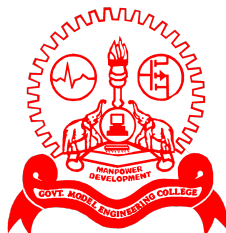


# A Secure method to send Emails using - Compression, Encryption and Pixel Value Differencing

CS492 Project

CSU 152 24 MDL15CS047 Joby Mathew  
CSU 152 34 MDL15CS070 Merin Francis  
CSU 152 39 MDL15CS087 Ria Rajan Alappat  
CSU 152 56 MDL15CS113 Tony Josi

B. Tech Computer Science & Engineering



Department of Computer Engineering  
Model Engineering College  
Thrikkakara, Kochi 682021  
Phone: +91.484.2575370  
<http://www.mec.ac.in>  
[hodcs@mec.ac.in](mailto:hodcs@mec.ac.in)

May 2019

Model Engineering College, Thrikkakara  
Department of Computer Engineering



C E R T I F I C A T E

This is to certify that, this report titled *A Secure method to send Emails using - Compression, Encryption and Pixel Value Differencing* is a bonafide record of the work done by

CSU 152 24 MDL15CS047 Joby Mathew  
CSU 152 34 MDL15CS070 Merin Francis  
CSU 152 39 MDL15CS087 Ria Rajan Alappat  
CSU 152 56 MDL15CS113 Tony Josi

Eighth Semester B. Tech. Computer Science & Engineering

students, for the course work in **CS492 Project**, which is the second part of the two semester project work, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, B. Tech. Computer Science & Engineering of **APJ Abdul Kalam Technological University**.

Guide

Raheena Salihin  
Assistant Professor  
Computer Engineering

Coordinator

Manilal D L  
Associate Professor  
Computer Engineering

Head of the Department

Manilal D L  
Associate Professor  
Computer Engineering

June 7, 2019

## **Acknowledgements**

This project would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to all of them.

First of all, We would like to thank our esteemed Principal, Dr. Vinu Thomas, for his guidance and support in maintaining a calm and refreshing environment to work in and also for providing the facilities that this work demanded.

We are highly indebted to our Project Coordinator and Head of the Department, Dr. Manilal D L, Associate Professor for his guidance, support and constant supervision throughout the duration of the work as well as for providing all the necessary information and facilities that this work demanded.

We would like to thank our Project Guide, Mrs. Raheena Salihin for her support and valuable insights and also for helping us out in correcting any mistakes that were made during the course of the work.

We offer our sincere gratitude to all our friends and peers for their support and encouragement that helped us get through the tough phases during the course of this work.

Last but not the least, we thank the Almighty God for guiding us through and enabling us to complete the work within the specified time.

Joby Mathew

Merin Francis

Ria Rajan Alappat

Tony Josi

### **Abstract**

The project aims at secure transmission of confidential data via emails using a new methodology that combines Compression, Encryption and Pixel Value Differencing. The process of securing the data involves: Arithmetic coding was applied on secret message for lossless compression, which provided 22% higher embedding capacity. The compressed secret message is subjected to AES encryption; this provides higher security in the cases of steganalysis attacks. After compression and encryption, LSB substitution and PVD are applied.

# Contents

<b>List of Figures</b>	<b>iv</b>
<b>List of Tables</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Proposed Project . . . . .	2
1.1.1 Problem Statement . . . . .	2
1.1.2 Proposed Solution . . . . .	2
<b>2 System Study Report</b>	<b>3</b>
2.1 Literature Survey . . . . .	3
2.2 Proposed System . . . . .	6
<b>3 Software Requirement Specification</b>	<b>7</b>
3.1 Introduction . . . . .	7
3.1.1 Purpose . . . . .	7
3.1.2 Intended Audience and Reading Suggestions . . . . .	7
3.1.3 Project Scope . . . . .	7
3.1.4 Overview of Developers Responsibilities . . . . .	8
3.2 Overall Description . . . . .	9
3.2.1 Product Perspective . . . . .	9
3.2.2 Product Functions . . . . .	9
3.2.3 User Classes and Characteristics . . . . .	9
3.2.4 Operating Environment . . . . .	9
3.2.5 Design and Implementation Constraints . . . . .	9
3.2.6 User Documentation . . . . .	9
3.2.7 Assumptions and Dependencies . . . . .	10
3.3 External Interface Requirements . . . . .	11
3.3.1 User Interfaces . . . . .	11
3.3.2 Hardware Interfaces . . . . .	11
3.3.3 Software Interfaces . . . . .	11
3.3.4 Communications Interfaces . . . . .	11
3.4 Hardware and Software Requirements . . . . .	12
3.4.1 Hardware Requirements . . . . .	12
3.4.2 Software Requirements . . . . .	12
3.5 Functional Requirements . . . . .	13

3.5.1	Mailing System Requirements . . . . .	13
3.5.2	Input Requirements . . . . .	13
3.5.3	Compression Stage Requirements . . . . .	13
3.5.4	Encryption Requirements . . . . .	13
3.5.5	Steganography . . . . .	13
3.6	Non-functional Requirements . . . . .	14
3.6.1	Performance Requirements . . . . .	14
3.6.2	Safety Requirements . . . . .	14
3.6.3	Security Requirements . . . . .	14
<b>4</b>	<b>System Design</b>	<b>15</b>
4.1	System Architecture . . . . .	15
4.2	Input phase from user . . . . .	16
4.3	Compression of secret message . . . . .	16
4.4	Encryption of compressed data . . . . .	16
4.5	Steganography using PVD . . . . .	17
4.6	Sending the message across the network . . . . .	17
4.7	Retrieval of Cover Image by the receiver . . . . .	17
4.8	Processing to obtain the secret message . . . . .	17
4.9	Input Design . . . . .	18
4.9.1	Use case diagram . . . . .	19
4.9.2	Class Diagram . . . . .	20
4.9.3	Activity diagram . . . . .	21
4.10	Libraries and Packages Used . . . . .	22
4.11	Module Description . . . . .	25
4.11.1	Compression Module . . . . .	25
4.11.2	Encryption Module . . . . .	25
4.11.3	Embedding Module . . . . .	26
<b>5</b>	<b>Data Flow Diagram</b>	<b>28</b>
5.1	Level 0 DFD . . . . .	28
5.2	Level 1 DFD . . . . .	29
5.3	Level 2 DFD . . . . .	30
<b>6</b>	<b>Implementation</b>	<b>31</b>
6.1	Algorithms . . . . .	31
6.1.1	Data Compression . . . . .	31
6.1.2	Encryption . . . . .	32
6.1.3	Steganography . . . . .	33
6.2	Development Tools . . . . .	35
6.2.1	Git . . . . .	35
6.2.2	Visual Studio Code . . . . .	35
<b>7</b>	<b>Testing</b>	<b>36</b>
7.1	Testing Methodologies . . . . .	36
7.2	Unit Testing . . . . .	36

