques are also proposed. First technique is used to embed data or secret messages into the cover image and in the second technique a secret gray scale image is embedded into another gray scale image. These techniques use four state table that produce pseudo random numbers. This is used for embedding the secret information. These two methods have greater security because secret information is hidden on random selected locations of LSBs of the image with the help of pseudo random numbers generated by the table.

Hemalatha et.al [2012][10] presented integer Wavelet Transform (IWT) is

used to hide the key thus it is very secure and robust because no one can realize the hidden information and it cannot be lost due to noise or any signal processing operations. Result shows very good Peak Signal to Noise Ratio, which is a measure of security. In this method the secret information is hidden in the middle bit-planes of the integer wavelet coefficients in high frequency sub-bands.

Huaiqing wang et al. [2014][11] Different techniques of steganography and steganalytic methods were discussed in detail in this paper. This paper focuses on LSB modification techniques, Masking techniques, Transformation domain techniques, Techniques incorporated in compression algorithms, and spread spectrum techniques. Then the important attributes of a steganographic system are presented, security, payload and robustness. This paper also presents various steganalytic methods such as, RS steganalysis, Chi-square test, Histogram analysis and universal blind detection.

Jessica Fridrich et al.[2012][12] This paper proposes a highly accurate steganalysis technique which can even estimate the length of secret message embedded in LSB method. In this method, the test image is divided into groups of n consecutive or disjoint pixels. This method exploits the modified pixel values to determine the content of secret message. A discriminating function is applied on the group of pixels. This discriminating function determines the regularity or smoothness of pixels. Then a permutation function called flipping is applied on the pixel groups. By using discriminating function and flipping, Pixels groups are classified in to three categories, i.e Regular groups, Singular groups and Unused Groups. For a given mask, fraction of Regular groups Rm and fraction of singular groups Sm are calculated. Presence of noise in the image causes Rm to be greater than Sm.

Johnson et.al [2002][13] This article explores the different methods of steganography such as LSB, Masking and Filtering and also explains about different software tools present in the market for Steganography, such as Stego Dos, White Noise Storm, S-tool etc.

Kazem Qazanfari et al. [2014][14] proposed an improved version of LSB++ approach. In this improved LSB++ they make distinction between sensitive pixels and allow protecting them from embedding of extra bits, which results in lower distortion in the co-occurrence matrices. They also extend this method to preserve DCT coefficients of JPEG 3 format images. This improved method results in fewer traces in the co-occurrence matrices then old LSB++ technique. This method is also secure against histogram based attacks because this method does not make any changes in the histogram and hence histograms of both cover image as well as stego image will be same. The quality of stego images is also high because of elimination of extra bit embedding.

M. Nusrati et al. [2015][15] have done study on heuristic genetic algorithm based steganographic method for hiding secret information in a cover image. This method optimally find the appropriate locations in cover image to embed the secret information by focusing on the "before embedding hiding techniques". It tries to make least changes in the bits which lead to minimal modifications in image histogram. To covert the LSBs and secret message to set of blocks, segmentation is done in this genetic algorithm. After this algorithm finds the appropriate locations for embedding, the secret blocks are embedded and it generates the key file which is used during message extraction process. Experimental results show that this genetic based method is more efficient than basic LSB algorithm with high stego image quality.

Marvel et.al [2010][16] It is proposed that (Spread Spectrum Image Steganography) SSIS is a blind scheme where the original image is not needed to extract the hidden information unless the receiver possesses the secret key to extract the secret message, otherwise it is virtually undetectable. Thus making this technique reliable and secure.

Mei-Yi Wu et al. [2012][17] this paper presents a new iterative method of image steganography based on palette which reduces the Root Mean Square error between an original image and its corresponding stego-image. Based on a palette

modification scheme, which can embed one message bit into each pixel in a palette-based image iteratively. The cost of removing an entry color in a palette and the profit of generating a new color to replace the old color are calculated. If the maximal profit exceeds the minimal cost, an entry color is replaced in iteration. On the based-on Huffman Coding, Amitava Nag et al.[2014][25] present a novel steganographic technique of LSB substitution. Their technique basically focuses on high security, larger embedding capacity and acceptable level of stego image quality. Firstly, Huffman tree is produced to encode every 8 bits of secret image. After encoding, they divide the encoded bits into four parts and have 0 to 3 decimal values. Location of embedding a message in cover image is determined by these decimal values. Experimental results show that it is very difficult for attacker to extract the secret information because Huffman table decrease the size of the cover image. Purposed techniques just have acceptable level of PSNR values and lie between 30 dB to 31 dB.

P. U. Deshmuk al. [2016][18] also present the edge adaptive steganography based on LSB substitution. They embed secret information in sharp (edges) regions of the carrier image using adaptive scheme and difference between two adjacent pixels of carrier image. Their technique performs well than other LSB and Pixel difference-based techniques and maintains the quality of stego image.

Prabakaran, G.et.al [2013][19] Investigated on Medical records are extremely sensitive patient information a multi secure and robustness of medical image based steganography scheme is proposed. This methodology provides an efficient and storage security mechanism for the protection of digital medical images. Authors proposed a viable steganography method using Integer Wavelet Transform to protect the MRI medical image into a single container image. The patient's medical diagnosis image has been taken as secret image and Arnold transform was applied and scrambled secret image was obtained. In this case, the scrambled secret image was embedded into the dummy container image and Inverse IWT was taken to get a dummy secret image. It has been observed that

the quality parameters are improved with acceptable PSNR compared to the existing algorithms.

Reddy, H.S.M. et.al [2012][20] worked on the steganography is used to hide. Secure Steganography using Hybrid Domain Technique (SSHDT). The cover image of different formats and sizes are considered and resized to dimensions of power of 2. The Daubechies Lifting Wavelet Transforms (LWT) is applied on cover image to generate four sub bands XA, XH, XV and XD. The XD band is considered and divided into two equal blocks say upper and lower for payload embedding. The payload of different formats is considered and resized to dimensions of power of 2. The payload is fragmented into four equal blocks. The Decision Factor Based Manipulation (DFBM) is used to scramble further stego object to improve security to the payload. Dubechies Inverse LWT (ILWT) is applied on XA, XH, XV and XD stego objects to obtain stego image in spatial domain. It has been observed that PSNR and embedding capacity of the proposed algorithm is better compared to the existing algorithm.

Savita Goel et al. in [2014][21] proposed a new method of embedding secret messages in cover image using LSB method using different progressions. Authors compare the quality of stego image with respect to cover image using number of image quality parameters such as Peak Signal to Noise Ratio (PSNR), Mean Square Error (MSE), histograms and CPU time, Structure Similarity (SSIM) index and Feature Similarity Index Measure (FSIM). Their study and experimental results shows that their proposed method is fast and highly efficient as compared to basic LSB methods.

Soni, A et.al [2013][22] The Fractional Fourier transform (FrFT), [1] Investigated on as a generalization of the classical Fourier transform, introduced years ago in mathematics literature. The enhanced computation of fractional Fourier transform, the discrete version of FrFT came into existence DFrFT. This study of illustrates the advantage of discrete fractional Fourier transform (DFrFT) as compared to other transforms for steganography in image processing. The

result shows same PSNR in both domain (time and frequency) but DFrFT gives an advantage of additional stego key. The order parameter of this transform.

Sorina Dumitrescu et al. [2005][23] This paper proposes a new steganalysis technique to detect LSB steganography in digital signals such as image and audio. This technique is based on statistical analysis of sample pairs. By this technique the length of hidden message embedded via LSB steganography can be estimated with high precision.

Thenmozhi et.al [2012][24] presented the novel scheme embeds data in integer wavelet transform coefficients by using a cropping function in an 8×8 block on the cover image. The optimal pixel change process has been applied after embedding the message. Authors employed the frequency domain to increase the robustness of our steganography method. Integer wavelet transform avoid the floating point precision problems of the wavelet filter. Result shows that the method outperforms adaptive steganography technique based on integer wavelet transform in terms of peak signal to noise ratio and capacity.

Tseng, Y.C et al. [2008][25] This paper presents a secure steganographic scheme which makes sure that if any modified bit in the cover image should be adjacent to another bit that has the same value as the former's new value. By this way the detection becomes extremely difficult. But for achieving this, data hiding space has to be reduced.

Xinpeng Zhang et.al [2012][26] this paper proposes the steganalysis of PVD method proposed by Wu and Tsai. This steganalysis is based on Histogram analysis. The zigzag scan of the image pixels produces a vector called 'Image Vector' and the difference of every pair of pixels in this vector produces another vector called 'Substitute vector'. An image from Substitute vector is built which is named as substitute image. Histogram of substitute image is constructed and analyzed.

