R.V. COLLEGE OF ENGINEERING,

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Bangalore – 560059

A Report on

"More Secure Image Steganography in Spatial Domain"

Submitted in partial fulfillment of the requirements for the Seventh Semester of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE & ENGINEERING

by

Praveen Kumar D (USN: 1RV09CS074)

Sham Prasad PS (USN: 1RV09CS098)

Under the guidance

of

Mrs. Usha BA

Assistant Professor,

Department of CSE, R.V.C.E.,

Bangalore - 560 059

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590 014

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

R.V. COLLEGE OF ENGINEERING,

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Mysore Road, R.V.Vidyaniketan Post, Bangalore - 560059

CERTIFICATE

Certified that the Mini-project work entitled **“More Secure Image Steganography in Spatial Domain”** has been carried out by **Mr.Praveen Kumar D and Mr. Sham Prasad PS,** USNs: **1RV09CS074** and **1RV09CS098**, bonafide students of **R.V. College of Engineering, Bangalore** in partial fulfillment of the requirements of Seventh Semester of **Bachelor of Engineering** in **Computer Science and Engineering** during the year 2012 -2013. 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 requirement in respect of project work prescribed for the said Degree.

Mrs. Usha B A	**Prof. N.K. Srinath**
Assistant Professor,	Head of Department,
Department of CSE,	Department of CSE,
R.V.C.E., Bangalore –59	R.V.C.E., Bangalore –59

Name of the Examiners	**Signature with Date**
1.____________________	____________________
2.____________________	____________________

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM

R.V. COLLEGE OF ENGINEERING,

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Mysore Road, R.V.Vidyaniketan Post, Bangalore - 560059

DECLARATION

We, Praveen Kumar D and Sham Prasad PS, students of Seventh Semester, B.E., in the Department of Computer Science and Engineering, R.V. College of Engineering, Bangalore declare that the Mini-project entitled **"More Secure Image Steganography in Spatial Domain"** has been carried out by us and submitted in partial fulfillment of the course requirements for Seventh Semester of Bachelor of Engineering in Computer Science and Engineering during the year **2012 -2013**. The matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Praveen Kumar D
Sham Prasad P S
USNs: 1RV09CS074
1RV09CS098

Department of Computer Science and Engineering,
R.V. College of Engineering,
Bangalore-560059

ACKNOWLEDGEMENT

Our project was the result of the encouragement of many people who helped in shaping it and provided feedback, direction and valuable support. It is with hearty gratitude that we acknowledge their contributions to our project.

We are highly indebted to our internal guide **Mrs. Usha B A** for her continuous guidance and the help she provided during the course of the project work.

We would like to thank our **HOD**, **Dr. N K Srinath** for his constant support extended towards us during the course of the project. His help and advice has been entirely responsible for instilling the drive in us to complete the project on time.

We would also like to thank our principal **Dr. B S Satyanarayan** who has always been a great source of inspiration.

Last, but not the least, we would like to thank our peers and friends who provided us with valuable suggestions to improve our project.

Praveen Kumar D	USN: 1RV09CS074
Sham Prasad P S	USN: 1RV09CS098

ABSTRACT

This project presents a new approach for hiding message in digital image in spatial domain. In this method two bits of message is embedded in a pixel in a way that not only the least significant bit of pixel is allowed to change but also the second bit plane and fourth bit plane are allowed to be manipulated, But the point is in each embedding process only one alternation in one bit plane is allowed to happen. As it is compared by the method LSB-Matching, the results shows this method has an acceptable capacity of embedding data and hardly is detectable for steganalysis algorithm.

In this project we have implemented a novel method of data embedding in images using a more secure steganographic algorithm in spatial domain. An additional level of security is added by using the Advanced Encryption Standard (AES) for encrypting the data before embedding it.

Table of Contents

Chapter 1

Introduction

Safe communication is attracting attention these years. Steganography is considered a good way to reach this goal. The word steganography is derived from the Greek word “stagos” which means ‘hidden writing’. Steganography is the art of hiding information in a medium called cover. The information that is supposed to be hidden in the cover is called “message“ and the final manipulated signal which carries this message is called stego. Digital image, video, audio, text etc can all be a good medium to carry a hidden data.

1.1 Image Steganography

Image steganography is a technique of concealing confidential data that needs to be transmitted in digital cover images such that presence of data is hidden from third party other than sender and receiver. The purpose of Image steganography is to enhance the communication security by embedding different data files into digital cover images and to prevent an adversary from extracting the data. Data encryption can also provide secure communication but it reveals that the data is encrypted and may contain some valuable information.

The important factors that need to be considered while designing a Steganographic system are embedding capacity (i.e. number of secret bits that can be embedded per pixel), visual quality of stego image (i.e. image distortion), security (encryption) and amount of data (compression) shared. It is necessary to achieve high embedding capacity and good visual quality. The basic structure of Steganography is made up of three components: the carrier, the message, and the key. The carrier can be a painting, a digital image, an mp3, even a TCP/IP packet among other things. It is the object that will ‘carry’ the hidden message. A key is used to decode/decipher/discover the hidden message.

The existing schemes of data hiding can be roughly classified into the following schemes:

1.1.1 Frequency domain data hiding

Frequency domain data hiding is a technique in which images are transformed into frequency domain using Z Transform etc. The transformed coefficients are modified to embed data. However, the limitations of this technique are relatively higher computational cost and lower embedding capacity than the spatial domain data hiding. Steganography in the frequency domain involves the manipulation of algorithms and image transforms . These methods hide messages in more significant areas of the cover image, making it more robust . Many frequency domain methods are independent of the image format and the embedded message may survive conversion between lossy and lossless compression.

1.1.2 Spatial domain data hiding

Spatial domain data hiding is a technique in which image pixels are directly modified to embed data . This technique gives relatively high hiding capacity and high quality of stego image than the other methods. Spatial domain techniques encompass bit-wise methods that apply bit insertion and noise manipulation and are sometimes characterized as “simple systems” . The image formats that are most suitable for spatial domain steganography are lossless and the techniques are typically dependent on the image format .

1.2 Evaluation criteria of Image Steganography Techniques

Below is a list of criteria that measure the degree of success of an images steganography technique.

- Level of Visibility (Perceptible or Imperceptible):

Steganography techniques should embed information in such away that embed data don’t leave a signs of steganography. The visibility is directly influenced by the size of the secret message, the format and the content of the carrier image.

- Detectability:

The researchers of steganalysis develop a series of tools that detect using a specific steganography technique e.g StegSpy, Stegadetect, or Chi square that lead to investigate more and more techniques to avoid detectability problem.

- Robustness vs. Payload:

The embedded data should survive any reprocessing operation for the cover media which goes through and preserve its fidelity. A trade-off exists between the payload size (message size) and robustness. For example the LSB techniques have the capacity to hide larger amount of information in a cover image but a little reprocessing to the resulted image will destroy information completely, where embedding in blocks of pixels will be more robustness.

- Domain Type:

The techniques that use transform domain hide information in significant areas of the cover images but the main disadvantage of these is restricted to lossy format with different quality factor.

- File Format Dependence:

Some techniques employed to hide information depend upon specific characteristics to a carrier type or format while other techniques may work without relying on a specific file format.

1.3 Literature Review

In this section, a review of some of the existing and in use methods for steganography

1.3.1 Least Significant Bit substitution (LSB)

Least Significant Bit (LSB) substitution method is a very popular way of embedding secret messages with simplicity. The fundamental idea here is to insert the secret message in the least significant bits of the images. This actually works because the human visual system is not sensitive enough to pick out changes in color whereas changes in luminance are much better picked out. A basic algorithm for LSB substitution is to take the first N cover pixels where N is the total length of the secret message that is to be embedded in bits. After that every pixel's last bit will be replaced by one of the message bits. In some implementations of LSB coding, however, the two least significant bits of a sample are replaced with two message bits. This increases the amount of data that can be encoded but also increases the amount of resulting noise in the audio file as well. One must decide then on how to choose the subset of samples (in case of audio files) that will contain the secret message and communicate that decision to the receiver. One trivial technique is to start at the beginning of the sound file and perform LSB coding until the message has been completely embedded, leaving the remaining samples unchanged. This creates a security problem, however in that the first part of the sound file will have different statistical properties than the second part of the sound file that was not modified. One solution to this problem is to pad the secret message with random bits so that the length of the message is equal to the total number of samples.

1.3.2 JPEG steganography

One of the major characteristics of steganography is the fact that information is hidden in the redundant bits of an object and since redundant bits are left out when using JPEG it was feared that the hidden message would be destroyed. Even if one could somehow keep the message intact it would be difficult to embed the message without the changes being noticeable because of the harsh compression applied. However, properties of the compression algorithm have been exploited in order to develop a steganographic algorithm for JPEGs. One of these properties of JPEG is exploited to make the changes to the image invisible to the human eye.. In JPEG compression algorithm is actually divided into lossy and lossless stages. The DCT and the quantization phase form part of the lossy stage, while the Huffman encoding used to further compress the data is lossless. Steganography can take place between these two stages. Using the same principles of LSB insertion the message can be embedded into the least significant bits of the coefficients before applying the Huffman encoding. By embedding the information at this stage, in the transform domain, it is extremely difficult to detect, since it is not in the visual domain.

1.3.3 Inverted Pattern Approach

This method tries to process the secret messages prior to embedding. There is a decision about whether invert a specified section of secret images or not. Those bits which are used to record the transformation are treated as secret keys. This method follows a specific procedure to embed the image. Suppose that the embedding message is S, the replaced string is R**,** and the embedded bit string to divided to P parts. If we suppose that n-bit LSB substitution will be maid, then S and R would be n-bit length. In the retrieval phase, the bit will be inverted only if the key is equal to 1. After inversion of mentioned bits, the hidden data will be given. This method is also used with relative entropy. Relative entropy calculates the relative entropy, instead of the mean square error for inverted pattern approach, to decide whether S suits the pixel.

1.4 Proposed Algorithm

A More Secure Steganography Method in Spatial Domain

In this algorithm it is planned to introduce a method that embed 2 bits information in a pixel and alter one bit from one bit plane but the message does not necessarily place in the least significant bit of pixel and second less significant bit plane and fourth less significant bit plane can also host the message. Since in our method for embedding two bits message we alter just one bit plane, fewer pixels would be manipulated during embedding message in an image and it is expected for the steganalysis algorithm to have more difficulty detecting the covert communication. It is clear that in return complexity of the system would increase. A convolution decoder (3, 1) is shown in the Fig. 1. This machine is not going to work as a decoder and is supposed to help us to change the amount of pixels in the cover image to produce a new stego image. This machine works in this manner. Each time the 4 less Significant Bits of a pixel will enter this machine.

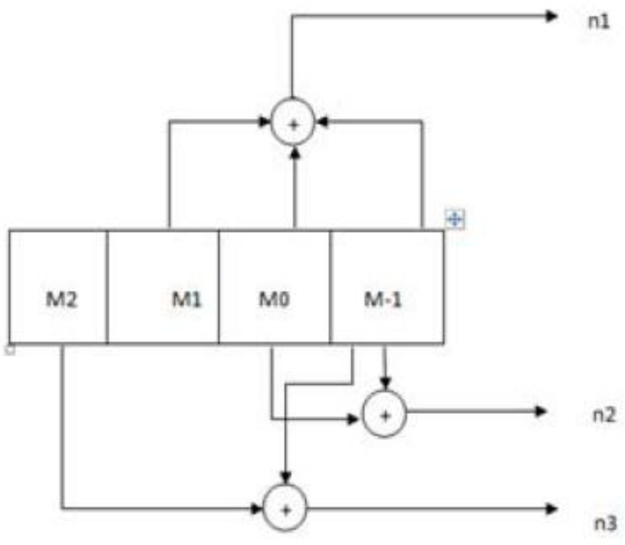


Fig. 1. Cover to stego transformer machine

Three XOR operations create three outputs for this machine. n1, n2, n3 are the outputs of this machine. So as it can be seen, this machine simply do just three XOR operations. Suppose n2, n3 as the hidden message which is embedded in the pixel. If n2, n3 be the same as hidden information, then there is no need to manipulate the original image. In this method there are only three ways that a pixel is allowed to be changed:

- Its least significant Bit would alter (So the gray level of the pixel would increased or decreased by one level)
- The second less significant bit plane would alter (So the gray level of the pixel would increase or decrease by two levels)
- The fourth less significant bit plane would alter (So the gray level of the pixel would increase or decrease by eight levels)

1.5 Motivation

- Any intruder will not be able to extract the data from the image and also, he cannot be able to even guess that a certain message is being transmitted.
- The data is compressed and encrypted to securely embed more and variable types of data.
- There is an increased need for securing the data that is transmitted. Every person wants the message to be read only by the intended recipient.

1.6 Purpose

The main purpose of this project is to develop improved data hiding technique to provide security by embedding the data files into the cover images without compromising the quality of the Steganographic image.

1.7 Scope

- Data is a very important entity that has to be securely transmitted between sender and the receiver. This can be achieved using Image Steganography .
- It can be used in various fields like military, multimedia applications where secured communication is essential.

1.8 Organization of the report

The main body of the report is preceded by detailed contents and introduction including scope, purpose and motivation. This is followed by executive summary giving briefly.