A Secure method to send Emails using - Comp., Enc. and PVD	Contents
7.3 Integration Testing . . . . .	37
7.4 System Testing . . . . .	37
<b>8 User Interface</b>	<b>38</b>
8.1 GUI Overview . . . . .	38
8.2 Main GUI Components . . . . .	40
8.2.1 Compose Mail . . . . .	40
8.2.2 Alert Boxes for Compose Mail Window . . . . .	40
8.2.3 Extracting Encrypted data from Cover Image . . . . .	41
8.2.4 Alert Boxes for Extract Data Window . . . . .	43
8.2.5 Alert Boxes for Successfully Sending of Mail . . . . .	43
<b>9 Results</b>	<b>45</b>
9.0.1 Image Quality Comparison . . . . .	45
9.0.2 Image Quality Comparison with Varying Input File Size done on Same Cover Image . . . . .	50
9.0.3 Plots for Image Quality Comparison . . . . .	52
<b>10 Conclusion</b>	<b>55</b>
<b>11 Future Scope</b>	<b>56</b>
<b>References</b>	<b>57</b>

# List of Figures

Figure 4.1:	Architecture Diagram . . . . .	16
Figure 4.2:	Use case diagram . . . . .	19
Figure 4.3:	Class diagram . . . . .	20
Figure 4.4:	Activity diagram . . . . .	21
Figure 5.1:	DFD Level 0 . . . . .	28
Figure 5.2:	DFD Level 1 . . . . .	29
Figure 5.3:	DFD Level 2 . . . . .	30
Figure 8.1:	Basic UI . . . . .	39
Figure 8.2:	Compose Mail . . . . .	40
Figure 8.3:	Alert Boxes for Compose Mail Window . . . . .	41
Figure 8.4:	Extracting Encrypted data from Cover Image . . . . .	42
Figure 8.5:	Alert Boxes for Extract Data Window . . . . .	43
Figure 8.6:	Alert Boxes for Successfully Sending of Mail . . . . .	44
Figure 9.1:	Test Image 1 – Orginal . . . . .	45
Figure 9.2:	Test Image 1 – Embedded . . . . .	46
Figure 9.3:	Test Image 2 – Orginal . . . . .	46
Figure 9.4:	Test Image 2 – Embedded . . . . .	47
Figure 9.5:	Test Image 3 – Orginal . . . . .	47
Figure 9.6:	Test Image 3 – Embedded . . . . .	48
Figure 9.7:	Test Image 4 – Orginal . . . . .	48
Figure 9.8:	Test Image 4 – Embedded . . . . .	49
Figure 9.9:	Test 1 - Input Size: 644 Bytes . . . . .	50
Figure 9.10:	Test 1 - Input Size: 1289 Bytes . . . . .	50
Figure 9.11:	Test 1 - Input Size: 1934 Bytes . . . . .	50
Figure 9.12:	Test 1 - Input Size: 2579 Bytes . . . . .	51
Figure 9.13:	Test 1 - Input Size: 3224 Bytes . . . . .	51
Figure 9.14:	Mean Square Error (MSE) . . . . .	52
Figure 9.15:	Peak Signal to Noise Ratio (PSNR) . . . . .	53
Figure 9.16:	Structural Similarity . . . . .	54



# List of Tables

# Chapter 1

## Introduction

Cryptography and information hiding are the two branches of information security system. Cryptography is used for encrypting and decrypting the data into ciphertext, which is meaningless and hard to understand. In spite of very high levels of security provided by various advanced cryptographic techniques, illegible nature of ciphertext easily draws attention of adversaries and, thus, may result in failure of communication.

Information/data hiding is a mechanism which ensures that the presence of the secret data remains undetected. Data hiding can further be sub-divided into digital watermarking and steganography. In digital watermarking, noise tolerant signals such as an audio, image or video etc. are used to covertly hide a kind of signal (watermark) which is used to establish the ownership of such signal. Steganography is used for embedding high amount of data in cover files (image, video, audio etc.). Image based covers are the most frequently used covers. Steganography is divided into spatial and frequency domains. Spatial domain involves embedding of secret message by direct modification of intensity of cover image pixels. The transformed domain coefficients are altered to embed secret data in frequency domain methods. Spatial domain methods require less computational complexity and provide higher embedding capacity than frequency domain methods.

Steganographic methods, devised to fulfill the requirements of recovery of original cover image are termed as reversible embedding. Reversible embedding is required for applications like military, medical and legal etc. Generally, during embedding, the LSBs of cover image are overwritten by secret message bits, the original bits are lost and cannot be recovered, thus named as irreversible data embedding. Both spatial and frequency domains have been used for reversible and irreversible embedding methods. Irreversible embedding provides higher embedding capacity than reversible embedding.

The performance evaluation of image steganography methods is based on the parameters like: hiding capacity, visual quality/imperceptibility and security/un-detectability. However, these evaluating parameters produce opposite effects with each other e.g. the steganographic methods designed to achieve higher hiding capacity result in visual distortions to the steganographed images and reduced security. Thus, proper corrective/balancing measures are required to make balance between these parameters. Qualities like high embedding capacity, un-detectability and satisfactory visual quality are required for real applications.

## 1.1 Proposed Project

The proposed project is python software package that aims at a new method for sending highly confidential information using cryptographic and stenographic methods. The textual data to be sent is embedded in a cover image and sent across the network.

The main steps involved in this method include compression, encryption and steganography. The data is first compressed to improve efficiency by decreasing size. Then the data is undergone encryption using AES. The encrypted data is then embedded to a cover image using a new steganographic method - PVD (Pixel Value Differencing).

The embedded cover image is sent to the receiver using simple mail transfer protocol library (smtplib), email (pyhton library) and MIME protocols.

### 1.1.1 Problem Statement

In order to protect and secure the transmission of data over an open channel e.g. internet, an information security system must be in place.

The main approaches in data security are encryption and data hiding. Data hiding is mostly achieved via steganography. The simplest and most popular image steganographic technique is the LSB substitution. It involves the embedding of messages into cover image by directly replacing the LSBs. The hiding capacity can be as high as 4 LSBs per pixel. A common weakness of LSB embedding is that sample value changes asymmetrically.

Through LSB embedding, visual quality may decrease and become sensitive to steganalysis attacks.

### 1.1.2 Proposed Solution

The pixel value differencing (PVD) for improving imperceptibility in stego images. In PVD method, data embedding is done by readjusting the difference between two pixels. The PVD based methods are vulnerable to histogram analysis and provide low embedding rates.