CHAPTER 3

SYSTEM DESIGN

In this chapter, the various UML diagrams for the Implementing Text Message Concealment through Digital Watermarking is represented and various functionalities are explained.

3.1 UNIFIED MODELING LANGUAGE

Unified Modeling language (UML) is a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system. Thus, UML makes these artifacts scalable, secure and robust in execution. It uses graphic notation to create visual models of software systems. UML is designed to enable users to develop an expressive, ready to use visual modeling language. In addition, it supports high-level development concepts such as frameworks, patterns and collaborations. Some of the UML diagrams are discussed.

3.1.1 Use Case Diagram of Digital Watermarking

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So it can be said that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors.

Actors can be defined as something that interacts with the system. The actors can be human user,some internal applications or may be some external applications.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements.

In this diagram, there are three actors: the Watermarker, the Content, and the Decryption Key. The Watermarker is responsible for embedding a digital watermark into the Content using both the AES encryption and DCT algorithms, while the Content represents the digital media file that the watermark is being embedded into. The Decryption Key is required to decrypt the watermarked content for further use.

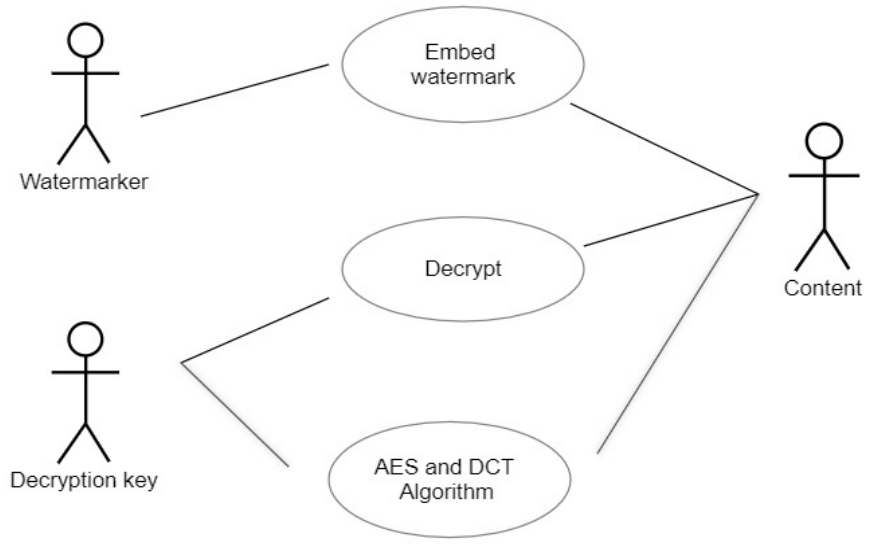

Figure 3.1 Use case diagram of Digital Watermarking

The Watermarker actor has one use case, which is to embed the watermark into the Content using both the AES encryption and DCT algorithms. This involves generating a unique watermark, encrypting it with the AES algorithm using the Decryption Key, and then using the DCT algorithm to embed the watermark into the content.

The Content actor has one use case, which is to decrypt the watermarked content using the Decryption Key and extract the watermark using the DCT algorithm. This involves using the same AES algorithm and Decryption Key that were used to embed the watermark to decrypt the content and then using the DCT algorithm to extract the watermark.

3.1.2 Class Diagram of Digital Watermarking

Figure 3.2 shows that class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system.

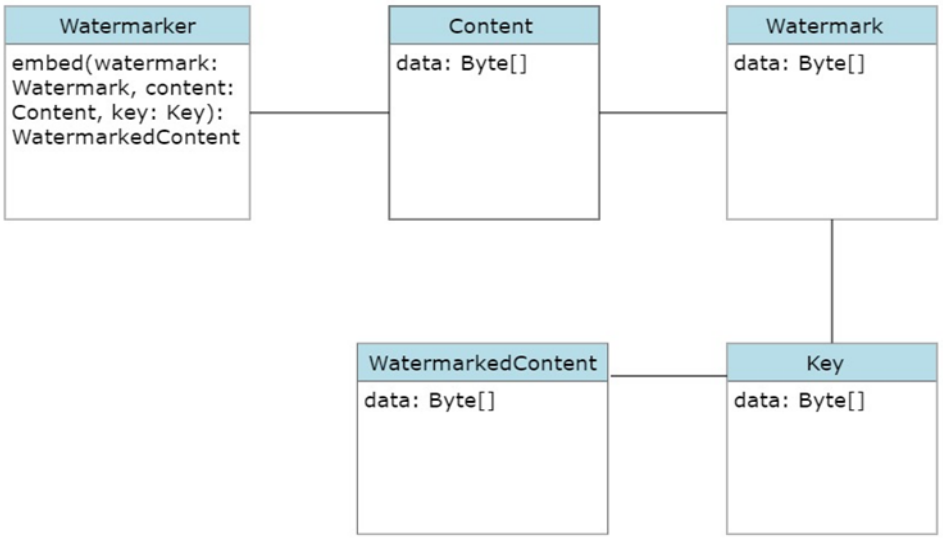

Figure 3.2 Class Diagram of Digital Watermarking

In this diagram, there are four classes: Watermarker, Content, Watermark, Key, and WatermarkedContent. The Watermarker class is responsible for embedding a digital watermark into the Content using both the AES encryption and DCT algorithms, while the Content represents the digital media file that the watermark is being embedded into. The Watermark class represents the digital watermark that is being embedded, and the Key class represents the key used for AES encryption and decryption. The WatermarkedContent class represents the watermarked content that is produced by the Watermarker.

The Watermarker class has one method, embed(), which takes a Watermark object, a Content object, and a Key object as parameters and returns a WatermarkedContent object. This method is responsible for encrypting the

watermark using the AES algorithm with the key, and then embedding it into the content using the DCT algorithm. The Content class has one property, data, which represents the raw data of the digital media file.

Overall, the class diagram for digital watermarking using AES and DCT algorithms focuses on the classes and their relationships that are involved in the process of embedding a watermark into a digital media file using both algorithms.

3.13 Sequence Diagram of Digital Watermarking

Figure 3.3 shows that UML sequence diagrams model the flow of logic within the system in a visual manner, enabling to both document and validate the logic, and are commonly used for both analysis and design purposes.

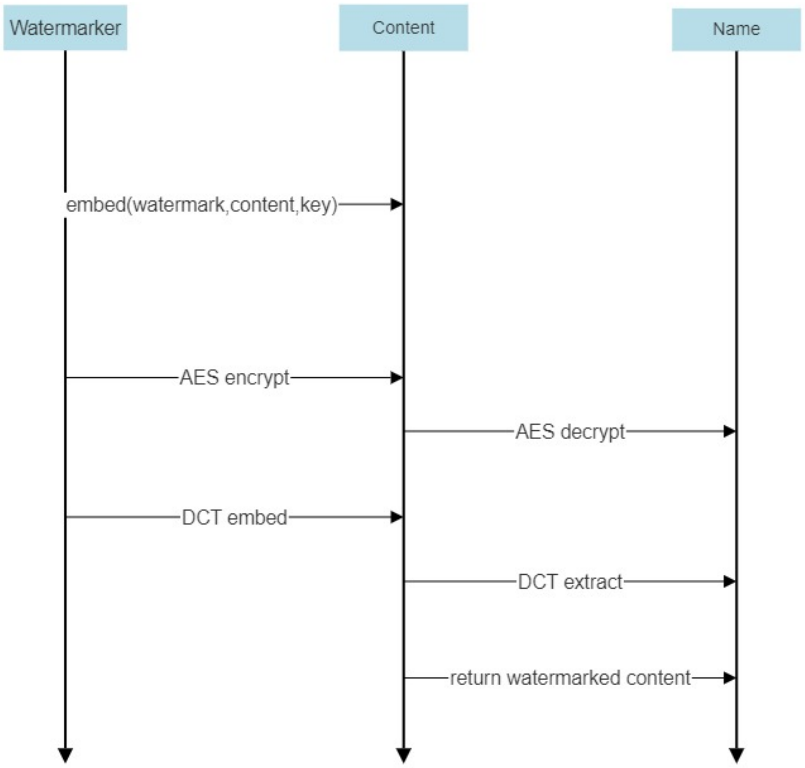

Figure 3.3 Sequence Diagram of Digital Watermarking

In this sequence diagram, there are three objects: Watermarker, Content, and Key. The Watermarker object is responsible for embedding a digital watermark into the Content using both the AES encryption and DCT algorithms, while the Content object represents the digital media file that the watermark is being

embedded into. The Key object represents the key used for AES encryption and decryption.