- Chapter 1 explains the importance of the topic, scope, purpose, motivation, related work and proposed work.
- Chapter 2 is Software Requirement Specification which explains the user characteristics, assumptions and dependencies, constraints and functional requirements.
- Chapter 3 is High Level Design which explains the architectural strategies, system architecture, and flow of data in the system with the help of Data Flow Diagrams (DFD).
- Chapter 4 is Detailed Design which explains the modules used in the application.
- Chapter 5 is Implementation which explains the programming language, difficulties encountered in the course of implementation of the project and strategies used to tackle them.
- Chapter 6 is Software Testing which explains briefly the test cases which were executed during the system testing and results.
- Chapter 7 is Conclusion conveying the summary, limitations and future enhancements of the project.

The main report is followed by appendices, giving the source code, snapshots, and paper. References which have been used for certain inputs are listed before the appendices.

Chapter 2

Software Requirements Specification

In system engineering and software engineering, requirement analysis encompasses those tasks that go into determining the requirements of a new or altered system, taking account of the possibly conflicting requirements of various stakeholders, such as users. Requirements analysis is critical to the success of a project. Requirements must be measurable, testable related to identified business needs or opportunities and defined to a level of detail sufficient to a system design.

It is important to note that SRS contains functional and non-functional requirements only. It does not offer design suggestions, possible solutions to technology or business issues or any other information other than what the development team understands the customer’s system requirements to be.

2.1 Overall description

Since the software requirements specification contains both functional and non-functional requirements, the following document states both of these requirements. The document also explains the product functions and user characteristics in a comprehensive manner.

2.1.1 Product features

The project is used for demonstrating a new mechanism to hide information secretly in an image.

The image can be in any format like JPEG, BMP or PPM. The project assumes no prior knowledge of steganography by user.

The GUI has been designed in such a way that any person can use the tool for hiding information in an image file. However, the user must not tamper the stego image.

2.1.2 Constraints

During the development of the application, various constraints were encountered. Some specific constraints for image steganography are as follows:

- The host file size is a major constraint. The smaller the host files, less information it can contain.
- Files can be altered only to a certain degree without losing functionality.
- The Application should provide the functionalities such as encrypting and decrypting of message in image.

2.1.3 Assumptions and dependencies

Some of the assumptions and dependencies are as follows:

- The amount of information that can be hidden inside a cover image is dependent on the size of the host file.
- No loss of information takes place during data transmission.
- User has the minimum knowledge to interact with the GUI.

2.2 Specific Requirements

This section covers the functional requirements that are to be satisfied by the system. The specific requirements are the actual data over which the customer and software provider can agree. That is, the final product is expected to satisfy all the requirements mentioned here.

2.2.1 Functional Requirements

The functional requirements for a system describe the functionality or the services that the system is expected to provide. These include

- Increasing the data security by encrypting the data before the actual data embedding.
- Efficient embedding and extraction of messages from the cover image.

2.2.2 Non-Functional Requirements

2.2.2.1 Usability

The system shall do the information hiding using the image (Carrier Image) specified by the user and the information to be hidden.

2.2.2.2 Reliability

The information hiding module will be reliable in hiding the required information without any obstacles.

2.2.2.3 Robustness

The hidden information in the images should be intact as much as possible whenever the stego images are subjected to manipulation. The module should be able to hide file of any format in a image.

2.2.2.4 Undetectablity

The stego image must not arouse a suspicion when passed over public networks. Only the intended person should be able to recover it.

2.2.2.5 Supportability

All types of images can be used as cover images. But most preferable one is .bmp image. Different media files of any format can be used as secret information when the option for embedding a file is selected. The stego image is always stored in .ppm format as it is lossless.

2.2.3 Interfaces

Interfaces are required for the easy access of underlying system. The tool consists of user interface for all the operations.

- **User Interfaces**

User interfaces include buttons for selecting operations like selection of cover image, key image and stego image. It also contains buttons for viewing the cover image before and after embedding the data. AES encryption can be used to encrypt the data before embedding into the cover image which can be decrypted using the same key while extracting the data from the stego image.

2.2.4 Software Requirements

- A 32-bit/64-bit machine running LINUX.
- A C++ (Qt-compiler) compiler to compile the source file.

2.2.5 Hardware Requirements

The hardware requirements are pretty much standard and are as follows:

- Processor – Intel Pentium processor or above
- Processor Speed – 1GHz or above
- RAM – 256MB or above

Chapter 3

High level design

Design is the second phase in Software Development Life Cycle (SDLC). The objective of the design stage is to produce the overall design of the software. The design stage involves two sub stages:

- High Level Design
- Low level Design

In high level design, the technical architect of the project will study functional and non-functional (qualitative) requirements and design overall solution architecture of the application, which can handle those requirements.

The detailed design includes the explanation for all the modules. It throws light on the purpose, functionality, input and output.

The software specification requirements have been studied to design steganography in which a secret data is embedded into digital cover (image) and can be extracted later.

3.1 Design Consideration

There are several design consideration issues that need to be addressed or resolved before designing a complete solution for the system. The following sections describe assumptions and dependencies of the software, any global limitations or constraints that have significant impact on the software, a method or approach used for the development and the architectural strategies. It also describes system design that provides a high level overview of the functionality and responsibilities.

3.1.1 General Constraints

Given below are certain constraints that have impacted the design of application:

- Interoperability: The project has been implemented on Linux platform. The end user’s environment might be different from that of which the project has been developed.
- Encryption if involved increases security and running time.
- Size of cover image affects the amount of information embedded.

3.1.2 Development methods

The design method employed is highlighted in this section:

- The data flow model has been the design method employed for development of the image steganography using LSB matching for secured data transmission package.
- A data flow model is modeling system based on data transformation that takes place as the data is being processed. The notations used represent functional processing, data stores.
- Data flow models are valuable because tracking and documenting how the data associated with a particular process moves through the system helps users better understanding the system.

3.2 Architectural Strategies

3.2.1 Programming Language

C++ is the programming language chosen for implementation of the application. C++ is the optimal programming language for the project because of the following reasons:

- Developing graphical user interface is very simple in c++. It has a rich set of classes and methods for creating forms, buttons, labels and events. As a result a clean and nice looking user interface can be developed.
- C++ is platform independent which is one of its design goals. This results in great level cross-platform independence and portability.

- C++ has simple classes for reading images from files of different formats. This is an important feature as reading input images is the basic requirement of project.

3.2.2 Future Plans

- Separate and more efficient compression techniques can be used for compressing audio and video files which will result in fewer amounts of data to be transmitted.
- Use of third party in exchanging the encryption keys will enhance the security of the system.
- After embedding filename, filesize and data to cover image, lossless images are obtained. Use of lossy images will minimize the amount of data to be sent across the network.

3.2.3 User Interface Paradigm

- Start page for user.
- After successfully starting, user chooses embed or extract.
- Sender chooses the cover image secret file and embeds filesize, filename and data.
- Receiver extracts the data from received stego images.

3.2.4 Error Detection and Recovery

- Hash codes such as md5 or SHA can be used to check integrity of data.
- Exceptional handling mechanism is used during execution of the application to handle run time errors.

3.3 System Architecture

This section provides a high-level overview of how the functionality and the responsibilities of the image steganography for secured data transmission is partitioned and then assigned to subsystems or the components or the modules appropriately.

The system architecture is shown in Fig 3.1 where the sender chooses the cover image, secret file and embeds the file size and filename followed by encrypted file onto cover images giving stego images which are in PPM/BMP format. The receiver on receiving the stego images extract the filesize first followed by filename and data and then decrypting..

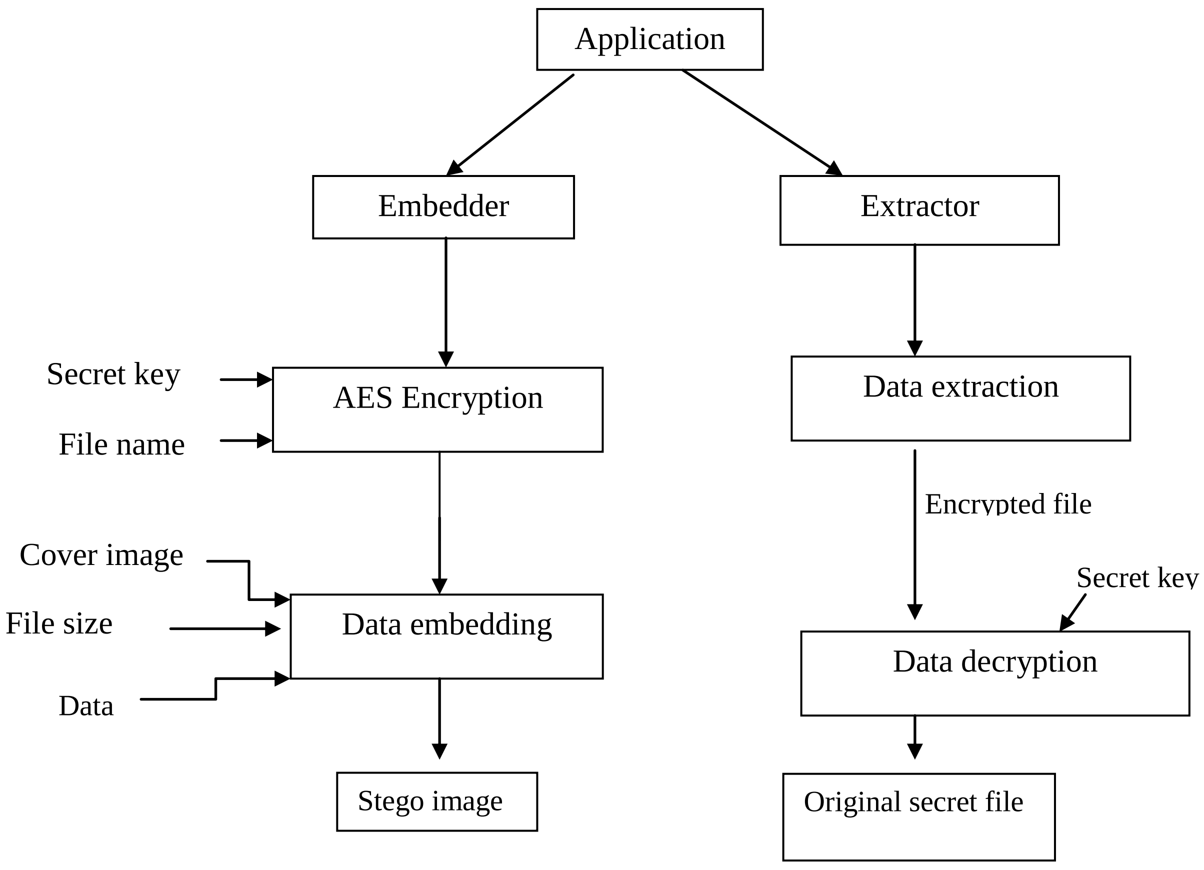


Fig 3.1 System Architecture

3.4 Data Flow Diagram

A data flow diagram shows the flow of data between the different processes and how data is modified in each of the process. There are various symbols to represent the various operations in DFD. There are symbols to represent processes, data stores, data flow directions and external entities.

There are large numbers of levels to represent the software. The level 0 DFD gives the general description and level 1 gives the deeper description. At higher levels of DFD, higher levels of descriptions will be given.

3.4.1 Data Flow Diagram- Level 0

Level 0 DFD gives the general operation of steganographic system as shown in Fig 3.2. There are 3 major components. Two external entities and one major system called the steganographic system. The two entities are sender and receiver.

- Sender: The sender sends information to the receiver. Sender embeds the data in the cover image and sends it over the network to the receiver.
- Receiver: The receiver receives the data sent from the sender and extracts the embedded data from the cover image. To do the extraction, the steganographic system is used.
- Steganographic system: It is the system that helps both sender and receiver in embedding and extracting the data and it acts as the heart of the system.

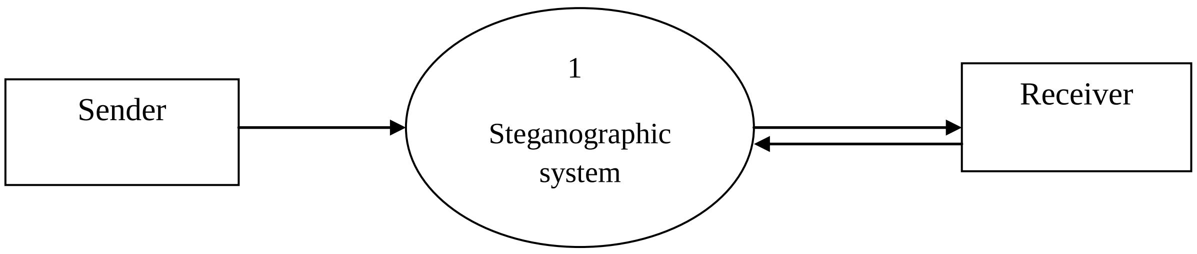


Fig 3.2 Level 0 DFD

3.4.2 Data Flow Diagram - Level 1

There are 3 major processes involved in this level. They are:

- Data embedding: The cover image is provided to hold the input data. The data to be embedded can be of any type like image, text, audio, video etc. For more security, encryption technique is used.
- Data extraction: This acts as the reverse process of embedding. The data embedded cover image is given as input to the system. Data extraction uses same algorithm used in embedding the data to extract the data.