Enhanced hiding capacity over PVD based methods was proposed by combining LSB and PVD. The PVD and LSB embedding methods are applied on smooth and edge areas of the cover image respectively, after partitioning the cover image into smooth and edge areas. The hiding capacity is reported to be improved. This approach is susceptible to RS steganalysis detection attacks. This resulted in enhanced visual quality and remains secure against RS steganalysis. But this method does not enhance the hiding capacity.

Adaptive steganographic method was proposed for achieving high embedding capacity by integrating LSB and PVD. The cover image is partitioned into 1x3 non-overlapping pixel blocks and second pixel of each block is selected as base pixel. The base pixel is subjected to k-bit LSB and first and third pixels of the block are subjected to PVD.

## Chapter 2

# System Study Report

### 2.1 Literature Survey

Sl.No	Title	Year	Author	Advantages	Disadvantages
1	A secure and High-capacity Data hiding using Compression, Encryption and Optimized Pixel Value Differencing	2017	Awdhesh K. Shukla Akanksha Singh Balvinder Singh Amod Kumar	22% higher embedding capacity. Security against regular/singular steganalysis attacks. Less chances of failures in communication. High levels of visual quality	High computation time
2	An Approach of Cryptography and Steganography using Rotor cipher for secure Transmission	2015	Sriram S Karthikeyan B Vaithiyanathan V Anishin Raj M. M	Robust and resilient The efficiency of this technique is $n(26)$ . Increased complexity decreased the decryption possibility of the message.	Method is relatively complex. Difficulty in finding the correct pixel positions in order to find the data bits.

Sl.No	Title	Year	Author	Advantages	Disadvantages
3	An Edge Based Image Steganography with Compression and Encryption	2015	Rina Mishra Atish Mishra Praveen Bhanodiya	Huge data hiding capacity Less distortion of stegno image Technique is robust and secure.	Less Embedding Capacity
4	A Clustering Based Steganographic Approach for Secure Data Communication	2015	G.Manikandan Sairam.N A.SaiKrishna	Embedding data capacity and image quality. Enhanced security and decreased distortions.	Cannot be used with colour images No cryptography
5	Secure Transmission of Data by Splitting Image	2015	Jitha Raj.T E.T Sivadasan	More safer Increased imperceptibility	Complex High chances of image distortion
6	Biometric Logical Access Control Enhanced by Steganography Over Secured Transmission Channel	2011	Adrian Kapczycki Arkadiusz Banasik	Image distortion is comparatively very less	The successful man-in-the-middle attack can result in obtaining the transmitted data
7	Securing data using Elliptic Curve Cryptography and Least Significant Bit Steganography	2017	Jayati Bhadra M.K Banga M Vinayaka Murthy	Time complexity is reduced Noise in stego object is minimised Inserting capability is increased Robustness is increased	Greater processing time  Increased cost of production.

Sl.No	Title	Year	Author	Advantages	Disadvantages
8	LSB Rotation and Inversion Scoring Approach to Image Steganography	2018	Ryan A. Subong Arnel C. Fajardo Yoon Joong Kim	Higher embedding capacity Stego images with superior visual quality. Less complex algorithm	Easier to detect the presence of hidden data. Less data security
9	An Improved LSB based Image Steganography Technique for RGB Images	2015	Amritpal Singh Harpal Singh	Images have higher PSNR value Better quality of image. Reduced the visible distortions in the embedded image.	Even though the proposed method of Least Significant bit (LSB) for secret message insertion is made on the basis of sensitivity of human eyes, the method uses constant number of LSB replacement for every pixel which in turn can lead to distortions for specific images. Lesser security for hidden data as encryption methods are not used. Poor embedding capacity
10	High payload image steganography with reduced distortion using octonary pixel pairing scheme	2013	C. Balasubramanian S. Selvakumar S. Geetha	Sustains any modified pixel in the same level in the stego-image also, where the difference between a pixel and its neighbor in the cover image belongs to, for imparting the statistical undetectability factor.	Process of extraction of data from cover image is more complex compared to the usual method Lesser embedding capacity Result of RS Steganalysis is poor compared to the proposed method

## 2.2 Proposed System

The main objective of this project is to present a high capacity data hiding method using lossless compression, Advanced Encryption Standard (AES), pixel value differencing (MPVD) and least significant bit (LSB) substitution. We propose a methodology to send textual mails using this method to enable more security. The content of the mail is encrypted and then embedded into a cover image which is in turn forwarded to the intended receiver. The mails are sent using the smtplib Library available in Python.

1. Compression using ARITHMETIC CODING
2. Encryption and decryption using AES
3. Embedding procedure using LSB substitution
4. Sending cover image and Extraction procedure

## Chapter 3

# Software Requirement Specification

### 3.1 Introduction

#### 3.1.1 Purpose

Electronic mail (email or e-mail) is a method of exchanging messages ("mail") between people using electronic devices. Email security refers to the collective measures used to secure the access and content of an email account or service. It allows an individual or organization to protect the overall access to one or more email addresses/accounts. An email service provider implements email security to secure subscriber email accounts and data from hackers - at rest and in transit.

This project aims at creating a platform for securing the sending of mails using Compression, Encryption and Pixel Value Differencing

#### 3.1.2 Intended Audience and Reading Suggestions

The document is intended for developers, project managers, testers and documentation writers. The rest of this SRS contains further details of the project, which includes scope of the project, as well as software and hardware requirements.

#### 3.1.3 Project Scope

The product is useful for securing the existing mailing system by using most efficient data security methods like Compression(by arithmetic coding), Encryption and Pixel Value Differencing. The arithmetic coding helps to compress the data size as well thus reducing the bandwidth and embedding requirements.



### 3.1.4 Overview of Developers Responsibilities

The responsibility of the developer is to implement the further features :

- Implement arithmetic coding to compress the given mail content
- Develop 128 bit AES to encrypt the encoded data after compression
- Implement a stenographic algorithm to embed encrypted data using LSB substitution and Pixel Value Differencing
- To implement a mailing system to send mails using the proposed security method
- Develop a system to decode the cipher text from image
- Implement inverse AES to get encoded textual data
- Reverse compression to obtain the original email text

## **3.2 Overall Description**

### **3.2.1 Product Perspective**

The proposed project is a modification of the original mailing platform used for sending emails by improving the security and efficiency. Data compression involved in this project compresses the total data that needs to be send across and thus increases the efficiency of further processes. AES and Steganography further adds the security of the current system.