The Watermarker object calls the embed() method, passing in the Watermark, Content, and Key objects as parameters. The Watermarker object then uses the AES algorithm to encrypt the Watermark object with the Key object, and then uses the DCT algorithm to embed the encrypted watermark into the Content object.

The Content object receives the watermarked content from the Watermarker object, and then uses the Key object and AES algorithm to decrypt the watermarked content. It then uses the DCT algorithm to extract the embedded watermark from the decrypted content. Finally, the Watermarker object returns the watermarked content to the caller.

Overall, the sequence diagram for digital watermarking using AES and DCT algorithms focuses on the sequence of steps involved in the process of embedding a watermark into a digital media file using both algorithms, and then extracting the watermark from the watermarked content. The Watermark class also has one property, data, which represents the raw data of the digital watermark.

The Key class has one property, data, which represents the raw data of the key used for AES encryption and decryption.The WatermarkedContent class has one property, data, which represents the raw data of the watermarked content.

Overall, the class diagram for digital watermarking using AES and DCT algorithms focuses on the classes and their relationships that are involved in the process of embedding a watermark into a digital media file using both algorithms.

3.1.4 Activity Diagram Of Digital Watermarking

Figure 3.4 shows that activity is a particular operation of the system. Activity diagram is suitable for modeling the activity flow of the system.

The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to

show message flow from one activity to another.Activity is a particular operation of the system.

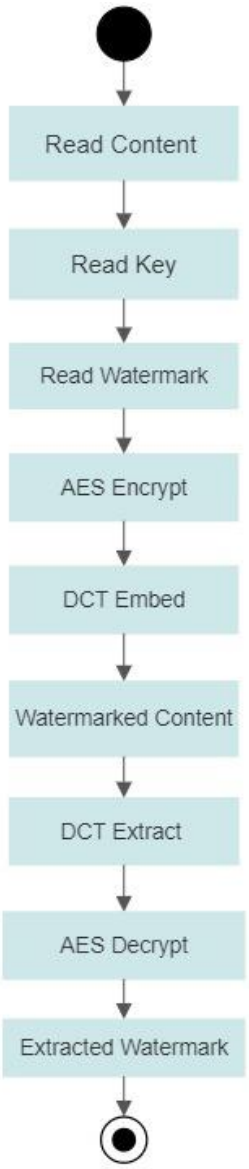

Figure 3.4 Activity Diagram of Digital Watermarking

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

In this activity diagram, there are seven actions, each represented by a rectangular shape, and one start and end point, represented by rounded rectangles.

The process starts with the Start Process action, which leads to the Read

Content action, where the digital media file to be watermarked is read. The Read Key and Read Watermark actions follow, where the encryption key and digital watermark are read, respectively.

The AES Encrypt and DCT Embed actions are then performed, where the digital watermark is encrypted using the encryption key with the AES algorithm and then embedded into the digital media file using the DCT algorithm. The result is the Watermarked Content action.

The DCT Extract and AES Decrypt actions are then performed, where the watermark is extracted from the watermarked content using the DCT algorithm and decrypted using the AES algorithm with the encryption key. The result is the Extracted Watermark action. Finally, the End Process action is performed, representing the completion of the watermarking process.

Overall, the activity diagram for digital watermarking using AES and DCT algorithms focuses on the sequence of actions involved in the process of embedding a watermark into a digital media file using both algorithms, and then extracting the watermark from the watermarked content.

CHAPTER 4
SYSTEM ARCHITECTURE

In this chapter, the System Architecture for Implementing Text Message Concealment through Digital Watermarking is represented and various functionalities are explained.

4.1 SYSTEM ARCHITECTURE DIAGRAM

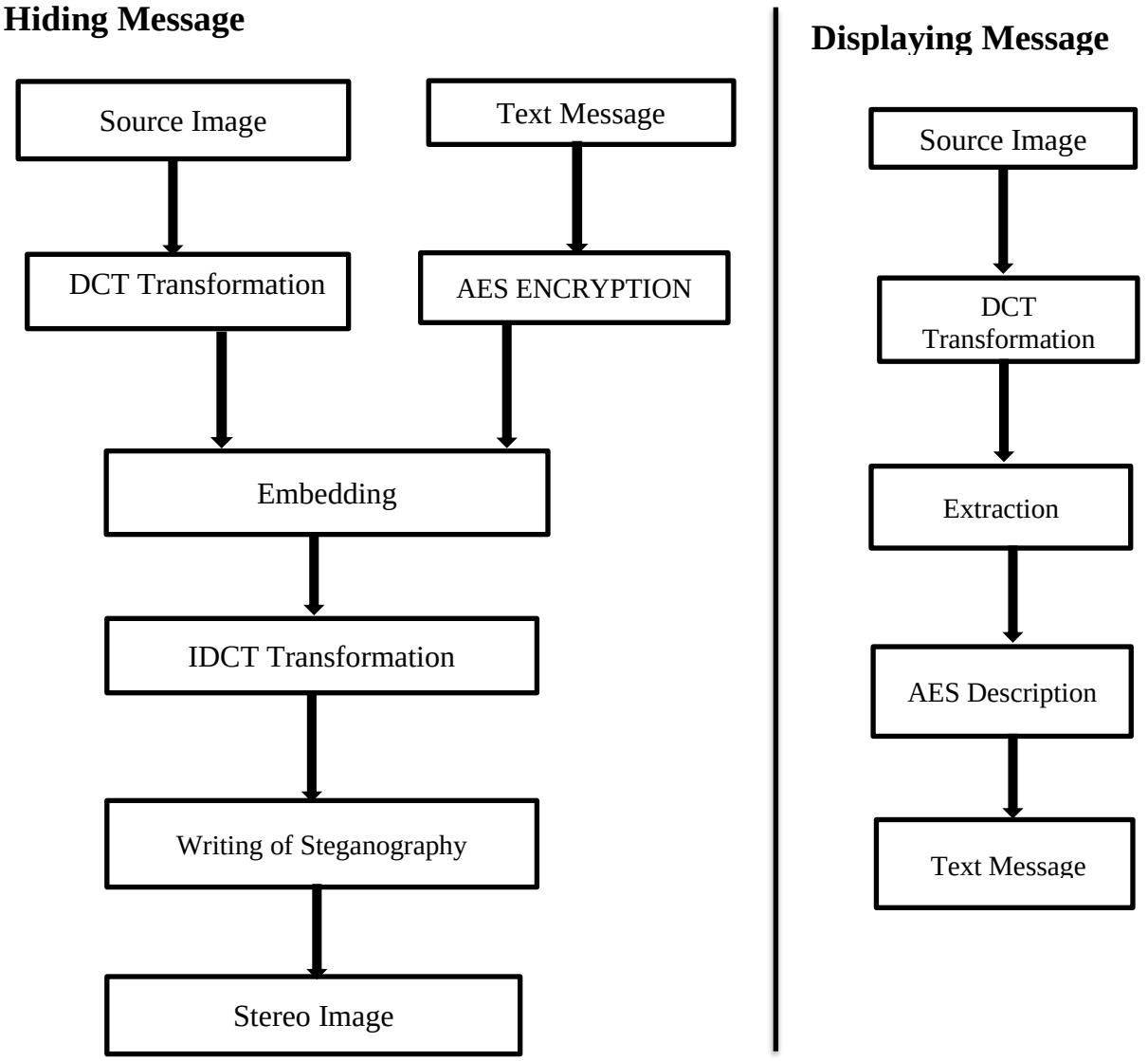

Figure 4.1 System Architecture Diagram

4.2 ARCHITECTURE DESCRIPTION

Steganography is the art of hiding messages within other innocuous looking objects, like images, audio or video files. One of the most popular methods of steganography is embedding a message within an image using Discrete Cosine Transform (DCT) and Advanced Encryption Standard (AES) algorithm. The process of hiding a message starts by converting the source image to its DCT form, while the text message is encrypted using AES. Once the DCT and encrypted message are ready, they are both entered into an embedding algorithm, where the message is hidden within the DCT coefficients of the image. After embedding, the image undergoes an inverse DCT transformation to obtain the modified image with the hidden message. Finally, the steganography is written to produce a stereo image. On the other hand, the process of displaying the message involves the reverse steps of the embedding process. First, the source image is transformed into its DCT form. Then, the hidden message is extracted from the DCT coefficients using a suitable algorithm. The extracted message is then decrypted using the same AES algorithm. The resulting decrypted message is the original text message that was hidden in the image. In conclusion, steganography provides a means to hide messages within other media, and the DCT and AES algorithms are popular methods used to embed and extract the message. While embedding, the DCT coefficients of the image are used to hide the encrypted message, while during extraction, these coefficients are used to extract the hidden message.