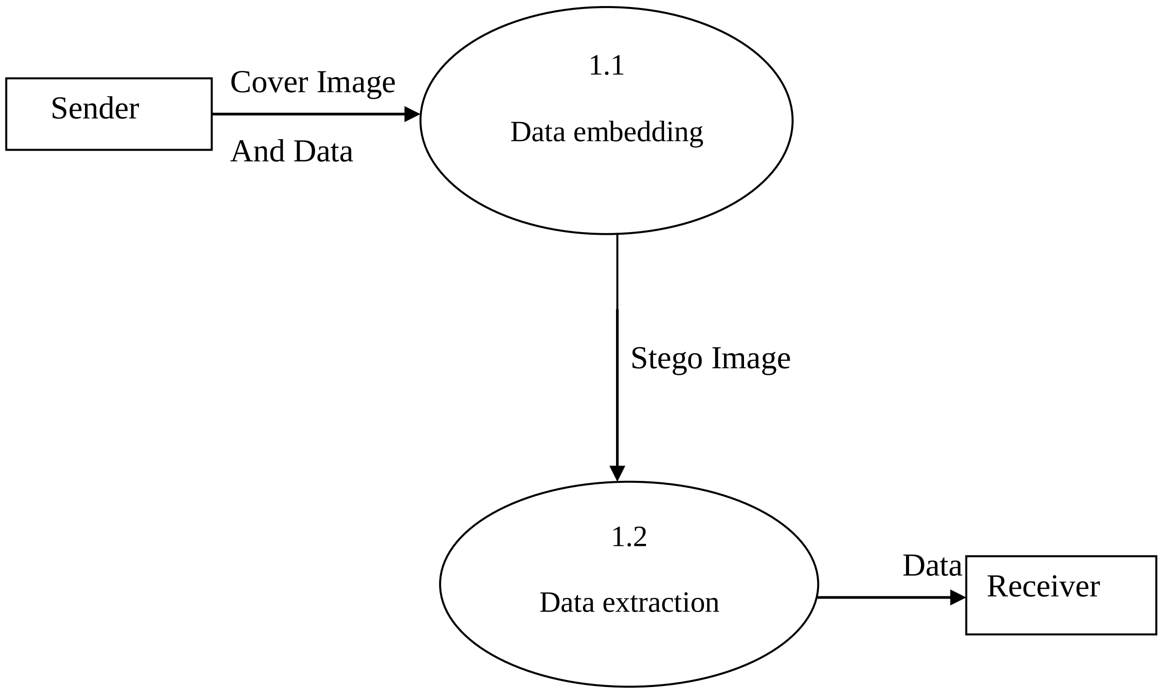


Fig 3.3 Level 1 DFD

3.4.3 Data Flow Diagram- Level 2

Level 2(a): Embedding

The process in level 1 is expanded here. There are three major operations involved:

- Embedding file size
- Embedding file name
- Embedding data

The receiver will not know the file format that is embedded in the stego image. Therefore this program embeds the file name of the secret data with its extension also. The file size of the secret file is also embedded.

The embedding process happens by a secure stenographic algorithm in spatial domain. In this algorithm the last four LSBs are used, but only one of those four bits of cover image are allowed to change. The bits to be changed are determined by a convolution decoder(3,1). This decoder helps to embed two bits of data of secret file in four LSBs of cover image.

The secret file chosen is encrypted using AES encryption as shown. The key used by the encryption is sent to the receiver by a reliable secure network. The cover image can be a jpeg, bmp or a ppm file. If jpeg is the cover image used then output stego image will be a ppm file. The secret file can be a file of any type a pdf, mp3 or a jpeg or any small file.

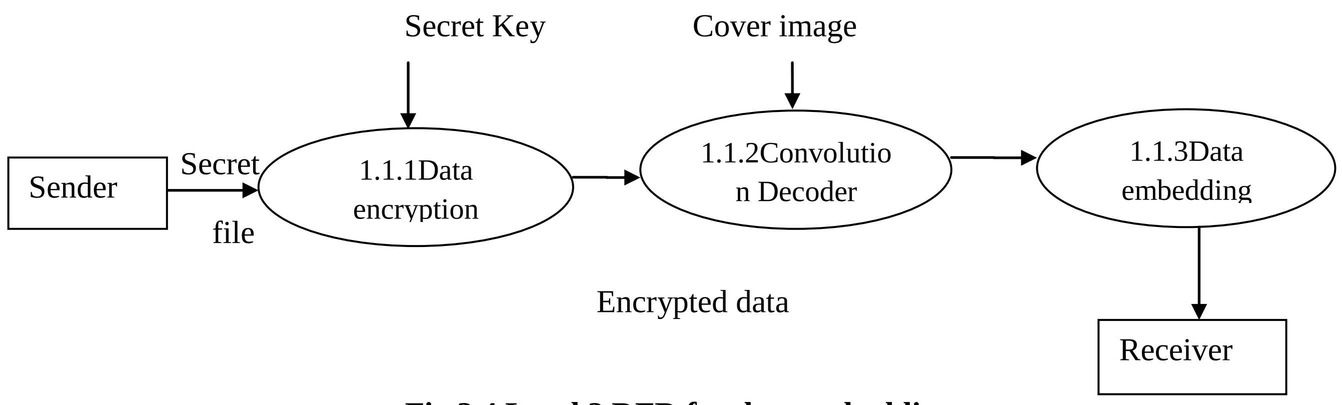


Fig 3.4 Level 2 DFD for data embedding

Level 2(b): Extraction

At the receiver end, the cover image containing the encrypted secret data is taken. Extraction process happens by XORing b0 and b1 of a pixel to get one bit and XORing b0 and b3 to get other bit. Hence two bits of data are retrieved.

Now the retrieved data is grouped into 8 bits and copied to a file. Since we have used AES encryption to encrypt the secret file the data we just obtained is not readable. Therefore we need to decrypt it. The same key used to encrypt should be used to decrypt, else the data cannot be read.

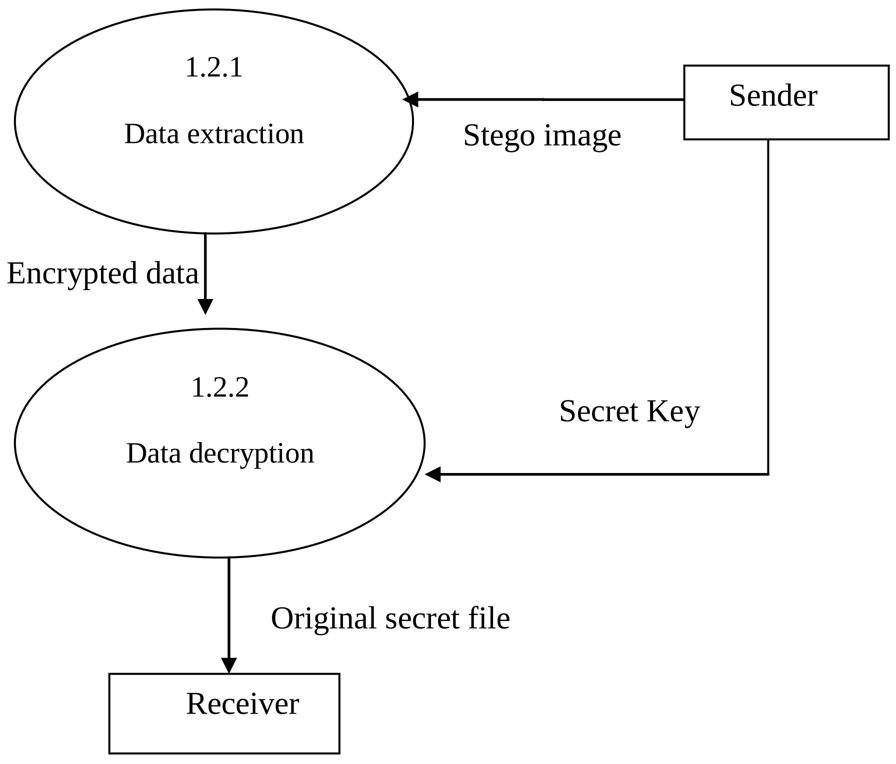


Fig 3.5 Level 2 DFD for data extracting

3.4.4 Data Flow Diagram- Level 3

Level 3(a): Data encryption

This program uses AES encryption of 128 bits to encrypt data. To do this first we take any length key from user and generate a md5 digest for that key to ensure that a constant length key of 16 bytes is used for encryption. Now in encryption phase we expand this key to 44 byte key to be used in further rounds of encryption. After various manipulations like shift rows and mixcoloums we finally XOR key with the resulting block and repeat same operations for 10 rounds. We process each and every 16 byte block this way and get the encrypted file.

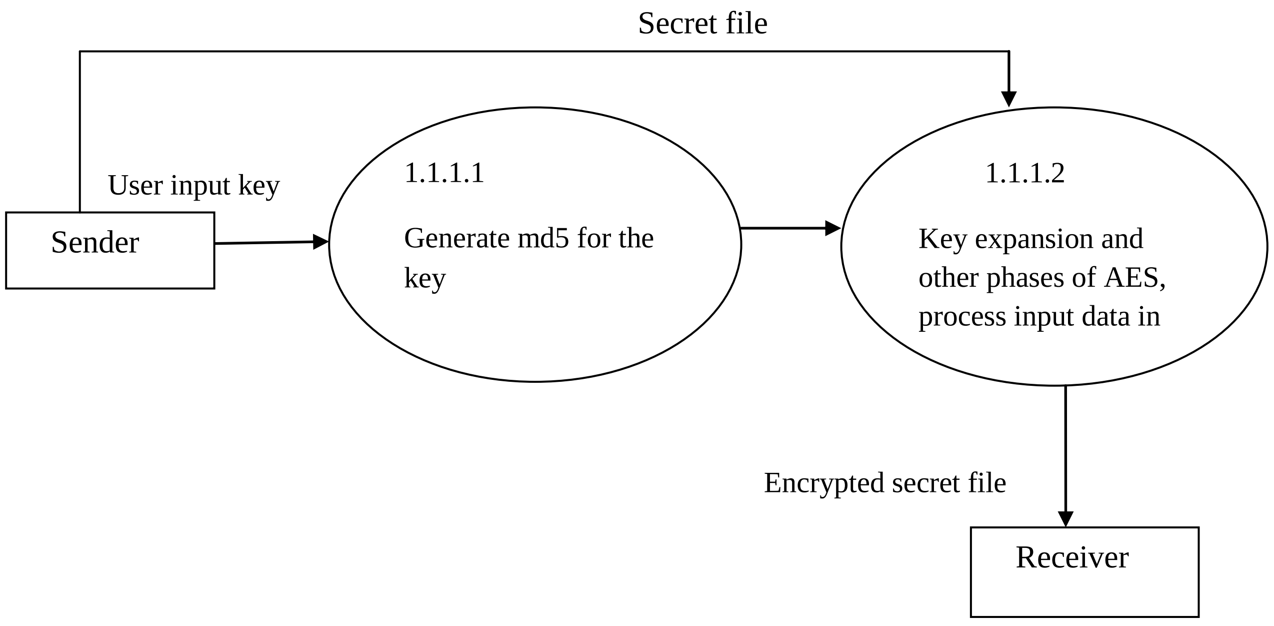


Fig 3.6 Level 3 DFD for data encryption

Level 3(b): Convolution decoder

The working of convolution decoder is as follows:

Four LSBs of cover images are taken. Two bits n2 and n3 are generated by XORing b0 with b1 and b3 respectively. Now two bits from the incoming bit stream from secret file are taken. These two bits are compared with n2 and n3 and how best can the four LSBs of cover images can be changed so as to make n2 and n3 equal to the incoming two bits is determined. This changed value of four LSBs is the stego pixel. We are allowed to manipulate only one bit out of four bits and store two bits of incoming data. At the maximum the color level can be changed by at most 8 values in any of the R,G,B components.

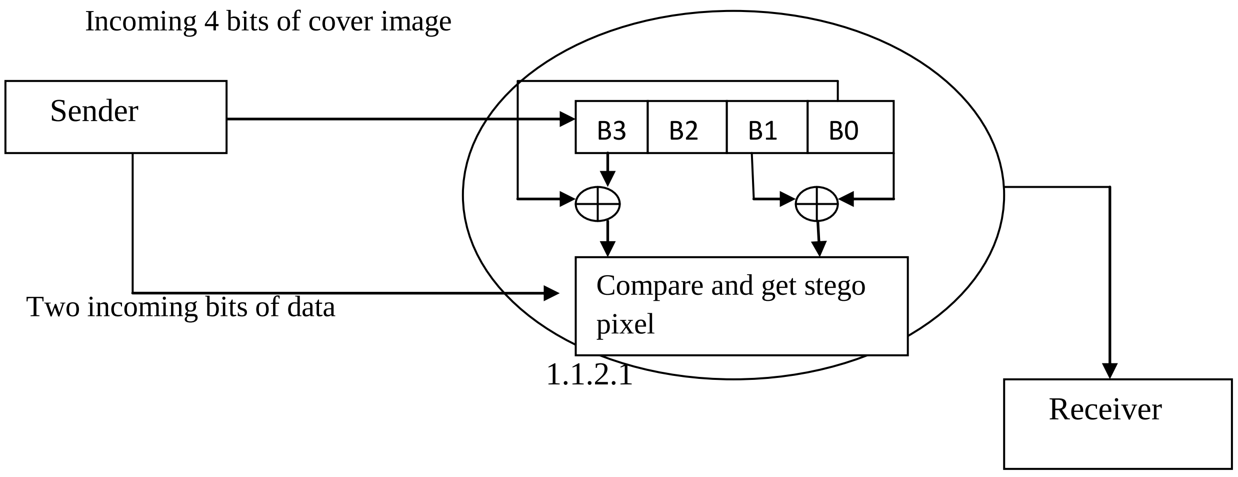

Incoming 4 bits of cover image
Sender
B3
B2
B1
B0
Two incoming bits of data
Compare and get stego pixel
1.1.2.1
Receiver

Fig 3.7 Level 3 DFD for convolution decoder process

Level 3(c): Data embedding

In data embedding first the size of the secret file is embedded in pixel values of R,G,B, values of first 5 pixels. The file size data undergoes the same data encoding procedure as mentioned earlier. Then the file name along with extension is embedded in R and G values only. The B value is used to indicate end of filename and from where actual data of the secret file starts. Then the data of secret file is embedded to form a stego image.