### **3.2.2 Product Functions**

- Input the details of the mail including content and sender
- Apply the proposed methods to make the data secure
- Send mails with the processed data to the given sender address

### **3.2.3 User Classes and Characteristics**

The application is intended to be used by regular users, Who may not possess any technical expertise and it aims at improving Quality Of Experience of users who are in need of sending emails more securely and efficiently. The user simply need to have the basic knowledge to give information to the respective field.

### **3.2.4 Operating Environment**

The product can be accessed by terminal commands on either Mac, Linux or Windows

- On any Operating system containing the required libraries there is a requirement of functioning internet connection.

### **3.2.5 Design and Implementation Constraints**

The Design and Implementation Constraints include:

- There will be a limit on the size of the data which can be embed to the cover image depending upon the cover image.
- Constraint on the resource utilization while using AES algorithm
- Only textual data can be given as input to the system

### **3.2.6 User Documentation**

This system doesnt require any user documentation since the user can simply type in the command and input data accordingly as prompted

### 3.2.7 Assumptions and Dependencies

- The designed algorithm is for English language and the text the application parses must be written in this language.
- The interface of resulting system will be easy to use and accessible without a time or location constraint.
- The application requires a stable network connection.
- The platform should be containing the updated python versions of cv2, docopt and numpy.

### **3.3 External Interface Requirements**

#### **3.3.1 User Interfaces**

The user interface is a simple command that further prompts user about the further data that is to be taken as input. This input data will be processed and then send to the mail of the given sender. The mail retrieval also takes place in the same way by using same command.

#### **3.3.2 Hardware Interfaces**

No particular hardware interface.

#### **3.3.3 Software Interfaces**

This application takes input from the user as textual data of sender address (mail) and content of the mail (text). The input is then sent for further processing of text compression, encryption and stenography with a given cover image. The processed data is then sent across the network to the given address. The application also makes use of APIs like cv2, docopt, numpy, etc....

#### **3.3.4 Communications Interfaces**

The only communication over the network is for sending the mail with the processed data. The mailing system uses Simple Mail Transfer Protocol (SMTP) for sending mails.

## 3.4 Hardware and Software Requirements

### 3.4.1 Hardware Requirements

- System: Any Quad core system clocked at 2.4GHz (Minimum)

AES and Stenography process consumes a lot of system resources, Therefore a system with Clock speed at least 2.4GHz is required.

- RAM : 4GB (Recommended)

For smooth and faster functioning of AES.

### 3.4.2 Software Requirements

- Operating system : Linux (64-bit)

Developer friendly, Powerful shell, Support, Flexibility.

- Language : Python 3.0

A large number of APIs for various purposes is available in python.

- Libraries : PIL, smtplib, MIME, email

PIL - Python library for image processing.

smtplib - Python library for sending emails

MIME - Library for creating MIME object for mail

email - Standard library for email in Python

## **3.5 Functional Requirements**

### **3.5.1 Mailing System Requirements**

- The system should be able to send textual mails to the given addresses after performing the required functions
- The input should only accept textual data
- The system should not preserve the text formatting

### **3.5.2 Input Requirements**

- The system should be able to define a maximum limit for the size of input data
- The input should only accept textual data
- The system should not preserve the text formatting

### **3.5.3 Compression Stage Requirements**

- The system should be able to compress the given text file using arithmetic coding
- The compression algorithm must accept the acceptable input size prescribed for the input
- The output file should also be a text file

### **3.5.4 Encryption Requirements**

- The algorithm used should be AES
- The cipher key should be 128 bits with block size of 128 bits

### **3.5.5 Steganography**

- The system should be using Pixel Value Differencing algorithm for embedding encrypted data to the cover image
- The cover image should be colour image
- The output should be also an embedded colour image

## **3.6 Non-functional Requirements**

### **3.6.1 Performance Requirements**

Calculation time and response time should be as little as possible, because one of the softwares features is timesaving. The capacity of servers should be as high as possible.

### **3.6.2 Safety Requirements**

The possible harm that can occur during the process is the chance of detection of data from cover image using steganalysis, but the chances are minimal due to the use of Pixel Value Differencing algorithm.

### **3.6.3 Security Requirements**

The data given by the user is undergone several steps for improving security before transmitting across the network.

## Chapter 4

# System Design

### 4.1 System Architecture

The whole system is divided into four major phases:

- Input phase from user
- Compression of secret message
- Encryption of compressed data
- Steganography using PVD
- Sending the message across the network
- Retrieval of Cover Image by the receiver
- Processing to obtain the secret message



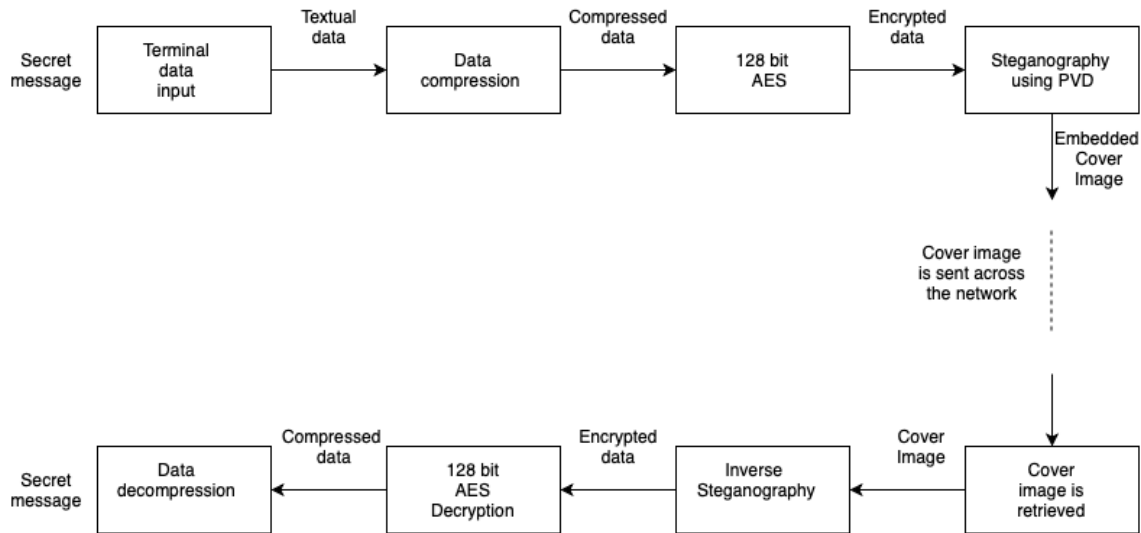


Figure 4.1: Architecture Diagram

## 4.2 Input phase from user

In this phase data is taken from the user along with the user log-in details and secret message that needs to be sent across using the system. The input is taken from the user via terminal or command line prompts.