CHAPTER 5

SYSTEM IMPLEMENTATION

In this chapter, the System Implementation for Implementing Text Message Concealment through Digital Watermarking is explained in detail.

5.1 IMPLEMENTATION OF DIGITAL WATERMARKING

The project is implemented in visual studio code. Here, the various functionalities required for the application are implemented by coding them in Python.

Data- (text, Image)

5.2 MODULES

5.2.1 Discrete Cosine Transform (DCT):

The Discrete Cosine Transform (DCT) is a mathematical transformation method used to convert signals or images from the spatial domain to the frequency domain. It is similar to the Discrete Fourier Transform (DFT) but uses only real numbers and real-valued cosine functions instead of complex exponentials. The DCT separates signals and images into spectral sub-bands of varying importance with respect to their visual or audible quality, making it an effective tool for compression and other applications in a wide range of industries, including multimedia, telecommunications, and data storage. The DCT can efficiently represent signals and images using only a small number of coefficients, making it computationally efficient and widely supported by hardware and software implementations. Its efficiency and wide support make it a standard tool for compression and other applications in signal processing, image processing, and data compression.

FORMULAS:

DCT-II

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N} \left(n + \frac{1}{2} \right) k \right] \qquad \text{for } k = 0, \ \ldots \ N-1 .$$

DCT-III or IDCT

$$X_k = 1/2 x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N} n \left(k + 1/2 \right) \right] \qquad \text{for } k = 0, \ \ldots \ N-1 .$$

5.2.1 AES Encryption System

The Advanced Encryption Standard (AES) is a widely used symmetric block cipher, designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. It is based on their earlier cipher, Rijndael, and is considered to be one of the most secure encryption algorithms available today. AES has a fixed block size of 128 bits and supports key sizes of 128, 192, or 256 bits. It uses a substitution-permutation network structure that transforms the input data into the final encrypted output. AES is known for its speed and efficiency, making it versatile and widely used in both software and hardware implementations. It is resistant to various cryptographic attacks, including brute force attacks, differential and linear cryptanalysis, and side-channel attacks. Overall, AES is a highly secure and efficient encryption algorithm that has become a standard for protecting sensitive information in various industries and applications.

5.3 LIBRARIES

The dependencies used for these implementations are

Tkinter: Tkinter is a Python library used for creating graphical user interfaces. It can be used to create a simple interface for selecting an image file for steganography, or for displaying the output of the steganography algorithm.

NumPy: NumPy is a Python library used for scientific computing, particularly for numerical operations on arrays. It can be used to manipulate image data by converting it into a numpy array, allowing for easy access and modification of individual pixels.

OpenCV: OpenCV (Open Source Computer Vision) is a library used for computer vision applications, including image and video processing. It can be used for tasks such as image manipulation, filtering, and analysis.

Pillow: Pillow is a Python library that provides tools for image processing and manipulation, including support for many different image file formats. It can be used to read and write image files, as well as perform basic image operations such as resizing and cropping.

PyCrypto: PyCrypto is a Python library used for encryption and decryption of data. It can be used for steganography by encrypting the data to be hidden within an image, making it more difficult to detect.

5.4 PSNR (peak signal noise ratio)

Peak signal-to-noise ratio (PSNR) is a widely used metric to measure the quality of an image. It is calculated based on the mean squared error (MSE) between a noise-free m×n monochrome image and its noisy approximation. The maximum possible pixel value of the image (MAXI) depends on the number of bits used per sample, which is typically 8 bits for images represented using linear

PCM PSNR is expressed in decibels (dB) and is a logarithmic measure of the ratio of the maximum possible signal power to the power of the noise affecting the signal.

FORMULA PSNR (in db) is defined as

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

5.5 ALGORITHM

(1) Algorithm For Gray Scale Images

Embedding:

1. Encrypt the message using AES encryption system.

2. Take the image and divide the image into 8X8 blocks and subtracting with 128.

3. ADCT to each block and round it off.pply

4. Embed the message in each LSB of DCT coefficients.

5. Apply inverse DCT to each block and add each block by 128 and stich back the image in lossless image format (e.g., .png format).

Extraction:

1. Take the stereo image and divide the image into 8X8 blocks and subtracting with 128.

2. Apply DCT to each block and round it off.

3. Extract the message from each LSB of DCT coefficients.

4. Decrypt the message using AES encryption system.

(2) Algorithm for RGB Images

Embedding:

1. Encrypt the message using AES encryption system.
2. Take the image and split it into three channels that is Red , Green and Blue.
3. Take the green channel and divide the image into 8X8 blocks and subtracting with 128.
4. Apply DCT to each block and round it off.
5. Embed the message in each LSB of DCT coefficients.
6. Apply inverse DCT to each block and add each block by 128 and stich back the image in lossless image format (e.g., .png format).

Extraction

1. Take the stereo image and Take the image and split it into three channels that is Red , Green and Blue.
2. Take green channel and the divide the image into 8X8 blocks and subtracting with 128.
3. Apply DCT to each block and round it off.
4. Extract the message from each LSB of DCT coefficients.
5. Decrypt the message using AES encryption system.