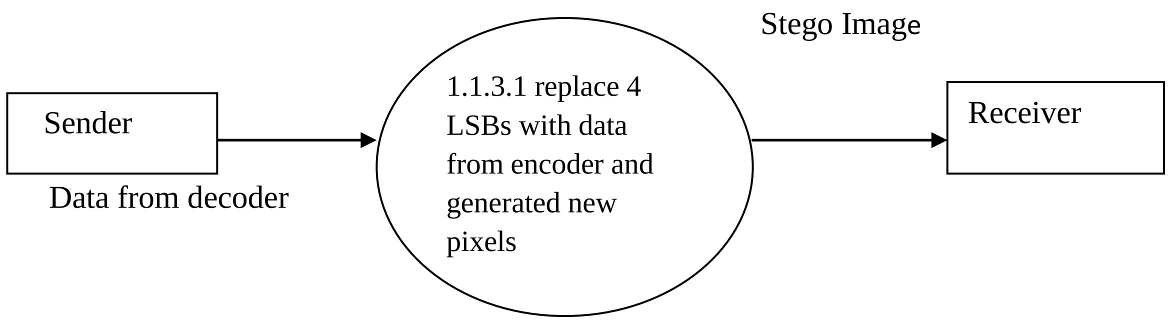

Stego Image
Sender
Data from decoder
1.1.3.1 replace 4 LSBs with data from encoder and generated new pixels
Receiver

Fig 3.8 Level 3 DFD for data embedding

Level 3(d): Data extraction

In data extraction we XOR b0 with b1 and b3 of the pixels in the stego image to get two bits of secret file. First we get the file size this way by extracting bits from first five pixels and then we get the filename of the embedded file by extracting bits from pixel values of R and G. After getting the file size and file name we proceed in extracting the data from stego image till we reach size of data reaches the value we obtained in file size. Then create a new file at receivers side with the obtained file name.

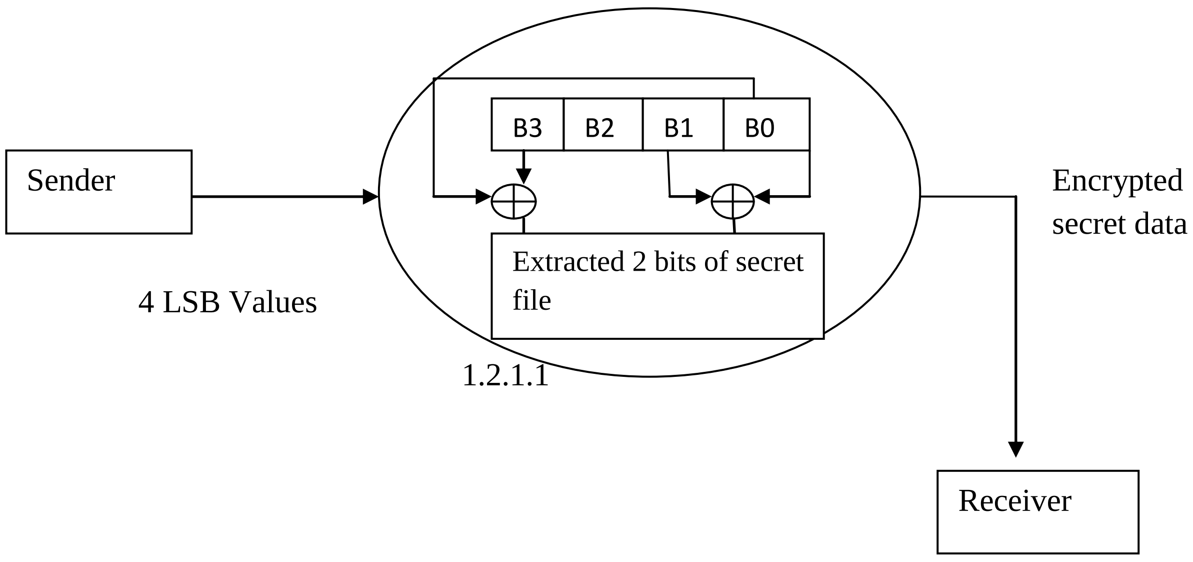


Fig 3.9 Level 3 DFD for data extraction

Level 3(e): Data decryption

Decryption part is similar to that of encryption. As AES is a symmetric key cipher we use the same key used in encryption to decrypt also. Usage of wrong keys to decrypt does not help as data produced is not readable. Hence this feature protects data that is embedded.

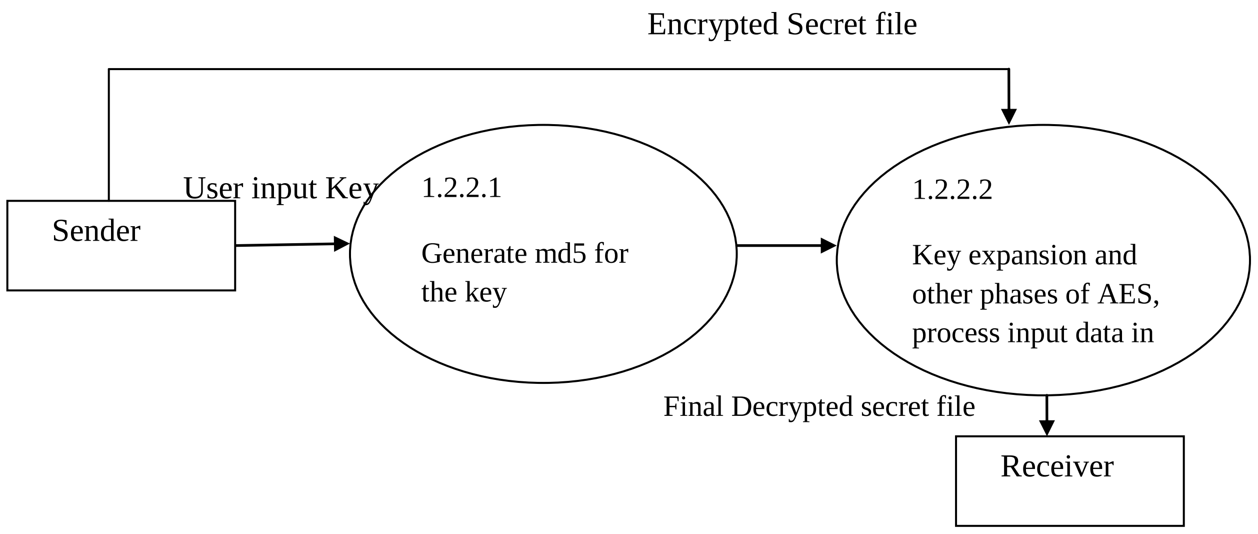

Encrypted Secret file
User input Key
Sender
1.2.2.1
Generate md5 for the key
1.2.2.2
Key expansion and other phases of AES, process input data in
Final Decrypted secret file
Receiver

Fig 3.10 Level 3 DFD for data decryption

Chapter 4

DETAILED DESIGN

4.1 Structure Chart

Structure charts are used to specify the high level design or architecture of a computer program. As a design tool, they help the programmer in dividing and conquering a large software problem, i.e. recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is called top-down design or functional decomposition. Programmers use a structure chart to build a program in a manner similar to how an architect uses a blueprint to build a house. In the design stage, the chart is drawn and used as a method for the client and various software designers to communicate. During the actual building of the program, the chart is continuously referred to as master plan. Often, it is modified as programmers learn new details about the program. After a program is completed, the structured chart is used to fix bugs and to make changes.

The entire program starts with user entering his choice of input as to either embed or extract. Based on this, the GUI module responds appropriately with success or failure. The image chosen by user is taken as input for embedding phase. The digital media is encrypted before embedding to stego image. Three additional bits n1,n2 and n3 are computed based on which the data can be embedded onto the cover image. This encrypted data is embedded onto image using embedding algorithm to get the stego image containing secret data.

At the receiver end, from the stego images, secret data is extracted. It will be in encrypted form. The original input data is extracted by using decryption algorithms.

The structural chart for data embedding and data extraction is shown in Fig 4.1.

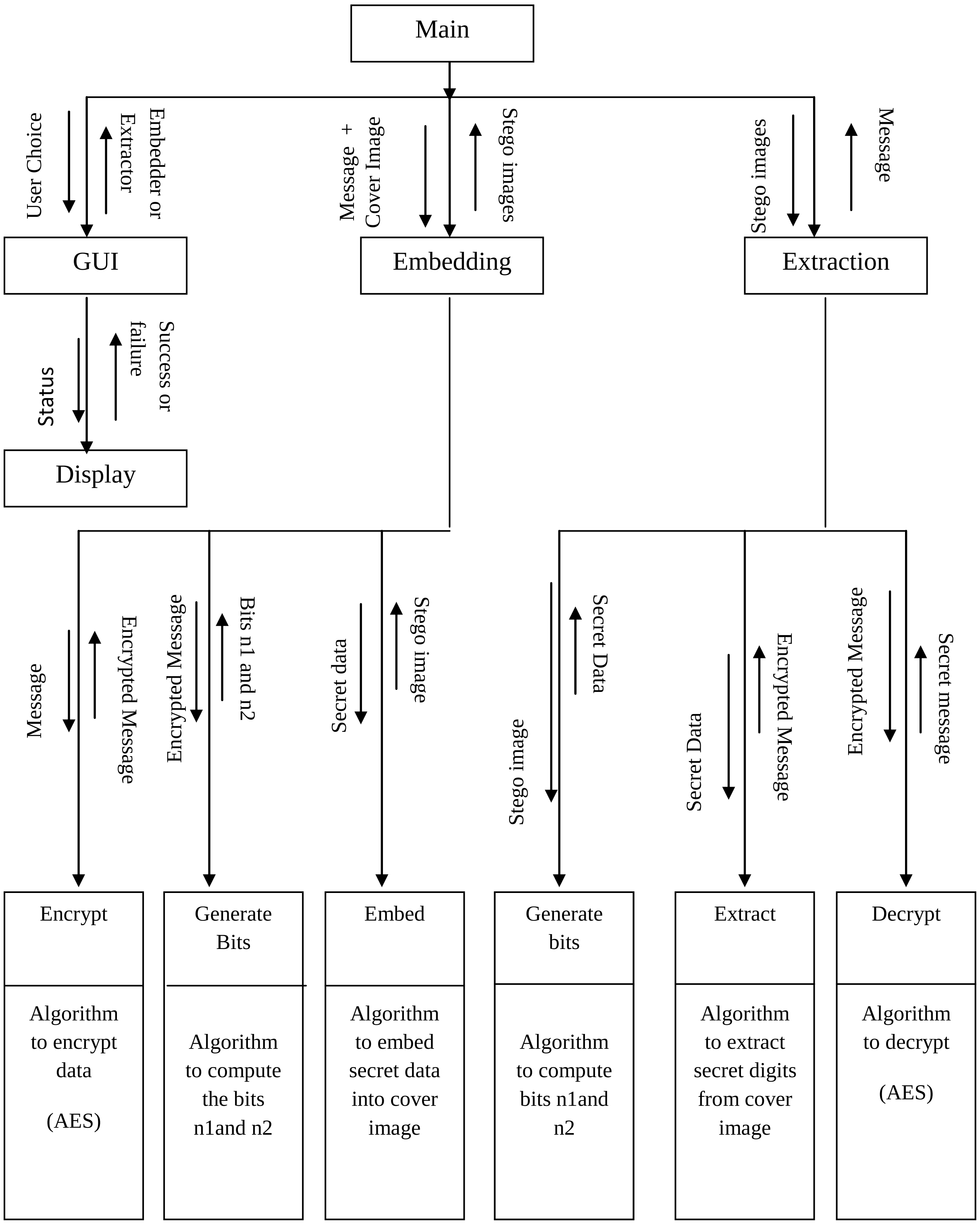


Fig 4.1 Structure chart

4.2 Functional Description of the Project

Definition:

This module is the core module which takes the users input to decide upon which operation to be performed as shown in Fig. Based upon user choice, the necessary inputs will be taken in this module. If any errors are generated during any operation, then suitable report is displayed to the user.

Resources:

The input files are different images like JPEG, PPM etc are chosen to embed data and cover image separately. If encryption is selected AES algorithm is used to encrypt the secret message using a key given by the user.