## 4.3 Compression of secret message

In this phase the secret message from the user input is compressed using arithmetic coding. The capacity of system can be improved by using this phase. Arithmetic coding is a form of entropy encoding used in lossless data compression. Normally, a string of characters such as the words "hello there" is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total.

On an average, about 22% higher embedding capacity was achieved through Arithmetic Coding.

## 4.4 Encryption of compressed data

During this phase the encoded data from the previous phase is encrypted using 128-bit AES Symmetric Encryption algorithm. The fundamental function of this phase is to provide additional security for the secret message prior to hiding it in cover image using steganography.

AES is based on a design principle known as a substitutionpermutation network, and is efficient in both software and hardware. AES operates on a 4 4 column-major order array of bytes, termed the state. Most AES calculations are done in a particular finite field.

## 4.5 Steganography using PVD

The encrypted data is then embedded in the cover image during this phase. The cover image should be a color image.

After compression and encryption of secret message, modified approach of Khodaei and Faezs LSB+PVD method is applied to embed the message in cover image. In the proposed embedding method, the cover image is divided into non-overlapping pixel blocks of 3x3 or 3x3 + 2x2 pixel blocks as per the cardinality of the cover image. That is, if the dimensions of the cover image are not divisible by 3x3 pixel blocks, the remaining pixels are taken 2x2 pixel blocks.

## 4.6 Sending the message across the network

The cover image is send to the receiver using the existing mailing technology. The mailing system uses Simple Mail Transfer Protocol (SMTP) for sending mails. The mail is sent using SMTPLIB - library available in Python.

The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

## 4.7 Retrieval of Cover Image by the receiver

The cover image along with the embedded message is retrieved by the receiver by using the same SMTPLIB - library the cover image is downloaded to the receiver machine.

## 4.8 Processing to obtain the secret message

The cover image is then undergone inverse steganography, AES decryption and Decompression to obtain the orginal secret message sent by the sender. The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order Add round key, Mix columns, Shift rows and Byte substitution

Also inverse arithmetic coding is applied to decompress the compressed message after the decryption phase of AES.

## 4.9 Input Design

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the various input devices being used in the system. Therefore, the quality of system input determines the quality of system output. Well designed input forms and screens have following properties

- It should serve specific purposes effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on users attention, consistency, and simplicity.

### Objectives for Input Design

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

#### 4.9.1 Use case diagram

Use case diagrams are considered to be used for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. The use cases are nothing but the system functionalities written in an organised manner. Now the second class of things that is relevant to the use cases are the actors. Actors can be defined as something that interacts with the system. Actors can be the human user, some internal application or some external application. To draw a use case diagram we should have the following items identified:

- Functionalities to be represented as an use case.
- Actors.
- Relationships among the use cases and actors.

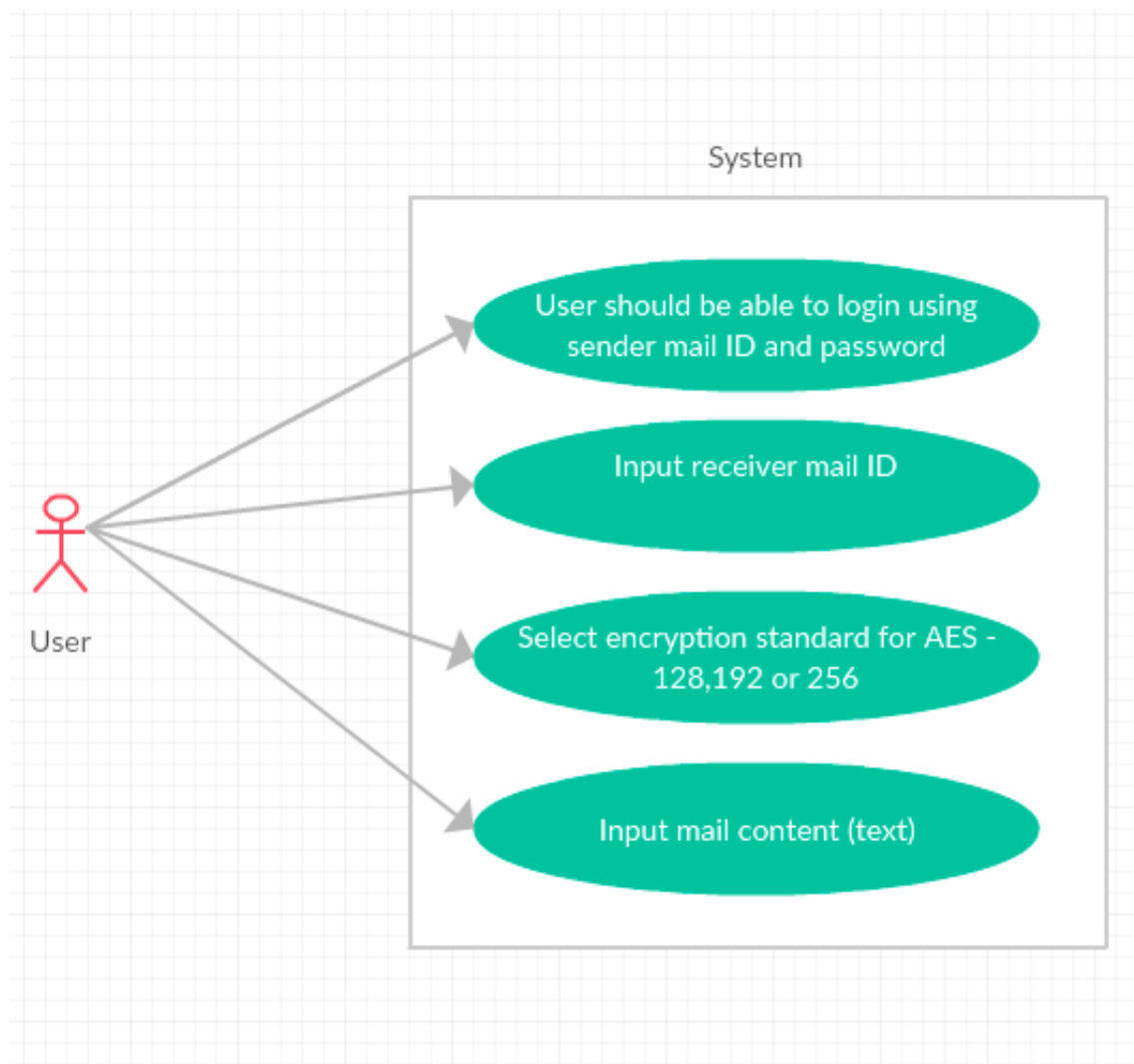


Figure 4.2: Use case diagram

### 4.9.2 Class Diagram

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP). The concept is several years old but has been refined as OOP modeling paradigms have evolved.