CHAPTER 6

CODING AND SCREENSHOTS

6.1 Sample Code

```
from PIL import Image
import numpy as np
import itertools
import cv2
from Crypto.Cipher import AES
from tkinter import *
from PIL import Image, ImageTk
from tkinter import messagebox
from tkinter import filedialog
from tkinter.scrolledtext import *
#import tkinter as tk
import ntpath
class DiscreteCosineTransform:
#created the constructor
def __init__(self):
 self.message = None
        self.numBits = 0
 #utility and helper function for DCT Based Steganography
#helper function to stich the image back together
    def chunks(self,l,n):
        m = int(n)
        for i in range(0,len(l),m):
            yield l[i:i+m]
    #function to add padding to make the function dividable by 8x8 blocks
    def addPadd(self,img,row,col):
```

```
        img = cv2.resize(img,(col+(8-col%8),row+(8-row%8)))
        return img
      def addPadd(self,img,row,col):
        img = cv2.resize(img,(col+(8-col%8),row+(8-row%8)))
#encoding function
   #applying dct for encoding
   def DCTEncoder(self,img,secret):
      self.message = str(len(secret)).encode()+b'*'+secret
      #get the size of the image in pixels
      row, col = img.shape[:2]
      if((col/8)*(row/8)<len(secret)):
         print("Error: Message too large to encode in image")
         return False
      if row%8 or col%8:
         img = self.addPadd(img,row,col)
      row,col = img.shape[:2]
      #split image into RGB channels
      hImg,sImg,vImg = cv2.split(img)
      #message to be hid in saturation channel so converted to type float32 for
dct function
      #print(bImg.shape)
      sImg = np.float32(sImg)
      #breaking the image into 8x8 blocks
      imgBlocks = [np.round(sImg[j:j+8,i:i+8]-128) for (j,i) in
itertools.product(range(0,row,8),range(0,col,8))]
      #print('imgBlocks',imgBlocks[0])
      #blocks are run through dct / apply dct to it
      dctBlocks = [np.round(cv2.dct(ib)) for ib in imgBlocks]
      print('imgBlocks', imgBlocks[0])
      print('dctBlocks', dctBlocks[0])
```

```
        #blocks are run through quantization table / obtaining quantized dct
coefficients
        quantDCT = dctBlocks
        print('quantDCT', quantDCT[0])
        #set LSB in DC value corresponding bit of message
        messIndex=0
        letterIndex=0
        print(self.message)
        for qb in quantDCT:
                #find LSB in DCT cofficient and replace it with message bit
           bit = (self.message[messIndex] >> (7-letterIndex)) & 1
           DC = qb[0][0]
#print(DC)
           DC = (int(DC) & ~31) | (bit * 15)
           #print(DC)
           qb[0][0] = np.float32(DC)
           letterIndex += 1
           if letterIndex == 8:
              letterIndex = 0
              messIndex += 1
              if messIndex == len(self.message):
                 break
        #writing the stereo image
        #blocks run inversely through quantization table
        #blocks run through inverse DCT
        sImgBlocks = [cv2.idct(B)+128 for B in quantDCT]
        #puts the new image back together
        aImg=[]
       for chunkRowBlocks in self.chunks(sImgBlocks, col/8):
```

```
            for rowBlockNum in range(8):
            for block in chunkRowBlocks:
                    aImg.extend(block[rowBlockNum])
        aImg = np.array(aImg).reshape(row, col)
        #converted from type float32
        aImg = np.uint8(aImg)
        #show(sImg)
              Label(hide,text="Message is hidden successfully.",
background="#dfdfdf", foreground='green',font=("Bold",10)).pack()
        return cv2.merge((hImg,aImg,vImg))
     #decoding
         #apply dct for decoding
    def DCTDecoder(self,img):
        row, col = img.shape[:2]
        messSize = None
        messageBits = []
        buff = 0
            #split the image into RGB channels
        hImg,sImg,vImg = cv2.split(img)
        #message hid in saturation channel so converted to type float32 for dct
function
sImg = np.float32(sImg)
        #break into 8x8 blocks
        imgBlocks = [sImg[j:j+8,i:i+8]-128 for (j,i) in
itertools.product(range(0,row,8),range(0,col,8))]
        dctBlocks = [np.round(cv2.dct(ib)) for ib in imgBlocks]
       # the blocks are run through quantization table
        print('imgBlocks',imgBlocks[0])
        print('dctBlocks',dctBlocks[0])
        quantDCT = dctBlocks
```

```
i=0
flag = 0
      #message is extracted from LSB of DCT coefficients
for qb in quantDCT:
   if qb[0][0] > 0:
      DC = int((qb[0][0]+7)/16) & 1
   else:
      DC = int((qb[0][0]-7)/16) & 1
   #print('qb',qb[0][0],'dc',DC)
   #unpacking of bits of DCT
   buff += DC << (7-i)
   i += 1
   #print(i)
   if i == 8:
      messageBits.append(buff)
      #print(buff,end=' ')
      buff = 0
      i =0
      if messageBits[-1] == 42 and not messSize:
      try:
            messSize = chr(messageBits[0])
            for j in range(1,len(messageBits)-1):
               messSize += chr(messageBits[j])
            messSize = int(messSize)
            print(messSize,'a')
         except:
            print('b')
             if len(messageBits) - len(str(messSize)) - 1 == messSize:
      return messageBits
```

```
print ("msgbits", messageBits)
        return None
   class DCTGrayscale:
    #created the constructor
    def __init__(self):
        self.message = None
        self.numBits = 0
#utility and helper function for DCT Based Steganography
    #helper function to stich the image back together
    def chunks1(self,l,n):
        m = int(n)
        for i in range(0,len(l),m):
            yield l[i:i+m]
    #function to add padding to make the function dividable by 8x8 blocks
    def addPadd1(self,img,row,col):
         img = cv2.resize(img,(col+(8-col%8),row+(8-row%8)))
         return img
   #main part
    #encoding function
    #applying dct for encoding
    def DCTEncoder1(self,img,secret):
        self.message = str(len(secret)).encode()+b'*'+secret
        #get the size of the image in pixels
        row, col = img.shape[:2]
        if((col/8)*(row/8)<len(secret)):
        print("Error: Message too large to encode in image")
            return False
        if row%8 or col%8:
            img = self.addPadd1(img,row,col)
```

```
        row,col = img.shape[:2]
        print(row,col)
        #split image into RGB channels
         #hImg,sImg,vImg = cv2.split(img)
        #message to be hid in saturation channel so converted to type float32 for
dct function
        #print(bImg.shape)
        img = np.float32(img)
        #breaking the image into 8x8 blocks
Print ("msgbits", messageBits)
        return None
   class DCTGrayscale:
    #created the constructor
    def __init__(self):
        self.message = None
        self.numBits = 0

    #utility and helper function for DCT Based Steganography
    #helper function to stich the image back together
    def chunks1(self,l,n):
        m = int(n)
        for i in range(0,len(l),m):
            yield l[i:i+m]
    #function to add padding to make the function dividable by 8x8 blocks
    def addPadd1(self,img,row,col):
         img = cv2.resize(img,(col+(8-col%8),row+(8-row%8)))
         return img
   #main part
    #encoding function
```

```
    #applying dct for encoding
    def DCTEncoder1(self,img,secret):
        self.message = str(len(secret)).encode()+b'*'+secret
        #get the size of the image in pixels
        row, col = img.shape[:2]
        if((col/8)*(row/8)<len(secret)):
        print("Error: Message too large to encode in image")
            return False
        if row%8 or col%8:
            img = self.addPadd1(img,row,col)
        row,col = img.shape[:2]
        print(row,col)
        #split image into RGB channels
         #hImg,sImg,vImg = cv2.split(img)
        #message to be hid in saturation channel so converted to type float32 for
dct function
        #print(bImg.shape)
        img = np.float32(img)
        #breaking the image into 8x8 blocks
aImg.extend(block[rowBlockNum])
        aImg = np.array(aImg).reshape(row, col)
        #converted from type float32
        aImg = np.uint8(aImg)
        #show(sImg)
        return aImg
       #decoding
    #apply dct for decoding
    def DCTDecoder1(self,img):
        row, col = img.shape[:2]
```

```
        messSize = None
        messageBits = []
        buff = 0
        #split the image into RGB channels
        #hImg,sImg,vImg = cv2.split(img)
        #message hid in saturation channel so converted to type float32 for dct
function
        img = np.float32(img)
        #break into 8x8 blocks
        imgBlocks = [img[j:j+8,i:i+8]-128 for (j,i) in
itertools.product(range(0,row,8),range(0,col,8))]
        dctBlocks = [np.round(cv2.dct(ib)) for ib in imgBlocks]
       # the blocks are run through quantization table
        print('imgBlocks',imgBlocks[0])
        print('dctBlocks',dctBlocks[0])
        quantDCT = dctBlocks
        i=0
        flag = 0
        #message is extracted from LSB of DCT coefficients
        for qb in quantDCT:
           if qb[0][0] > 0:
              DC = int((qb[0][0]+7)/16) & 1
           else:
              DC = int((qb[0][0]-7)/16) & 1
         #print('qb',qb[0][0],'dc',DC)
            #unpacking of bits of DCT
            buff += DC << (7-i)
i += 1
           #print(i)
           if i == 8:
```

```
            messageBits.append(buff)
            #print(buff,end=' ')
            buff = 0
            i =0
            if messageBits[-1] == 42 and not messSize:
                try:
                    messSize = chr(messageBits[0])
                    for j in range(1,len(messageBits)-1):
                        messSize += chr(messageBits[j])
                    messSize = int(messSize)
                    print(messSize,'a')
                except:
                    print('b')
        if len(messageBits) - len(str(messSize)) - 1 == messSize:
            return messageBits
    print("msgbits", messageBits)
    return None
  def msg_encrypt(msg,cipher):
 if (len(msg)%16 != 0):
    #a = len(msg)%16 != 0
    #print(a)
    msg = msg + ' '*(16 - len(msg)%16)
 #nonce = cipher.nonce
 t1 = msg.encode()
 enc_msg = cipher.encrypt(t1)
 return enc_msg
def msg_decrypt(ctext,cipher):
 dec_msg = cipher.decrypt(ctext)
 msg1 = dec_msg.decode()
```

```
    return msg1


def get_path1(hide_or_extract):
        # Browse button to search for files
        filename = filedialog.askopenfilename(filetypes=(("Template files",
".png"), ("All files", ".*") ))
        if hide_or_extract == 0:
            tp1.set(filename)
else:
            sp1.set(filename)
def update1(event):
        retrieved_text = textbox1.get("1.0", END)
        counter1.set('Charcters count: ' + str(len(retrieved_text)) + ' | Max allowed
characters = 96')


def get_path(hide_or_extract):
        # Browse button to search for files
        filename = filedialog.askopenfilename(filetypes=(("Template files",
".png"), ("All files", ".*") ))
        if hide_or_extract == 0:
            tp.set(filename)
        else:
            sp.set(filename)


def update(event):
        retrieved_text = textbox.get("1.0", END)
        counter.set('Charcters count: ' + str(len(retrieved_text)) + ' | Max allowed
characters = 96')
def hidetext1():
    message = textbox1.get("1.0", END)
```

```
    image = ip1.get()
    print(image)
    print(message)
    if(len(message) > 96):
        messagebox.showwarning("The message is too long. Shorten the
message.")
    else:
        img = cv2.imread(image,cv2.IMREAD_UNCHANGED)
        key = b'Sixteen byte key'
        cipher = AES.new(key,AES.MODE_ECB)
        enc_msg = msg_encrypt(message,cipher)
        d = DCTGrayscale()
        dct_img_encoded = d.DCTEncoder1(img, enc_msg)
        path = ntpath.split(image)[0] + "/enc_" + ntpath.basename(image)[:-
4]+".png"
        cv2.imwrite(path,dct_img_encoded)
def hide1():
    global hide1
    global tp1
    global ip1
    global textvar1
    global counter1
    global char_count1
    global textbox1
    hide1 = Toplevel(hm)
    hide1.title("hide")
    hide1.geometry("1200x700")
    hide1.configure(bg='#dfdfdf')
    #Label(hide, text="", bg ='#dfdfdf').pack()
    Label(hide1, text='                                        Hide Message
```