Functionality:

This is the main flowchart which shows all the functions provided by the software. In the beginning, the user is given three options, according to which the user either embeds extracts or exits from the application. This is shown in the decision symbol. According to the decision taken, the operations are performed. If the user chooses to embed, then the functions to select cover image, the secret data, and then embedding data and cover image are performed. Before embed secret data encryption is done. After performing these operations, the stego images are stored in the home directory. If the secret message in the stego image is to be extracted the user implements the functions to extract stego image values, extract secret digits data. After the implementation of these functions, if the user wants to exit from the software, that particular option is also provided to the user. This is shown in Fig 4.1.

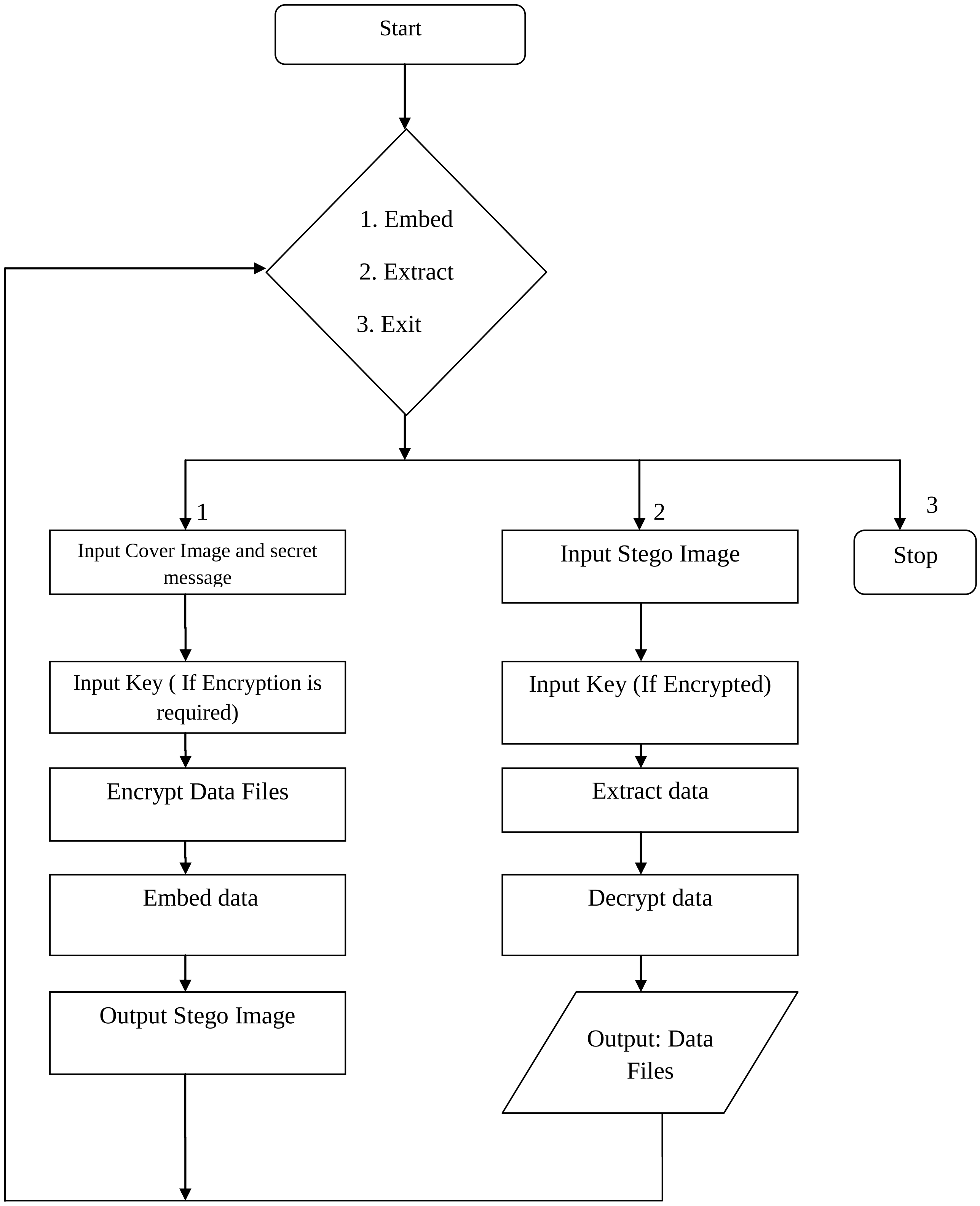

Start
1. Embed
2. Extract
3. Exit
1
2
3
Input Cover Image and secret message
Input Key (If Encryption is required)
Encrypt Data Files
Embed data
Output Stego Image
Input Stego Image
Input Key (If Encrypted)
Extract data
Decrypt data
Output: Data Files
Stop

Fig 4.1 GUI

Chapter 5

Implementation

The implementation phase of any project development is the most important phase as it yields the final solution, which solves the problem at hand. It is in this phase that the low level designs are converted into the language specific program such that they satisfy the requirements of the given software. The implementation phase involves the actual materialization of the ideas, which are expressed in the analysis document and developed in the design phase.

The technique used for implementing the software must support reusability, ease of maintenance and should be well documented. Implementation should be a perfect mapping of the design document in a suitable programming language in order to achieve the necessary final product. Often a product is ruined due to incorrect language chosen for implementation or an unsuitable method of programming. It is better for the coding phase do be directly linked to design phase in the sense if the design is in terms of object oriented way. The factors concerning the programming language and platform chosen are described in the next couple of sections.

5.1 Programming Language

Once the rapid prototype is developed, the language in which the final deliverable was to be written has to be decided. The necessary requirements are stated at the beginning of this section. C++ is a general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features. C++ is one of the most popular programming languages and is implemented on a wide variety of hardware and operating system platforms. As an efficient compiler to native code, its application domains include systems software, application software, device drivers, embedded software, high-performance server and client applications, and entertainment software such as video games. It is the optimal programming language for the project because of the following reason:

- C++ is an object oriented language and hence it can be used for functional development of product.
- Developing graphical user interface is very simple in C++ (Qt compiler). It has a rich set of classes and methods for creating forms, buttons, labels and events.

5.2 Challenges encountered and their solutions

There were a number of challenges that were faced while implementing the steganography using C++ Compilers. Some problems were challenging while trying to improve quality and efficiency of the software. Some of the major problems that were encountered have been stated in brief along with their solutions.

Problem 1: Initially the project was started off by trying to hide the data in JPEG images. The problem with the embedding of data in jpeg images was realized during the extraction of the data. It was then realized that jpeg images were lossy images, i.e., they compress the image changing the pixel values; as a result the data that was hidden in the image was completely lost.

Solution: The problem was solved by the use of PPM images since they are lossless images and hence no data is lost.

Problem 2: Project was designed in such a manner that any kind of data can be hidden behind the image. The data can either be an image, video or a text file. There are different kinds of extensions for each of the input data. It could either be a txt, java, jpeg, png, wmv etc. The major problem that occurs here is that the receiver will not be aware of the kind of file that is hidden.

Solution: This problem is easily solved by hiding the name of the data file that is hidden. The name is hidden before the embedding of the data. The receiver then extracts the extension from the image and adds the extension to the data extracted.

Problem 3: The receiver was not aware of the file size that is hidden in image. Hence he did not know how much data to extract from the stego image.

Solution: This problem can be solved by embedding the information regarding the file size in the image itself. Then the receiver will know how much to extract from the stego image.

Problem 4: Since a file may be sent across the network, there is a chance that attacker may get the file and find the content inside the file. There was a security threat faced by the system.

Solution: This problem is solved by encrypting the file before sending. Even if the third party gets the file, since it is encrypted, he will not be able to access the contents of the file.

Chapter 6

SOFTWARE TESTING

6.1 Testing Environment

Software testing is process used to determine the quality of software as specified by the requirements. Testing mainly verifies and validates the system as specified by the clients. Testing is an empirical technical investigation conducted to provide clients with information about the quality of the product or service under test, with respect to the context in which it is intended to operate. This includes, but is not limited to the process of executing a program or application with the intent of finding software bugs. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification.

The proposed software mainly deals with generation of stenographic images using a cover image. So this program should be able to embed any piece of secret data be it a .jpg image, audio file, a pdf file or even a .exe executable file. We test the program by providing each of the above mentioned formats.

6.2 System testing

System testing is testing where all modules which are tested by Integrated testing are combined to form single system .System is tested such that all the units are linked properly to satisfy user specified requirement. This test helps in removing the overall bugs and improves quality and assurance of the system. The Proper functionality of the system is concluded in system testing.

6.2.1 System testing case 1

Input: A .pdf file as the secret file and a .bmp image as the cover image.

Expected output : A generated stego image that is similar to cover image but with embedded data.

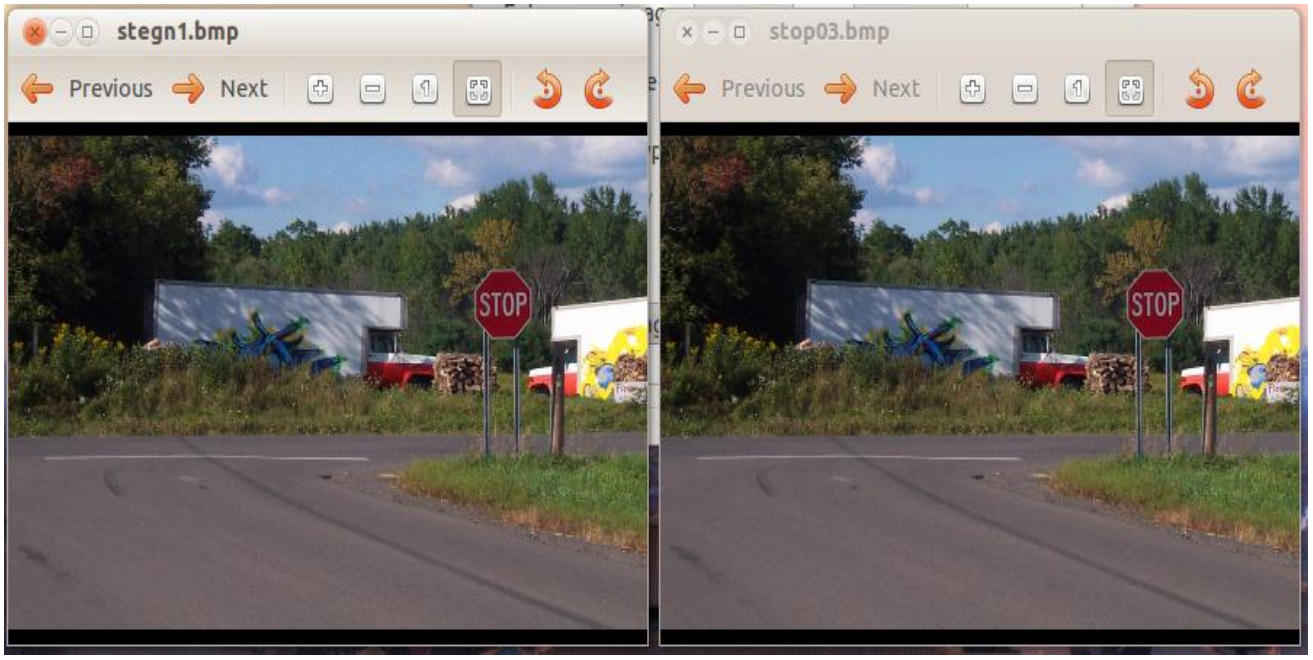


Fig 6.1: stego image and its cover image after embedding pdf file

Extraction: This Generated stego image is given as input to the extraction part and original pdf file is expected as the output.

Output : We get the exact copy of the embedded pdf file created with the same file name as the original copy and the same size and checksum indicating no loss or change of data.

6.2.2 System testing case 2

Input: A jpeg image file with AES encryption as secret file and a .ppm as cover image

Expected output: A stego image with the embedded data, which requires a key to display original data on extraction.

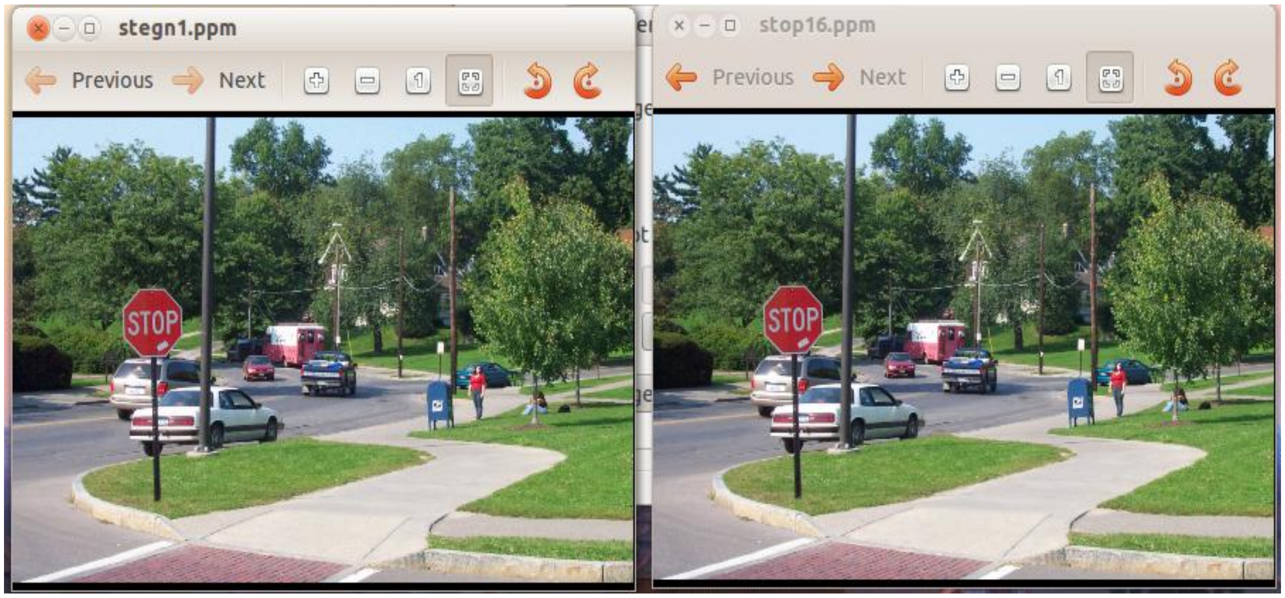


Fig 6.2: Image containing encrypted jpg data with its cover image

Extraction: This Generated stego image is given as input to the extraction part and original jpeg image file is expected as the output. The key to be required in order to decrypt the data must also be compulsorily provided.