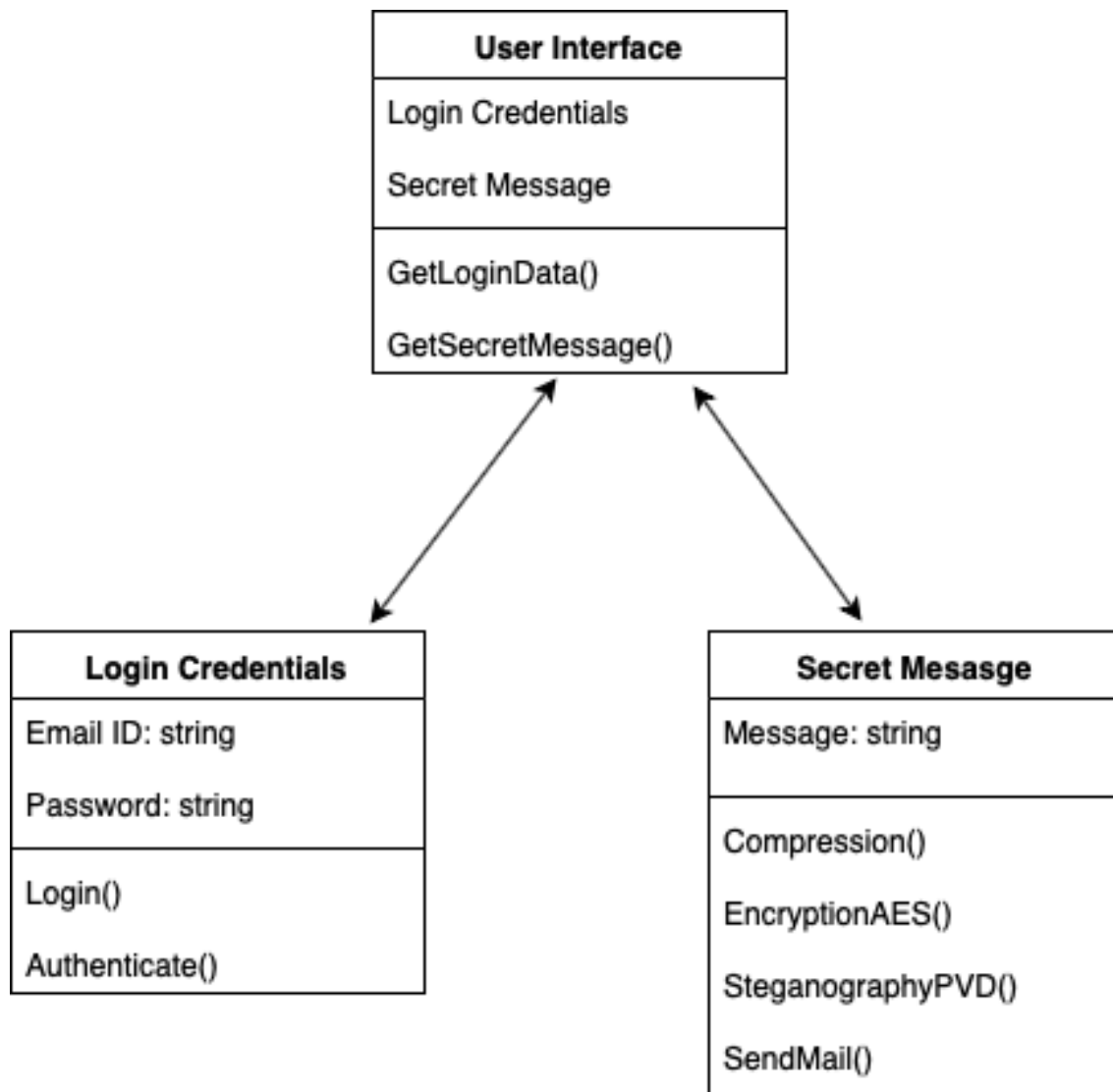


Figure 4.3: Class diagram

### 4.9.3 Activity diagram

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. 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.