', fg='white', font=("Bold", 36),bg ='#6a057e').pack()

Label(hide1, text='', bg ='#dfdfdf').pack()

Label(hide1, bg='#dfdfdf', fg='black',font=("Bold", 16), text="Choose The Image (Image Should Be In JPG and PNG Format):").pack()

Label(hide1, text='', bg ='#dfdfdf').pack()

tp1 = StringVar()

ip1 = Entry(hide1, width=55, textvariable=tp1)

ip1.pack()

Label(hide1, text='', bg ='#dfdfdf').pack()

Button(hide1, text="Select target image", bg='#6a057e', fg='white',font=("Bold", 10),command=lambda: get_path1(0)).pack()

Label(hide1, text='', bg ='#dfdfdf').pack()

#print(str(tp))

#print(ip)

#button2.configure(bg="black", fg='white', activebackground='#0080ff', activeforeground='white')

Label(hide1, text=" Enter The Text to hide: ", background="#dfdfdf", foreground='black',font=("Bold", 16)).pack()

Label(hide1, text='', bg ='#dfdfdf').pack()

textvar1 = StringVar()

textbox1 = Text(hide1, height=5, width=70, wrap='word', undo=True)

textbox1.pack()

textbox1.bind("<Key>", update1)

Label(hide1, text='', bg ='#dfdfdf').pack()

counter1 = StringVar()

counter1.set('Charcters count: ' +'0'+ ' | Max allowed characters = 96')

char_count1 = Label(hide1, textvariable=counter1, bg="#dfdfdf", fg= '#0080ff').pack()

Label(hide1, text='', bg ='#dfdfdf').pack()

Button(hide1, text="Hide Text",bg='#6a057e', fg='white',font=("Bold", 10),command=lambda: hidetext1()).pack()

```
def showtext1():
    #message = textbox.get("1.0", END)
    image = imp1.get()
    print(image)
    #print(message)
    img = cv2.imread(image,cv2.IMREAD_UNCHANGED)
    key = b'Sixteen byte key'
    cipher = AES.new(key,AES.MODE_ECB)
    #enc_msg = msg_encrypt(message,cipher)
    d = DCTGrayscale()
msg = d.DCTDecoder1(img)
    #Exception Handling need to be done in decoding phase using tkinter messagebox or labels
    try:
        a = msg.index(42)
        decoded = bytes(msg[a+1:])
        text = msg_decrypt(decoded,cipher)
        extmsg1.set(text)
    except UnicodeDecodeError:
        extmsg1.set('\nError 1: Falied To Decode The Message.\n')
    except AttributeError:
        extmsg1.set('\nError 2: Failed To Extract The Message.\n')
    except ValueError:
        extmsg1.set('\nError 3: Failed To Extract The Message.\n')
    except:
        extmsg1.set('\nError 4: Unexpected Error\n')
def show1():
    pass
```

```
    global show1
    global imp1
    global sp1
    global u1
    global extmsg1
    global exm1
    show1 = Toplevel(sm)
    show1.title("Show")
    show1.geometry("1200x800")
    show1.configure(bg='#dfdfdf')
    #Label(hide, text="", bg ='#dfdfdf').pack()
    Label(show1, text='                                        Show Message
', fg='white', font=("Bold", 36),bg ='#6a057e').pack()
    Label(show1, text='', bg ='#dfdfdf').pack()
    Label(show1, bg='#dfdfdf', fg='black',font=("Bold", 16), text="Choose The
Image (Image Should Be In PNG Format Only):").pack()
    Label(show1, text='', bg ='#dfdfdf').pack()
  sp1 = StringVar()
    imp1 = Entry(show1, width=55, textvariable=sp1)
    imp1.pack()
    Label(show1, text='', bg ='#dfdfdf').pack()
    Button(show1, text="Select target image", bg='#6a057e',
fg='white',font=("Bold", 10),command=lambda: get_path1(1)).pack()
    Label(show1, text='', bg ='#dfdfdf').pack()
    u1 = LabelFrame(show1, bg='white', fg='#6a057e', text=" Extracted text
",font=("Bold", 14))
    u1.pack(expand=1, fill="both",padx=5, pady=5)
    extmsg1 = StringVar()
    extmsg1.set('\nExtracted Text\n')
    exm1 = Label(u1, textvariable=extmsg1, background='white',
foreground="black",font=("Bold", 12))
```

```
    exm1.pack(expand=1, fill="both", padx=5, pady=5)
    Label(show1, text='', bg ='#dfdfdf').pack()
    Button(show1,text="Extract Text",bg='#6a057e', fg='white',font=("Bold", 10),command=lambda: showtext1()).pack()
    #mainloop()
def hidetext():
    message = textbox.get("1.0", END)
    image = ip.get()
    print(image)
    print(message)
    if(len(message) > 96):
        messagebox.showwarning("The message is too long. Shorten the message.")
    else:
        img = cv2.imread(image,cv2.IMREAD_UNCHANGED)
        key = b'Sixteen byte key'
        cipher = AES.new(key,AES.MODE_ECB)
        enc_msg = msg_encrypt(message,cipher)
        d = DiscreteCosineTransform()
        dct_img_encoded = d.DCTEncoder(img, enc_msg)
        path = ntpath.split(image)[0] + "/enc_" + ntpath.basename(image)[:-4]+".png"
        cv2.imwrite(path,dct_img_encoded)
def hide_menu():
    global hm
    hm = Toplevel(main_win)
    hm.title("Hide Menu")
    hm.geometry("400x200")
    hm.configure(bg='#dfdfdf')
    Label(hm, text='                    Choose The Image Type
```

```
', fg='white', font=("Bold", 24),bg ='#6a057e').pack()
    Label(hm, text='', bg ='#dfdfdf').pack()
    Button(hm, text="GrayScale - 8 Bit Depth", width=30, height=1, fg='white', bg="#6a057e", font=("Bold", 12),command=hide1).pack()
    Label(hm, text='', bg ='#dfdfdf').pack()
    Button(hm, text="RGB - 24 Bit Depth", width=30, height=1, fg='white', bg="#6a057e", font=("Bold", 12),command=hide).pack()
def hide():
    global hide
    global tp
    global ip
    global textvar
    global counter
    global char_count
    global textbox
    hide = Toplevel(hm)
   hide.title("hide")
    hide.geometry("1200x700")
    hide.configure(bg='#dfdfdf')
    #Label(hide, text="", bg ='#dfdfdf').pack()
    Label(hide, text='                                                    Hide Message
', fg='white', font=("Bold", 36),bg ='#6a057e').pack()