Output: We get the exact jpeg image that we embedded on extracting and decrypting. If the wrong key is given then the data obtained is not readable and hence avoids eavesdropping. Hence the data can be retrieved only when the exact key is given. We can also verify the integrity of data by checksum.

6.2.3 System testing case 3

Input: A .mp3 audio file as the secret file and a .ppm image as the cover image.

Expected output : A generated stego image that is similar to cover image but with embedded data.

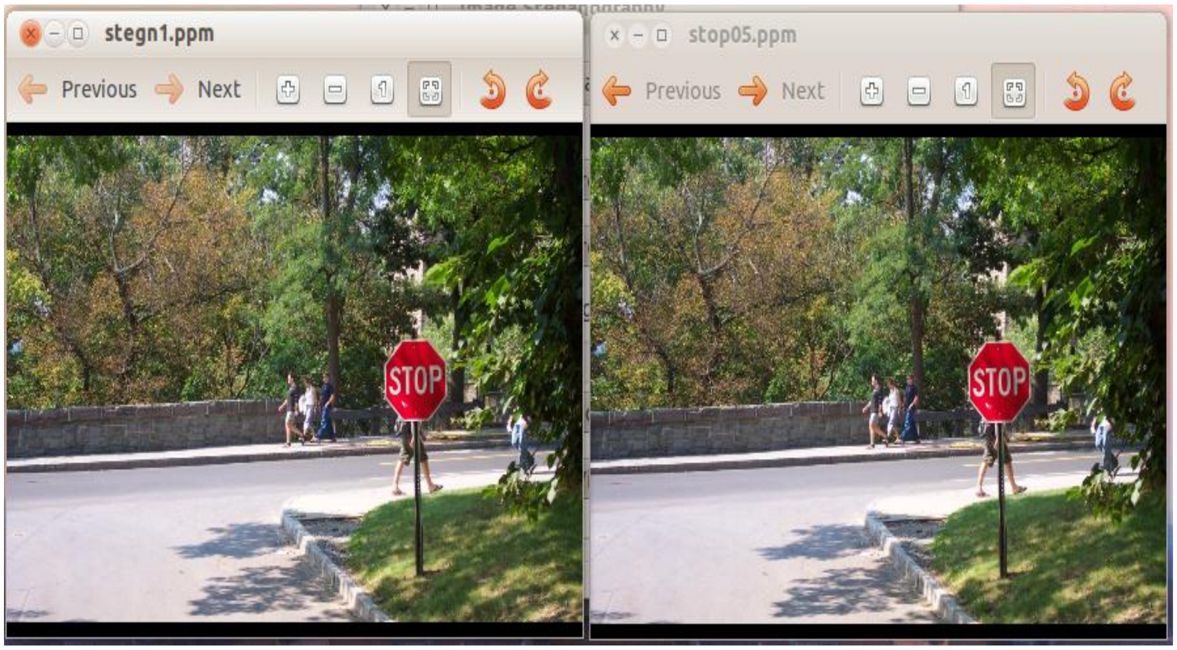


Fig 6.3: Image containing mp3 audio file with its cover image

Extraction: This Generated stego image is given as input to the extraction part and original mp3 file is expected as the output.

Output : We get the exact copy of the embedded mp3 audio created with the same file name as the original copy and the same size and checksum indicating no loss or change of data.

6.2.4 System testing case 4

Input: A executable .exe file with AES encryption as secret file and a .bmp as cover image

Expected output: A stego image with the embedded data, which requires a key to display original data on extraction.

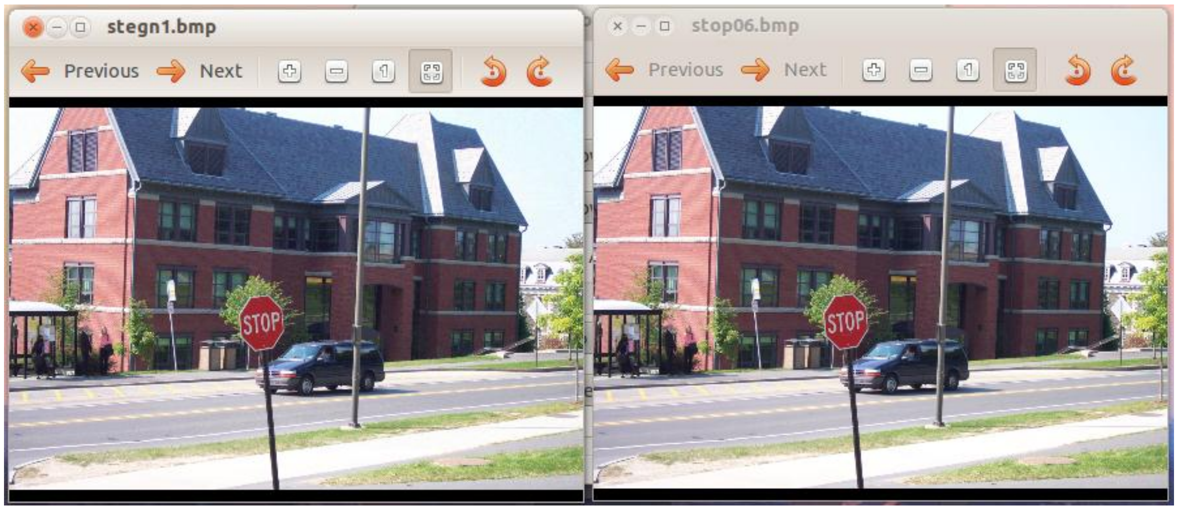


Fig 6.4: Image containing encrypted executable file with its cover image

Extraction: This Generated stego image is given as input to the extraction part and original binary executable file is expected as the output. The key to be required in order to decrypt the data must also be compulsorily provided.

Output: We get the exact executable file that we embedded on extracting and decrypting. If the wrong key is given then the data obtained is not readable and hence avoids eavesdropping. Hence the data can be retrieved only when the exact key is given. We can also verify the integrity of data by checksum.

6.3 Experimental Results

While designing image steganography, main factors to be considered are

- Embedding more data securely in a cover image.
- Number bits to be embedded per pixel.
- Less distortion in embedded cover image compared to original cover image.

The number of bits embedded per pixel is 6 bits which achieved by using R, G and B components of each cover image pixel. The distortion in cover image depends upon the change in value of pixels and number of pixels of cover image used for embedding which are in turn depending upon the number of components of the pixel used and amount of input data.

PSNR is used to evaluate the quality of an image . The PSNR is defined as in eqn (1):

$$\mathrm{PSNR} = 10.\log_{10} \frac{(2^{\wedge}24-1)^2}{\mathrm{MSE}} \ \mathrm{dB} \qquad \text{eqn (1)}$$

where MSE is the mean square error between the original image and the stego image.

The MSE is defined as in eqn (2):

$$\mathrm{MSE} = \frac{1}{(3xMxN)} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (R_{ij}-R_{ij}^{|})^2 + (G_{ij}-G_{ij}^{|})^2 + (B_{ij}-B_{ij}^{|})^2 \qquad \text{eqn (2)}$$

where R_{ij}, G_{ij}, B_{ij} and $R_{ij}^{|}$, $G_{ij}^{|}$, $B_{ij}^{|}$ are RGB value of the pixel of the original and the stego image respectively, MxN gives number of pixels present in the cover image. A larger PSNR indicates that the quality of the stego image is closer to the original one. Normally, human's eyes find it hard to distinguish between the distortions on a stego image compared to original image when its PSNR value is greater than 30 dB.

The PSNR values given by the system for variable percentage of different cover image of resolution 1024x768 used for embedding are shown in Table 6.1. So if almost all the pixels of original cover image (93.47%) are used for embedding then value of PNSR is equal to 133.894 dB, which is higher than threshold value (30dB) required.

Percentage of Image Used	PSNR FOR IMAGE 1024X768
22.66%	138.589340
26.01%	137.963409
70.18%	133.646057
93.47%	133.894882

Table 6.1: Percentage of image used and respective PSNR values.

Chapter 7

Conclusion

An overview of the design and development of a Steganography system is presented in this report. Data hiding techniques embed the important data into multimedia data such as images, videos or sounds. Digital images are considered good cover carriers because of their insensitivity to human visual system. It embeds the secret message in the cover image to hide the existence of the message. Image Steganography is often used in secret communication.

7.1 Summary

Steganography is the method of hiding the presence of data in cover images. In this work, RED, GREEN& BLUE components of cover image pixel are used. Prior to embedding, data encryption is done, so that variable types of data are securely embedded in cover image.

The image used in the proposed method is a 24-bit colored image. Initially, the data to be hidden is chosen which can be any digital media file such as text, image, audio, video etc. In this method we add 6 bits of information in a pixel and alter one bit from one bit plane but the message does not necessarily place in the least significant bit of pixel and second less significant bit plane and fourth less significant bit plane can also host the massage. Since in our method for embedding two bits message we alter just one bit plane, fewer pixels would be manipulated during embedding message in an image and it is expected for the steganalysis algorithm to have more difficulty detecting the covert communication. Before embedding, one or more media files are encrypted to increase the efficiency and security of the method.

7.2 Limitations

- No compression techniques used for audio and video files. So the compression achieved in case of audio and video files are not efficient.
- Other encryption techniques can further increase the security of the data.

7.3 Future Enhancements

Any development is a continuous process and does not conclude once an executable has been generated. Every project has certain limitations due to various reasons. Some of these limitations could be overcome, given sufficient time while others could be overcome with better technologies. Hence, some of the further enhancements that could be performed on our Steganography system as follow:

- Separate and more efficient compression techniques can be used for compressing files which will result in fewer amounts of data to be embedded.
- While embedding data to cover image, lossless images are used. Use of lossy images will minimize the amount of data to be sent across the network.

Chapter 8

References:

[1] 2011 Second International Conference on Intelligent Systems, Modelling and Simulation A More Secure Steganography Method in Spatial Domain Ali Daneshkhah, Hassan Aghaeinia, Seyed Hamed Seyedi

[2] International Journal of Computer Applications (0975 – 8887) Volume 2 – No.3, May 2010,41 A Comparative Analysis of Image Steganography R.Amirtharajan et al.

[3] New Genetic Algorithm Approach for Secure JPEG Steganography Amin Milani Fard.

[4] Niels Provos and Peter Honeyman, Hide and seek: An introduction to steganography, IEEE Security and Privacy, vol. 1, no.3, pp. 32-44, 2003.

[5] R. Chandramouli, M. Kharrazi, and N. Memon, Image steganography and steganalysis concepts and practice, Proc. of IWDW'03, vol. 2939, pp. 35-49, Springer, 2003.