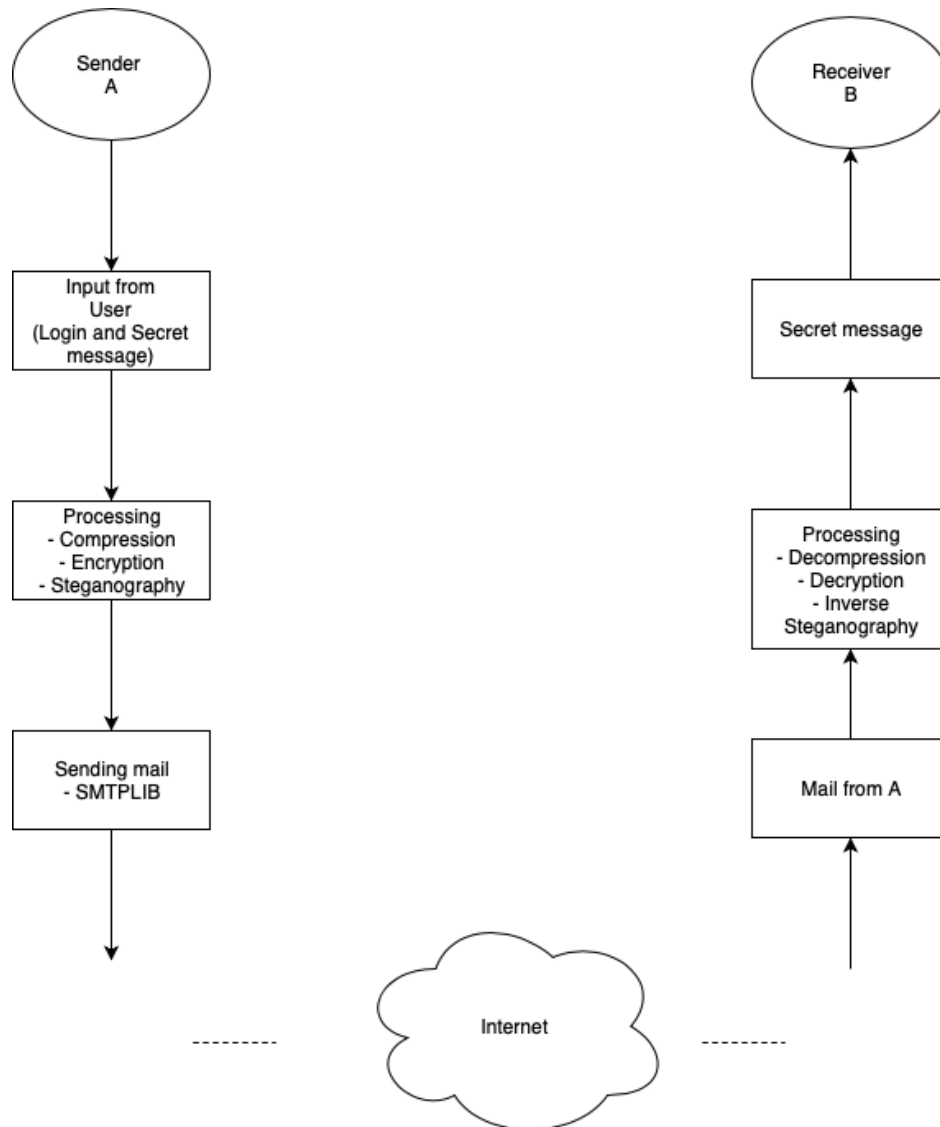


Figure 4.4: Activity diagram

## 4.10 Libraries and Packages Used

- **Python Imaging Library** - Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It is available for Windows, Mac OS X and Linux. The latest version of PIL is 1.1.7, was released in September 2009 and supports Python 1.5.22.7, with Python 3 support to be released "later". Development appears to be discontinued with the last commit to the PIL repository coming in 2011.[2] Consequently, a successor project called Pillow has forked the PIL repository and added Python 3.x support.[4] This fork has been adopted as a replacement for the original PIL in Linux distributions including Debian[5] and Ubuntu (since 13.04).

Pillow offers several standard procedures for image manipulation. These include:

- per-pixel manipulations, masking and transparency handling,
  - image filtering, such as blurring, contouring, smoothing, or edge finding,
  - image enhancing, such as sharpening, adjusting brightness, contrast or color,
  - adding text to images and much more.
- **smtplib** - The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. SMTP stands for Simple Mail Transfer Protocol. The smtplib module is useful for communicating with mail servers to send mail. Sending mail is done with Python's smtplib using an SMTP server. Actual usage varies depending on complexity of the email and settings of the email server, the instructions here are based on sending email through Gmail.

The SMTP protocol includes a command to ask a server whether an address is valid. Usually VRFY is disabled to prevent spammers from finding legitimate email addresses, but if it is enabled you can ask the server about an address and receive a status code indicating validity along with the user's full name.

The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult RFC 821 (Simple Mail Transfer Protocol) and RFC 1869 (SMTP Service Extensions).

Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.