    Label(hide, text='', bg ='#dfdfdf').pack()
    Label(hide, bg='#dfdfdf', fg='black',font=("Bold", 16), text="Choose The Image (Image Should Be In JPG and PNG Format):").pack()
    Label(hide, text='', bg ='#dfdfdf').pack()
    tp = StringVar()
    ip = Entry(hide, width=55, textvariable=tp)
    ip.pack()
    Label(hide, text='', bg ='#dfdfdf').pack()
```

```
    Button(hide, text="Select target image", bg='#6a057e',
fg='white',font=("Bold", 10),command=lambda: get_path(0)).pack()

    Label(hide, text='', bg ='#dfdfdf').pack()

    #print(str(tp))

    #print(ip)

    #button2.configure(bg="black", fg='white', activebackground='#0080ff',
activeforeground='white')

    Label(hide, text=' Enter The Text to hide: ',  background="#dfdfdf",
foreground='black',font=("Bold", 16)).pack()

    Label(hide, text='', bg ='#dfdfdf').pack()

    textvar = StringVar()

    textbox = Text(hide, height=5, width=70,  wrap='word', undo=True)

    textbox.pack()

    textbox.bind("<Key>", update)

    Label(hide, text='', bg ='#dfdfdf').pack()

    counter = StringVar()

    counter.set('Charcters count: ' +'0'+ ' | Max allowed characters = 96')

char_count = Label(hide, textvariable=counter, bg="#dfdfdf", fg=
'#0080ff').pack()

    Label(hide, text='', bg ='#dfdfdf').pack()

Button(hide, text="Hide Text",bg='#6a057e', fg='white',font=("Bold",
10),command=lambda: hidetext()).pack()#command=lambda: self.hidetext())

#button21.pack(side=tk.RIGHT, padx=5, pady=5)

    #button21.configure(bg="black", fg='white', activebackground='#0080ff',
activeforeground='white')

    #mainloop()


def showtext():

    #message = textbox.get("1.0", END)

    image = imp.get()

    print(image)
```

```
    #print(message)
    img = cv2.imread(image,cv2.IMREAD_UNCHANGED)
    key = b'Sixteen byte key'
    cipher = AES.new(key,AES.MODE_ECB)
    #enc_msg = msg_encrypt(message,cipher)
    d = DiscreteCosineTransform()
    msg = d.DCTDecoder(img)
    #Exception Handling need to be done in decoding phase using tkinter messagebox or labels
try:
        a = msg.index(42)
        decoded = bytes(msg[a+1:])
        text = msg_decrypt(decoded,cipher)
        extmsg.set(text)
    except UnicodeDecodeError:
        extmsg.set('\nError 1: Falied To Decode The Message.\n')
    except AttributeError:
        extmsg.set('\nError 2: Failed To Extract The Message.\n')
    except ValueError:
        extmsg.set('\nError 3: Failed To Extract The Message.\n')
    except:
        extmsg.set('\nError 4: Unexpected Error\n')

#path = ntpath.split(image)[0] + "/enc_" + ntpath.basename(image)[:-4]+".png"
    #cv2.imwrite(path,dct_img_encoded)

def show_menu():
    global sm
    sm= Toplevel(main_win)
```

```
    sm.title("Show Menu")
    sm.geometry("400x200")
    sm.configure(bg='#dfdfdf')
    Label(sm, text='                    Choose The Image Type
', fg='white', font=("Bold", 24),bg ='#6a057e').pack()
    Label(sm, text='', bg ='#dfdfdf').pack()
    Button(sm, text="GrayScale - 8 Bit Depth", width=30, height=1, fg='white',
bg="#6a057e", font=("Bold", 12),command=show1).pack()
    Label(sm, text='', bg ='#dfdfdf').pack()
    Button(sm, text="RGB - 24 Bit Depth", width=30, height=1, fg='white',
bg="#6a057e", font=("Bold", 12),command=show).pack()
def show():
    pass
    global show
    global imp
    global sp
    global u
    global extmsg
    global exm
    show = Toplevel(sm)
    show.title("Show")
    show.geometry("1200x800")
    show.configure(bg='#dfdfdf')
    #Label(hide, text="", bg ='#dfdfdf').pack()
    Label(show, text='                                        Show Message
     Label(show, text='', bg ='#dfdfdf').pack()
    Label(show, bg='#dfdfdf', fg='black',font=("Bold", 16), text="Choose The
Image (Image Should Be In PNG Format Only):").pack()
    Label(show, text='', bg ='#dfdfdf').pack()
```

```
    sp = StringVar()
    imp = Entry(show, width=55, textvariable=sp)
    imp.pack()
    Label(show, text='', bg ='#dfdfdf').pack()
    Button(show, text="Select target image", bg='#6a057e',
fg='white',font=("Bold", 10),command=lambda: get_path(1)).pack()
    Label(show, text='', bg ='#dfdfdf').pack()
    u = LabelFrame(show, bg='white', fg='#6a057e', text=" Extracted text
",font=("Bold", 14))
    u.pack(expand=1, fill="both",padx=5, pady=5)
    extmsg = StringVar()
    extmsg.set('\nExtracted Text\n')
    exm = Label(u, textvariable=extmsg, background='white',
foreground="black",font=("Bold", 12))
    exm.pack(expand=1, fill="both", padx=5, pady=5)
    Label(show, text='', bg ='#dfdfdf').pack()
    Button(show,text="Extract Text",bg='#6a057e', fg='white',font=("Bold",
10),command=lambda: showtext()).pack()
    #mainloop()
def main_win():
    global main_win
    main_win = Tk()
    main_win.title("Image Steganography")
    main_win.geometry("626x417")
    bg = PhotoImage(file = "C:/Users/KRISHNAKANTH/Downloads/Image-
Steganography/Assignment/Lenna1.png")
    canvas1 = Canvas( main_win, width = 626,height = 417)
    canvas1.pack(fill = "both", expand = True)
    canvas1.create_image( 0, 0, image = bg, anchor = NW)
    b1 = Button(main_win, text="Hide Message", font=("Bold", 16),
fg='black',bg='white', command = hide_menu)
```

```
    b2 = Button(main_win, text="Show Message", font=("Bold", 16), fg='black',
bg='white',command = show_menu)

    button1_canvas = canvas1.create_window( 235,  175, anchor = "nw", window
= b1)

    button2_canvas = canvas1.create_window( 230, 275,anchor = "nw", window
= b2)

    mainloop()


main_win()
```