[6] An Overview of image steganography. T. Morkel , J.H.P. Eloff , M.S. Olivier

[7] Fahim Irfan Alam et al. / International Journal of Engineering Science and Technology (IJEST) An Investigation into Encrypted Message Hiding Through Images Using LSB ISSN : 0975-5462 Vol. 3 No. 2 Feb 2011

Appendix A

Source Code Listing

A.) Steganography code

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include<string.h>
#define CREATOR "yespee_4444@hotmail.com"
#define RGB_COMPONENT_COLOR 255
int count = 0;
char buf[513];
unsigned char bit8[32];
typedef struct
{
unsigned char red, green, blue;
} PPMPixel;
typedef struct
{
int x, y;

PPMPixel * data;
} PPMImage;
void
loadbit8 (char h,int offset)
{
unsigned char k;
int i;
unsigned char j = 0x80;
for (i = sizeof (char) * 8; i > 0; i--)
    {
k = h & j;
k = k >> (i - 1);
bit8[offset+(sizeof (char) * 8 - i)] = k;
j = j >> 1;
}
}
static PPMImage *

readPPM (const char *filename)
{
 char buff[16];
 PPMImage * img;
 FILE * fp;
 int c, rgb_comp_color;
```

```
      //open PPM file for reading
    fp = fopen (filename, "rb");
 if (!fp)
        {
 fprintf (stderr, "Unable to open file '%s'\n", filename);
exit (1);
}
     if (!fgets (buff, sizeof (buff), fp))
    {
 perror (filename);
exit (1);

}

    if (buff[0] != 'P' || buff[1] != '6')
    {
fprintf (stderr, "Invalid image format (must be 'P6')\n");
exit (1);

}


//alloc memory form image
    img = (PPMImage *) malloc (sizeof (PPMImage));
if (!img)
    {

fprintf (stderr, "Unable to allocate memory\n");
exit (1);

}

    c = getc (fp);

while (c == '#')

    {


while (getc (fp) != '\n');
c = getc (fp);
}

ungetc (c, fp);
    if (fscanf (fp, "%d %d", &img->x, &img->y) != 2)

    {
fprintf (stderr, "Invalid image size (error loading '%s')\n",filename);

exit (1);
```

```
}

   if (fscanf (fp, "%d", &rgb_comp_color) != 1)
   {
fprintf (stderr,"Invalid rgb component (error loading '%s')\n",filename);
exit (1);
}
   if (rgb_comp_color != RGB_COMPONENT_COLOR)
   {

fprintf (stderr, "'%s' does not have 8-bits components\n",filename);

exit (1);
}

while (fgetc (fp) != '\n');
   img->data = (PPMPixel *) malloc (img->x * img->y * sizeof (PPMPixel));
if (!img)

   {
fprintf (stderr, "Unable to allocate memory\n");

exit (1);

}

   if (fread (img->data, 3 * img->x, img->y, fp) != img->y)
   {
fprintf (stderr, "Error loading image '%s'\n", filename);

exit (1);

}
fclose (fp);
return img;

}
void
writePPM (const char *filename, PPMImage * img)
{

FILE * fp;
   fp = fopen (filename, "wb");

if (!fp)

   {

fprintf (stderr, "Unable to open file '%s'\n", filename);
exit (1);
```

```
}
    fprintf (fp, "P6\n");
    fprintf (fp, "# Created by %s\n", CREATOR);
    fprintf (fp, "%d %d\n", img->x, img->y);
    fprintf (fp, "%d\n", RGB_COMPONENT_COLOR);
    fwrite (img->data, 3 * img->x, img->y, fp);
fclose (fp);
}

 unsigned char decode(unsigned char temp,int bcount)
{
unsigned char n2 = 0x00,n3 = 0x00,temp2 = temp,k = 0x01;
n2 = (temp & 0x01) ^ ((temp & 0x02) >> 1);
n3 = (temp & 0x01) ^ ((temp & 0x08) >> 3);
while((n2 != bit8[bcount]) || (n3 != bit8[bcount+1]))
{
temp2 = temp ^ k;
n2 = (temp2 & 0x01) ^ ((temp2 & 0x02) >> 1);
n3 = (temp2 & 0x01) ^ ((temp2 & 0x08) >> 3);
k = k << 1;
}
temp = temp2;
return temp;
}
void changeColorPPM (PPMImage * img, int fd,off_t fsize,char fname[20])
{

int i=0, j=0,k=0,n,bitcount = 0;
  off_t temp = 1;
 if (img)
     {
for(int offtcount = 0 ; offtcount < sizeof(off_t)*8;offtcount++)
{
bit8[offtcount] = (unsigned char)((fsize & temp) >> offtcount);
temp = temp << 1;
}
for(bitcount = 0; bitcount < 32; bitcount += 2){
img->data[i * img->x + j].red = decode(img->data[i * img->x +
j].red,bitcount);bitcount += 2;
if(bitcount == 32) break;
img->data[i * img->x + j].green = decode(img->data[i * img->x +
j].green,bitcount);bitcount += 2;
img->data[i * img->x + j].blue = decode(img->data[i * img->x + j].blue,bitcount);
j++;
}
j = 6;
for(k = 0; fname[k] != '\0';k++)
buf[k] = fname[k];
buf[k] = '\0';
```

```
for (int m = 0;buf[m] != '\0';m++)
{
loadbit8(buf[m],0);
img->data[i * img->x + j].red = decode(img->data[i * img->x +
].red,bitcount);bitcount += 2;
img->data[i * img->x + j].green = decode(img->data[i * img->x +
j].green,bitcount);bitcount += 2;
img->data[i * img->x + j].blue &= 0xfe;
j++;
if(j >= img->x)
{
i++;
j = 0;
}
img->data[i * img->x + j].red = decode(img->data[i * img->x +
j].red,bitcount);bitcount += 2;
img->data[i * img->x + j].green = decode(img->data[i * img->x +
j].green,bitcount);bitcount += 2;
img->data[i * img->x + j].blue &= 0xfe;
j++;
if(j >= img->x)
{
i++;
j = 0;
}
bitcount = 0;
}
j--;
if(j < 0)
{
i--;
j = img->x - 1;
}
 img->data[i * img->x + j].blue |= 0x01;
j++;
if(j >= img->x)
{
i++;
j = 0;
}

while( (n = read(fd,buf,sizeof(buf))) > 0)
for (int m = 0;m < n;m += 3)
{
loadbit8(buf[m],0);
loadbit8(buf[m+1],8);
loadbit8(buf[m+2],16);
for(bitcount = 0; bitcount < 24; bitcount += 2){
img->data[i * img->x + j].red = decode(img->data[i * img->x +
j].red,bitcount);bitcount += 2;
img->data[i * img->x + j].green = decode(img->data[i * img->x +
j].green,bitcount);bitcount += 2;
img->data[i * img->x + j].blue = decode(img->data[i * img->x + j].blue,bitcount);
j++;
if(j >= img->x)
{
i++;
j = 0;
}

}

}

}
}
int
main (int argc, char *argv[])
{
  char systm[20] = {'\0'};
PPMImage * image;
int fd;
  struct stat statv;
stat(argv[3],&statv);

if (argc != 4)
printf("usage: ./a.out <inputfile> <outputfile> <secretfile>\n");
  else
    {

fd = open (argv[3], O_RDONLY);

image = readPPM (argv[1]);
     char *basnam = basename(argv[3]);
changeColorPPM (image, fd,statv.st_size,basnam);

writePPM (argv[2], image);
       close(fd);
sprintf(systm,"eog %s",argv[2]);
system(systm);
    }
}
```

B.) Inverse Steganography

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define CREATOR "yespee_4444@hotmail.com"
#define RGB_COMPONENT_COLOR 255

off_t count = 0;
int n = 0;
char buf[513];
unsigned char bit8[32];
char fname[20];
typedef struct
{

unsigned char red, green, blue;
} PPMPixel;

typedef struct
{
int x, y;
PPMPixel * data;
} PPMImage;

static PPMImage *

readPPM (const char *filename)
{
 char buff[16];

PPMImage * img;

FILE * fp;

int c, rgb_comp_color;
      //open PPM file for reading
    fp = fopen (filename, "rb");

if (!fp)
        {

fprintf (stderr, "Unable to open file '%s'\n", filename);
exit (1);
}
```

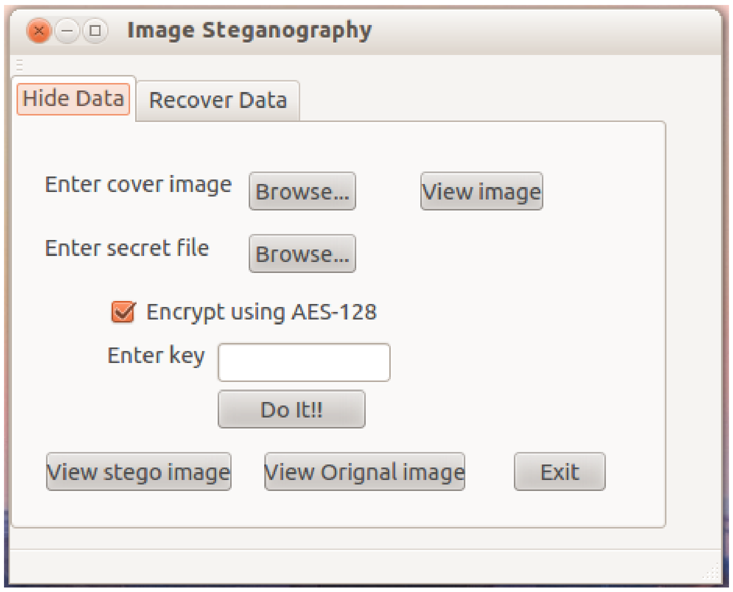
```
    if (!fgets (buff, sizeof (buff), fp))
    {

perror (filename);
exit (1);

}

    if (buff[0] != 'P' || buff[1] != '6')
    {
fprintf (stderr, "Invalid image format (must be 'P6')\n");
exit (1);

}


//alloc memory form image
    img = (PPMImage *) malloc (sizeof (PPMImage));
if (!img)
    {

fprintf (stderr, "Unable to allocate memory\n");
exit (1);

}

    c = getc (fp);

while (c == '#')

    {


while (getc (fp) != '\n');
c = getc (fp);
}

ungetc (c, fp);
    if (fscanf (fp, "%d %d", &img->x, &img->y) != 2)

    {
fprintf (stderr, "Invalid image size (error loading '%s')\n",filename);

exit (1);
}

    if (fscanf (fp, "%d", &rgb_comp_color) != 1)
    {
fprintf (stderr,"Invalid rgb component (error loading '%s')\n",filename);
exit (1);
}
   if (rgb_comp_color != RGB_COMPONENT_COLOR)
   {

fprintf (stderr, "'%s' does not have 8-bits components\n",filename);

exit (1);
}

while (fgetc (fp) != '\n');
   img->data = (PPMPixel *) malloc (img->x * img->y * sizeof (PPMPixel));
if (!img)

   {
fprintf (stderr, "Unable to allocate memory\n");

exit (1);

}

   if (fread (img->data, 3 * img->x, img->y, fp) != img->y)
   {
fprintf (stderr, "Error loading image '%s'\n", filename);

exit (1);

}
fclose (fp);
return img;

}
void
writePPM (const char *filename, PPMImage * img)
{

FILE * fp;
   fp = fopen (filename, "wb");

if (!fp)

   {

fprintf (stderr, "Unable to open file '%s'\n", filename);
exit (1);

}
   fprintf (fp, "P6\n");
   fprintf (fp, "# Created by %s\n", CREATOR);
   fprintf (fp, "%d %d\n", img->x, img->y);
   fprintf (fp, "%d\n", RGB_COMPONENT_COLOR);
   fwrite (img->data, 3 * img->x, img->y, fp);
fclose (fp);
}
void addtobuff(unsigned char k)
{
unsigned char h;
int i;
 h = '\0';
for (int m = 0; m < sizeof (char) * 8; m++)
h = h | (bit8[m] << (sizeof (char) * 8 - 1 - m));
fname[n++] = h;
if(k == 0x01)
fname[n] = '\0';
}
int addtobuff(int fd,off_t fsize)
{

unsigned char h;
 h = '\0';
for (int m = 0; m < sizeof (char) * 8; m++)
h = h | (bit8[m] << (sizeof (char) * 8 - 1 - m));
buf[n++] = h;
if(n == sizeof(buf))
{
write(fd,buf,n);
n = 0;
}
if(count == fsize+1){
write(fd,buf,n-1);
close(fd);
return 1;
}
return 0;
}
void
changeColorPPM (PPMImage * img, int fd)
{
int i = 0, j = 0, bitcount = 0;
 off_t fsize = 0;

unsigned char k = 0x00;

if (img)
   {
while(1)
{
bit8[bitcount++] = (img->data[i * img->x + j].red & 0x01) ^ ((img->data[i * img->x
+ j].red & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].red & 0x01) ^ ((img->data[i * img->x
+ j].red & 0x08) >> 3);
```

```
if(bitcount == 32) break;
bit8[bitcount++] = (img->data[i * img->x + j].green & 0x01) ^ ((img->data[i * img-
>x + j].green & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].green & 0x01) ^ ((img->data[i * img-
>x + j].green & 0x08) >> 3);
bit8[bitcount++] = (img->data[i * img->x + j].blue & 0x01) ^ ((img->data[i * img->x
+ j].blue & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].blue & 0x01) ^ ((img->data[i * img->x
+ j].blue & 0x08) >> 3);
j++;
}
for(int offtcount = sizeof(off_t)*8-1 ; offtcount >= 0;offtcount--)
fsize = (fsize << 1) | bit8[offtcount];

j = 6;
bitcount = 0;
n=0;

while(k == 0x00)
{


bit8[bitcount++] = (img->data[i * img->x + j].red & 0x01) ^ ((img->data[i * img->x
+ j].red & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].red & 0x01) ^ ((img->data[i * img->x
+ j].red & 0x08) >> 3);
bit8[bitcount++] = (img->data[i * img->x + j].green & 0x01) ^ ((img->data[i * img-
>x + j].green & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].green & 0x01) ^ ((img->data[i * img-
>x + j].green & 0x08) >> 3);
k = 0x01 & img->data[i * img->x + j].blue;
if(bitcount == 8) { bitcount = 0;addtobuff(k);}
j++;
if(j >= img->x)
{
i++;
j = 0;
}
}
n=0;
bitcount = 0;

  for (i; i < img->y; i++)
 {
for (j; j < img->x; j++)
          {

bit8[bitcount++] = (img->data[i * img->x + j].red & 0x01) ^ ((img->data[i * img->x
+ j].red & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].red & 0x01) ^ ((img->data[i * img->x
+ j].red & 0x08) >> 3);
if(bitcount == 8) {count++; bitcount = 0;if(addtobuff(fd,fsize)) return;}
bit8[bitcount++] = (img->data[i * img->x + j].green & 0x01) ^ ((img->data[i * img-
>x + j].green & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].green & 0x01) ^ ((img->data[i * img-
>x + j].green & 0x08) >> 3);
if(bitcount == 8) {count++; bitcount = 0;if(addtobuff(fd,fsize)) return;}
bit8[bitcount++] = (img->data[i * img->x + j].blue & 0x01) ^ ((img->data[i * img->x
+ j].blue & 0x02) >> 1);
bit8[bitcount++] = (img->data[i * img->x + j].blue & 0x01) ^ ((img->data[i * img->x
+ j].blue & 0x08) >> 3);
if(bitcount == 8) {count++; bitcount = 0;if(addtobuff(fd,fsize)) return;}
}
 j=0;
}
}
 }
int
main (int argc, char *argv[])
{
PPMImage * image;

int fd, n;

if (argc != 2)

printf ("usage: ./a.out <inputfile>\n");
  else

    {
fd = open ("temp", O_WRONLY | O_CREAT|O_TRUNC, 0777);
image = readPPM (argv[1]);
changeColorPPM (image, fd);
fname[0] = 'k';
printf("%s",fname);
rename("temp",fname);
    }
}
```

Appendix B

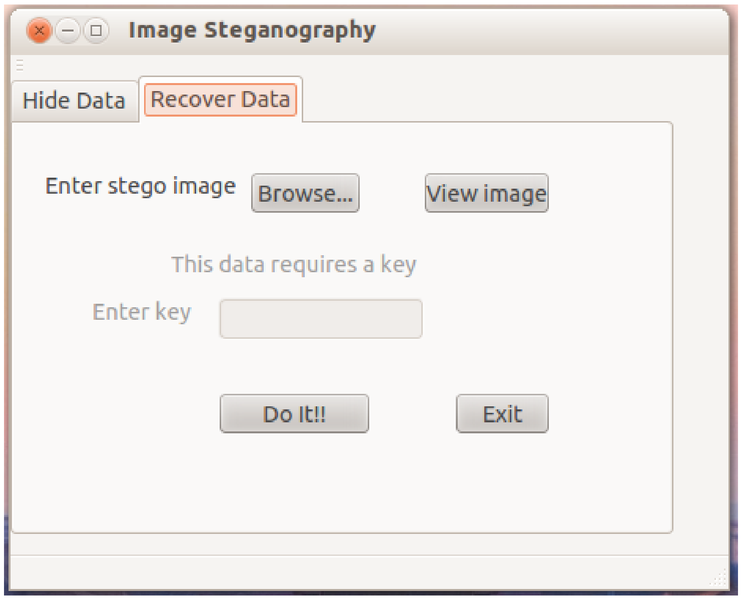
Screen Shots

I Start screen for users



[✓] Encrypt using AES-128

Fig B.1 Start screen 1



This data requires a key

Fig B.2 Start screen 2

II Data embedding

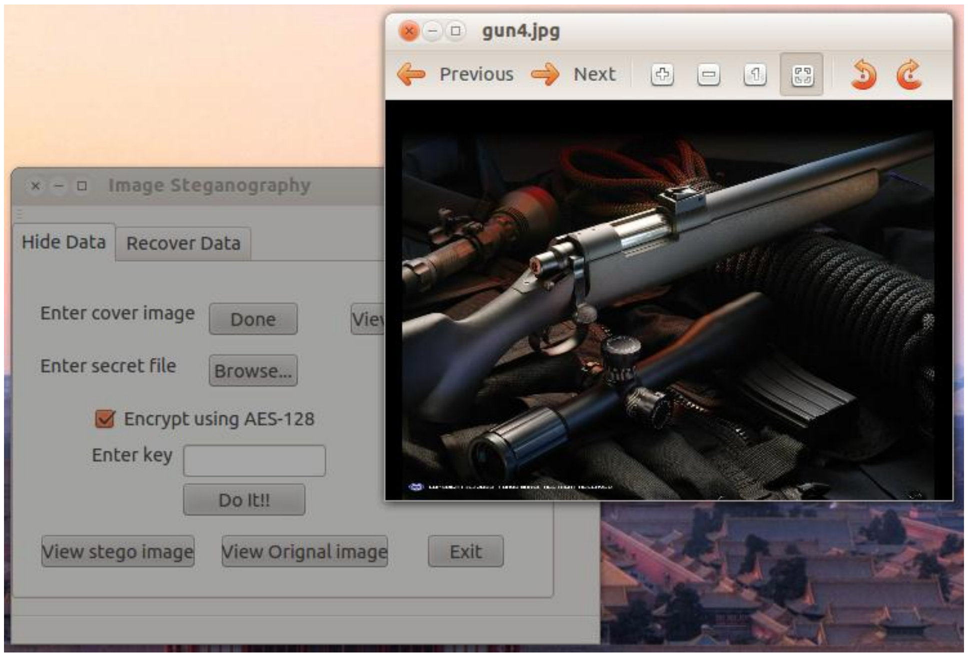


Fig B.3 Data embedding

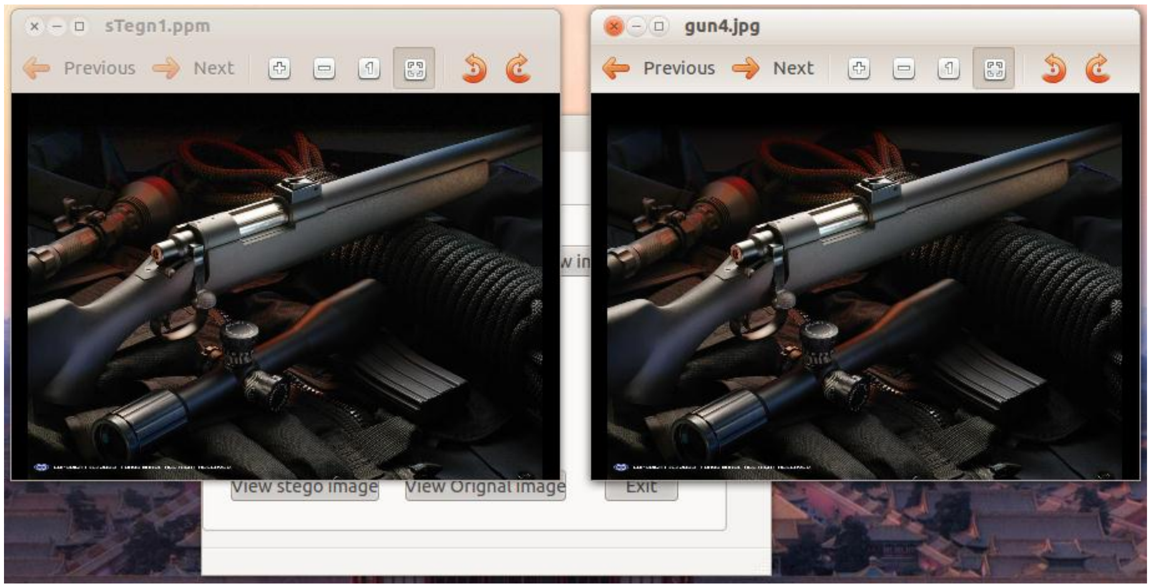


Fig B.4 Image after and before embedding

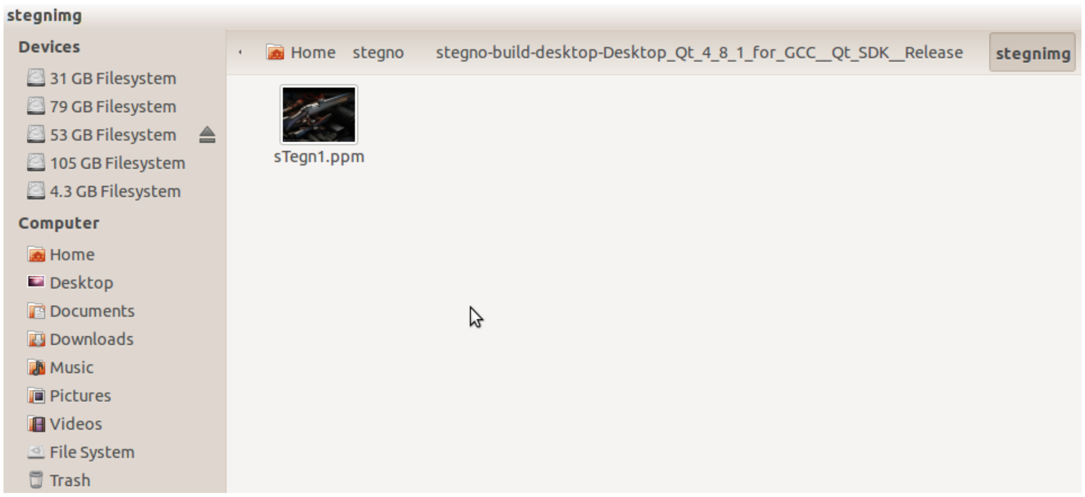


Fig B.5 Embedded image produced at sender side

III Data extraction at receiver side

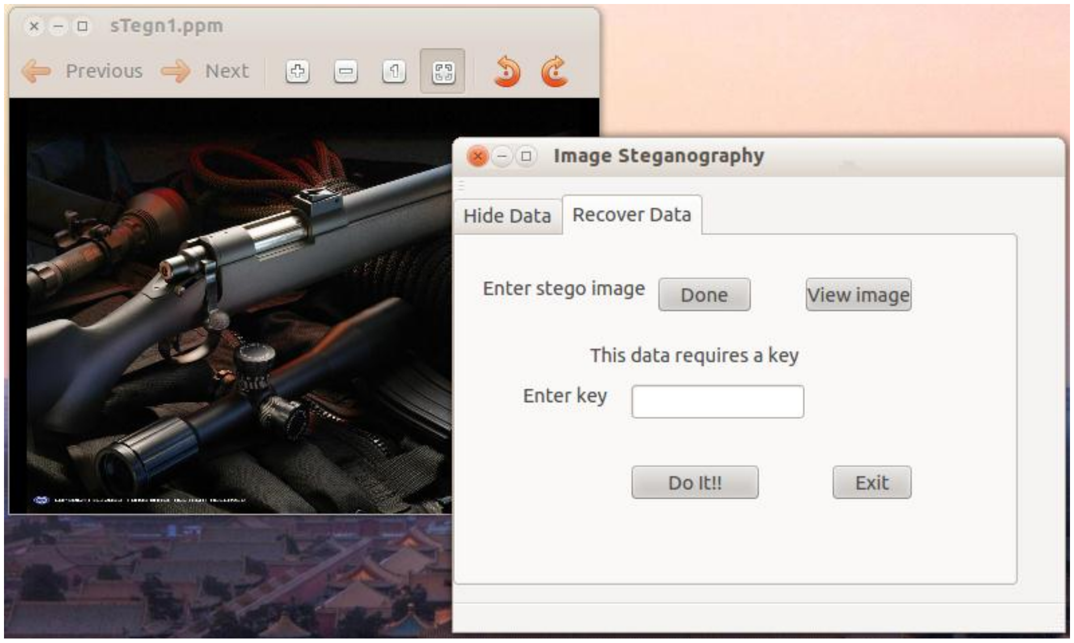


Fig B.6 Data extraction

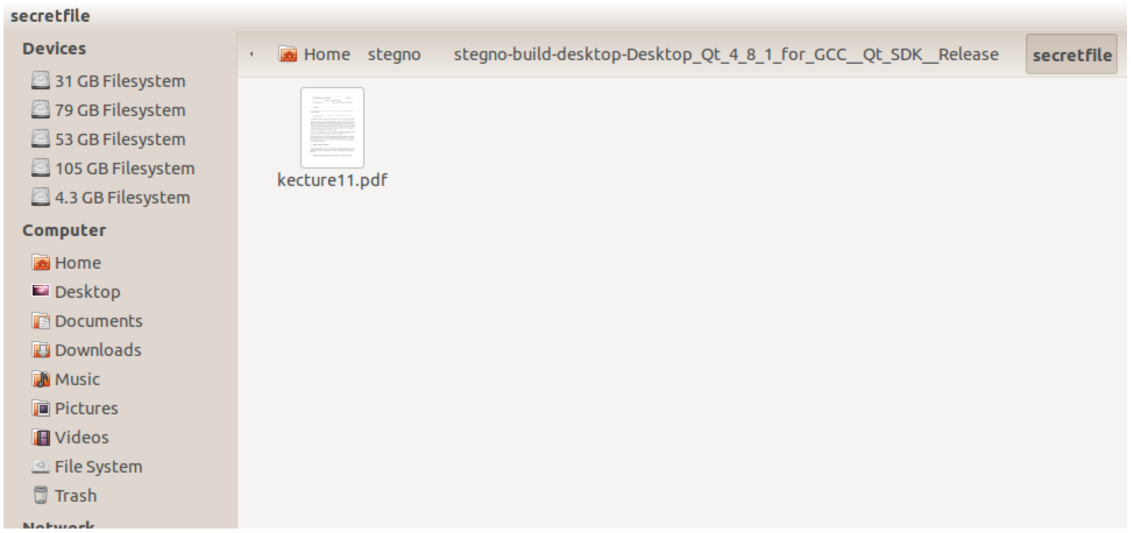


Fig B.7 Recovered file