6.1 SAMPLE SCREENSHOTS

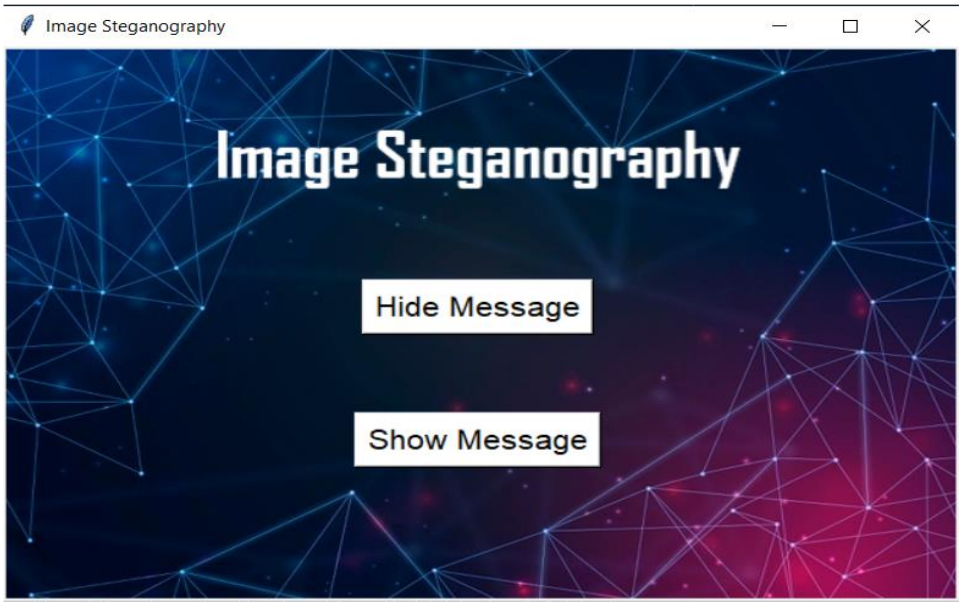

Figure 6.2 To select an option Hide or Show a message

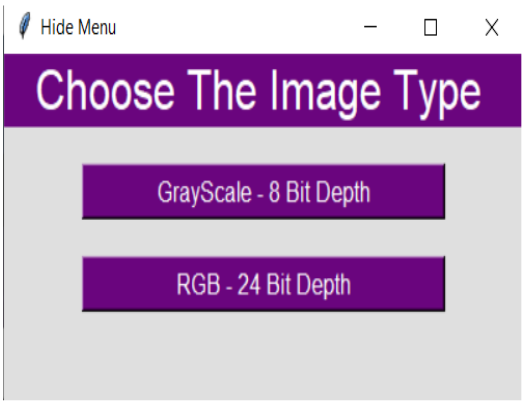

Figure 6.3 Select an image for Encrption

Figure 6.4 Select an Encrypted Image

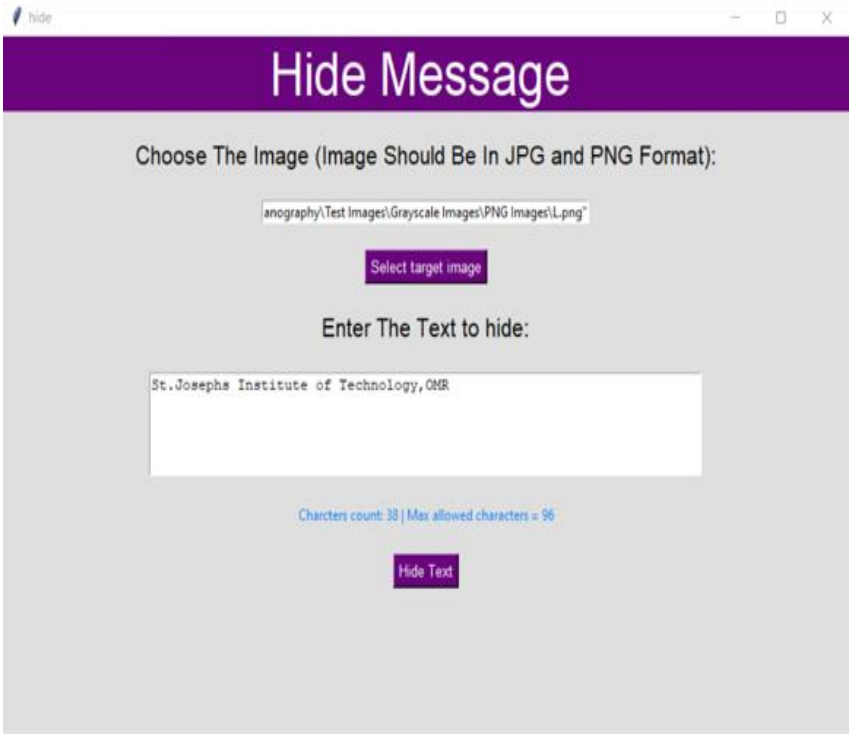

Figure 6.5 Encoding a message

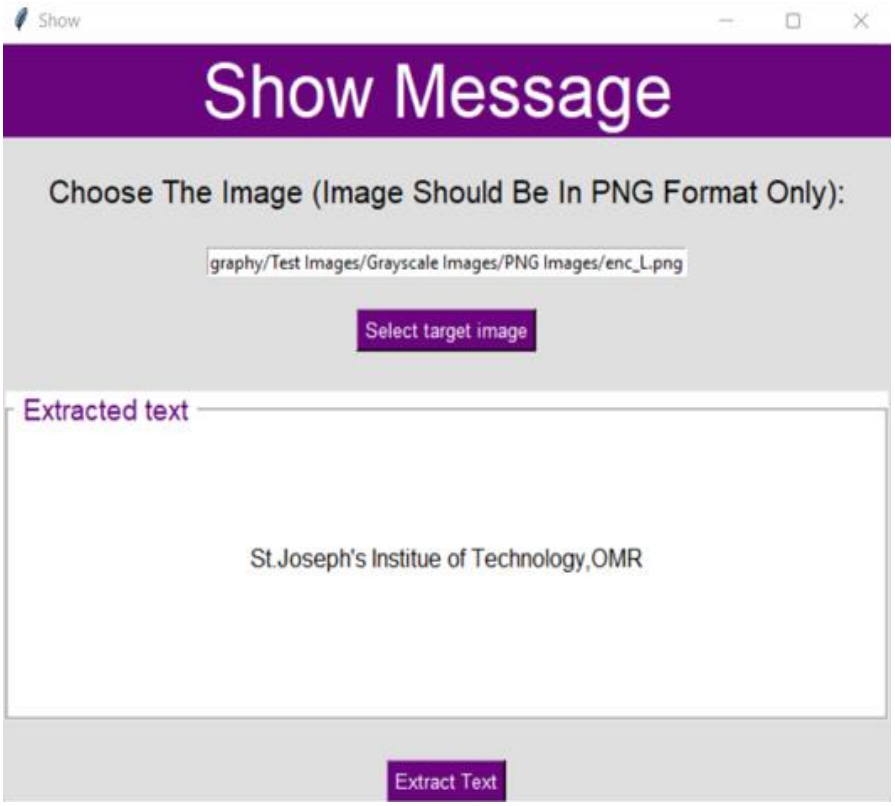

Figure 6.6 Decoding a message

OUTPUT

BEFORE

Figure 6.7 Original image - APPLE

AFTER

Figure 6.8 Encoded image – enc_Apple

BEFORE

Figure 6.9 Original image – Rose

AFTER

Figure 6.10 Encoded image - enc_Rose

CHAPTER 7

7.1 CONCLUSION AND FUTURE WORK

Since the proposed technique is more secure than spartial technique, but it has some disadvantages. Firstly, it AES – 128 is not much secure in nature. Secondly, ECB mode used in AES -128 is least secure. Lastly, however dct is computationally fast stores images with less distortion but some of the messages cannot be extracted properly due to the increase in the noise level above 0.32bpp. DCT also does not provide proper robostness to geometric attacks. To overcome the disadvantages, one can use DWT and DFT instead of DCT or they can use Genetic algorithms and Lossless Image Compression techniques for solving message extraction problem and one can use AES-256 with more secure modes such as CBC and EAX for enhancing the security. Researchers are still figuring out many ways for finding out highly secure image steganography methods.

In future work, we can use Lossless image compression techniques like Huffman coding and Arithmetic coding can be used to compress the image before steganography. These techniques can reduce the size of the image, making it easier to hide the message and providing better security. And Machine learning techniques can be used to develop more intelligent steganography systems that can learn from past attacks and adapt to new ones. This approach can improve the security and robustness of image steganography techniques. In conclusion, image steganography is an area of research that is constantly evolving, and there is a need for more secure and robust techniques. Researchers can focus on the above areas to develop more secure and robust image steganography techniques in the future.

REFRENCES

[1]. Akhtar, N.; Johri, P.; Khan, S., "Enhancing the Security and Quality of LSB Based Image Steganography," Computational Intelligence and Communication Networks (CICN), 2013 5th International Conference on , vol., no., pp.385,390, 27-29 Sept. 2013. based image Steganography," Recent Trends in Information Systems (ReTIS), 2011 International Conference on , vol., no., pp.214,217, 21-23 Dec. 2011.

[2]. Andrew D. Ker (2010). "Steganalysis of LSB matching in grayscale images." Proceedings of the 2010 2nd International Conference on Signal Processing Systems (ICSPS), pp. V2-334-V2-337

[3]. Bingwen Feng, Jianwei Liu, Xuan Kong, and Zhangjie Fu (2015). "Rotation, Complement and Mirroring Invariant Texture Patterns for Binary Image Steganography," Information Sciences, vol. 300, pp. 19-33.

[4]. C.-C. Chang, C.-S. Tsai, and T.-S. Chen (2007). "A Novel Steganographic Method by Pixel-Value Modification," Proceedings of the IEEE International Conference on Multimedia and Expo (ICME), pp. 2038-2041.

[5]. Da-Chun Wu, Ming-Shing Hsieh, Chin-Feng Lee, and Ya-Fen Chang (2013). "A differencing steganographic method with a range table and no distortion compensation," Journal of Visual Communication and Image Representation, vol. 24, pp. 792-800.

[6]. R. Das, S. Sarkar, and S. Nandi (2012). "A novel approach for image steganography using Huffman encoding technique," International Journal of Computer Applications, vol. 55, no. 16, pp. 29-34.

[7]. Della Baby, P. V. Sridevi, and K. Rajesh (2015). "A novel DWT based image securing method using steganography," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, no. 9, pp. 858-864.

[8]. Dagar, E., & Dagar, S. (2013). A novel approach for color image steganography based on X-box mapping. International Journal of Advanced Research in Computer Science, 4(5), 213-216.

[9]. G. Prashanti, N. Sudhakar, and P. Rajesh Kumar, "A survey on recent achievements of LSB based image steganography," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, no. 8, pp. 764-769, 2015.

[10]. Hemalatha, S.; Acharya, U.D.; Renuka, A.; Kamath, P.R., "A secure image steganography technique using Integer Wavelet Transform," Information and Communication Technologies (WICT), 2012 World Congress on , vol., no., pp.755,758, Oct. 30 2012-Nov. 2 2012.

[11]. Huaiqing Wang, Qingzhong Liu, Zhenxing Qian, and Jingjing Liu. (2014). A survey on steganography and steganalysis. Journal of Information Hiding and Multimedia Signal Processing, 5(3), 443-452.

[12]. Jessica Fridrich, Miroslav Goljan, Dorin Hogea, "Estimation of secret message length in LSB steganography in spatial domain", Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security (WIFS), 2012, pp. 59-64.

[13]. Johnson, N. F., Jajodia, S., & Stavrou, A. (2002). Exploring steganography: seeing the unseen. IEEE computer graphics and applications, 22(4), 26-34.

[14]. Qazanfari, K., Mohammadi, M., & Boroumand, L. (2014). A novel improved LSB++ approach for image steganography. Journal of Electronic Imaging, 23(1), 013024. doi: 10.1117/1.JEI.23.1.013024

[15]. Nusrati, M., Mirzakuchaki, S., & Zareapoor, M. (2015). Heuristic genetic algorithm based steganographic method for hiding secret information in a cover image. Journal of Information Security and Applications, 21, 54-65.