Python provides smtplib module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail

```
import smtplib
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Here is the detail of the parameters

- **host** This is the host running your SMTP server. You can specify IP address of the host or a domain name like tutorialspoint.com. This is optional argument.

- `port` If you are providing host argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
- `local_hostname` If your SMTP server is running on your local machine, then you can specify just localhost as of this option.

An SMTP object has an instance method called `sendmail`, which is typically used to do the work of mailing a message. It takes three parameters

- The sender A string with the address of the sender.
- The receivers A list of strings, one for each recipient.
- The message A message as a string formatted as specified in the various RFCs.



- **MIME** - To get a message object structure by passing a file or some text to a parser, which parses the text and returns the root message object. Its also possible to build a complete message structure from scratch, or even individual Message objects by hand. In fact, you can also take an existing structure and add new Message objects, move them around, etc. This makes a very convenient interface for slicing-and-dicing MIME messages.

MIME improves the ease of sending emails by object structure by creating Message instances, adding attachments and all the appropriate headers manually. For MIME messages though, the email package provides some convenient subclasses to make things easier.

- **Python email Library** - The email package is a library used in this project to managing email messages, including MIME and other RFC 2822-based message documents. It also subsumes most of the functionality in several older standard modules including rfc822, mime-tools, multifile, and other non-standard packages such as mimecntl and various other mail sending and supporting tools. It is specifically not designed to do any sending of email messages to SMTP (RFC 2821), NNTP, or other servers; those are functions of modules such as smtpplib and nntplib, which are also used in this module. The email package attempts to be as RFC-compliant as possible, supporting in addition to RFC 2822, such MIME-related RFCs as RFC 2045, RFC 2046, RFC 2047, and RFC 2231.

The primary distinguishing feature of this email package is that it splits the parsing and generating of email messages from the internal object model representation of email regardless of the type of the mail. Applications using the email package deal primarily with objects; you can add sub-objects to messages, remove sub-objects from messages, completely re-arrange the contents, etc. There is a separate parser and a separate generator which handles the transformation from flat text to the object model, and then back to flat text again. There are also handy subclasses for some common MIME object types, and a few miscellaneous utilities that help with such common tasks as extracting and parsing message field values, creating RFC-compliant dates, etc.

It is perfectly feasible to create the object structure out of whole cloth i.e. completely from scratch. From there, a similar progression can be taken as above.

Also included are detailed specifications of all the classes and modules that the email package provides, the exception classes you might encounter while using the email package, some auxiliary utilities, and a few examples. For users of the older mimelib package, or previous versions of the email package, a section on differences and porting is provided.

## 4.11 Module Description

This section focuses on the various modules in the project development phase

### 4.11.1 Compression Module

The compression is achieved through Arithmetic Coding. In arithmetic coding, fewer bits are used for frequently used characters and infrequently used characters are stored with more number of bits, thus, resulting in fewer bits required for total encoding.

A pseudo random number generator (PRNG), deployed in MATLAB was used for generating test messages of various capacities. Arithmetic coding and extraction codes were also deployed in MATLAB. On an average, about 22% higher embedding capacity was achieved through Arithmetic Coding. After compression, output of this step would be a bit stream of compressed secret message. Extra 0 bits are added at the end, if required, to make the sequence divisible by 8. This bit stream is used as input for AES based encryption.

- When a string is converted to arithmetic encoding, Here, fewer bits are used for frequently used characters and infrequently used characters are stored with more number of bits.
- This results in fewer bits required for total encoding.
- After compression, output of this step would be a bit stream of compressed secret message.
- Extra 0 bits are added at the end if required, to make the sequence divisible by 8.
- This bit stream is used as input for AES encryption.

### 4.11.2 Encryption Module

Compressed text file is taken as input for encryption using AES . AES encryption ensures that the message does not get disclosed even if the existence of the message is disclosed, thus, ensuring higher security level.

AES uses 128-bit data and variable length (128/192/256- bit) keys. AES is an iterative cipher, the number of rounds in AES vary with the length of the key. Different 128-bit round keys are used in each of these rounds, which are computed out of original AES key. All the computations in AES are performed on bytes rather than bits. 128 bits of a data block is treated as 16 bytes and is arranged as a 4x4 matrix, referred to as state array in AES.

Input: 128-bit data, 128-bit key Output: Cipher text Steps involved: For each round, repeat the following steps:

- SubBytes()
- ShiftRows()
- MixColumns()
- AddRoundkey()

Step c is not required in the final round.

Step 1: SubBytes (state) All the input bytes of the block are taken as independent entities and are substituted by their substituent from the S\_Box, with single S\_Box used for entire data.

Step 2: ShiftRows (state) Each row of the data matrix is shifted to the left and any elements that drop off are re-inserted from the right side of the row. The number of shifts a row encounters varies for different rows.

Step 3: MixColumns (state) A mathematical function is used to transform each four-byte column. Four bytes of one column are taken as input by this function and four new output bytes are generated and original column bytes are replaced with these new output bytes. This step is not executed in the last round.

Step 4: AddRoundKey (state, Key[i]) XOR is performed on received round key and state.

### DECRYPTION USING AES

For each round, with the state and key as inputs (except for last round), following steps are repeated:

- Inverse\_SubBytes()
- Inverse\_ShiftRows()
- Inverse\_MixColumns()
- Inverse\_AddRoundkey()

Step 3 is not performed for last round. All the encryption steps should be performed in reverse to achieve decryption

#### 4.11.3 Embedding Module

After compression and encryption of secret message, modified approach of Khodaei and Faezs LSB+PVD method is applied to embed the message in cover image. In the proposed embedding method, the cover image is divided into nonoverlapping pixel blocks of 3x3 or 3x3 + 2x2 pixel blocks as per the cardinality of the cover image. That is, if the dimensions of the cover image are not divisible by 3x3 pixel blocks, the remaining pixels are taken 2x2 pixel blocks. The embedding process is presented below:

1. Read the cover image I and divide it into 3x3 non-overlapping blocks. However, if the height and width of the image are not divisible by 3x3 blocks, the image is divided into 3x3 plus 2x2 blocks. For example, a 512x512 pixel image is partitioned into 28900 blocks of 3x3 pixels and 511 blocks of 2x2 pixels.
2. Read block B<sub>i</sub> and name the pixels as a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub>, a<sub>4</sub>, a<sub>5</sub>, a<sub>6</sub>, a<sub>7</sub>, a<sub>8</sub>, a<sub>9</sub>. For 2x2 pixel blocks, the pixels would be named as a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub>, and a<sub>4</sub>
3. Select a<sub>1</sub> as base pixel or reference pixel.
4. For reference pixel, embed 3 bits directly into 3 LSBs of a<sub>1</sub> to get a<sub>1</sub>'
5. Calculate the difference d<sub>i</sub> for all pixel values except for a<sub>1</sub>. ie,  $d_i = \text{mod}(a_1 - a_i)$

6. Compute the ranges  $R_i$  to which  $d_i$  belongs. As stated in the range table, compute  $C_i$  i.e. number of bits to be hidden.
7. Read the  $C_i$  bits continuously from  $S$  secret message as  $S_i$ .
8. Replace  $C_i$  bits of  $a_i$  with  $S_i$  to obtain  $a_i'$ .

## Chapter 5

# Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the flow of data through an information system, modelling its process aspects. Often they are viewed as a preliminary step used to create an overview of the system which can be later elaborated. DFDs can also be used for the visualization of data processing.

### 5.1 Level 0 DFD

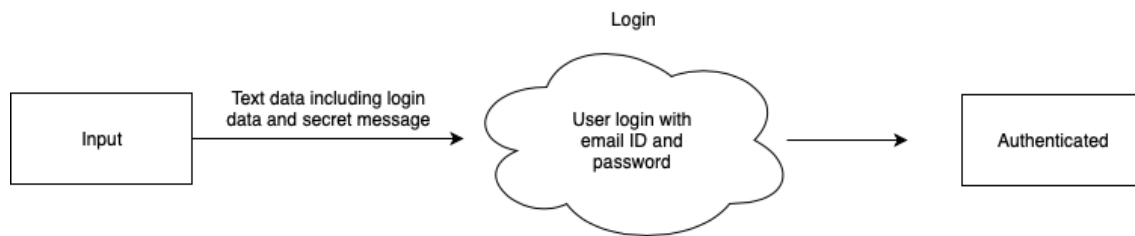


Figure 5.1: DFD Level 0

## 5.2 Level 1 DFD

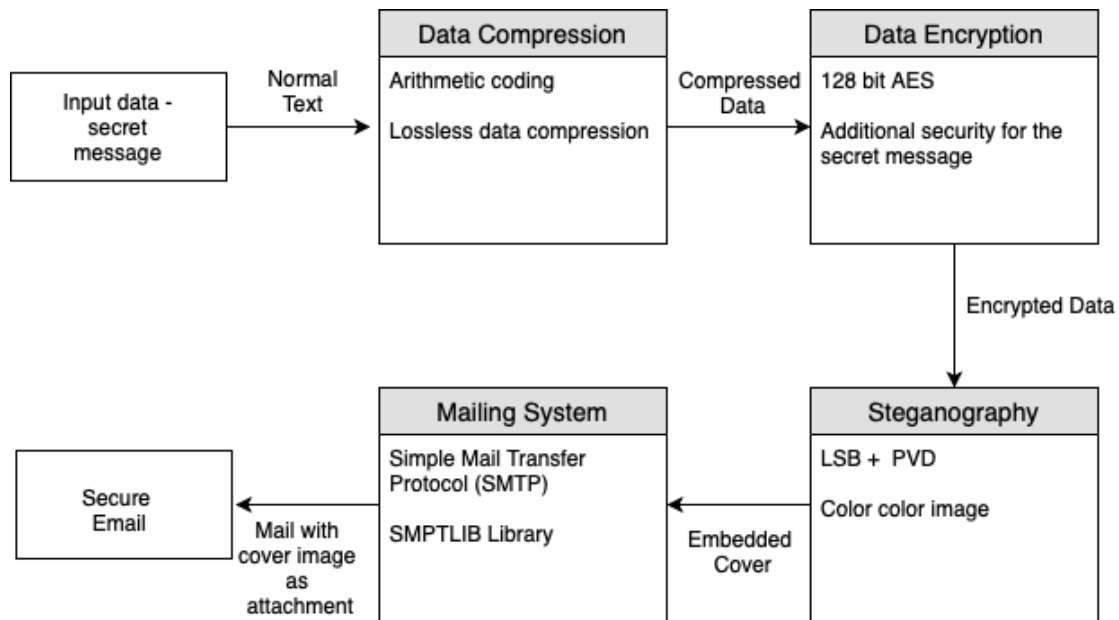


Figure 5.2: DFD Level 1

### 5.3 Level 2 DFD

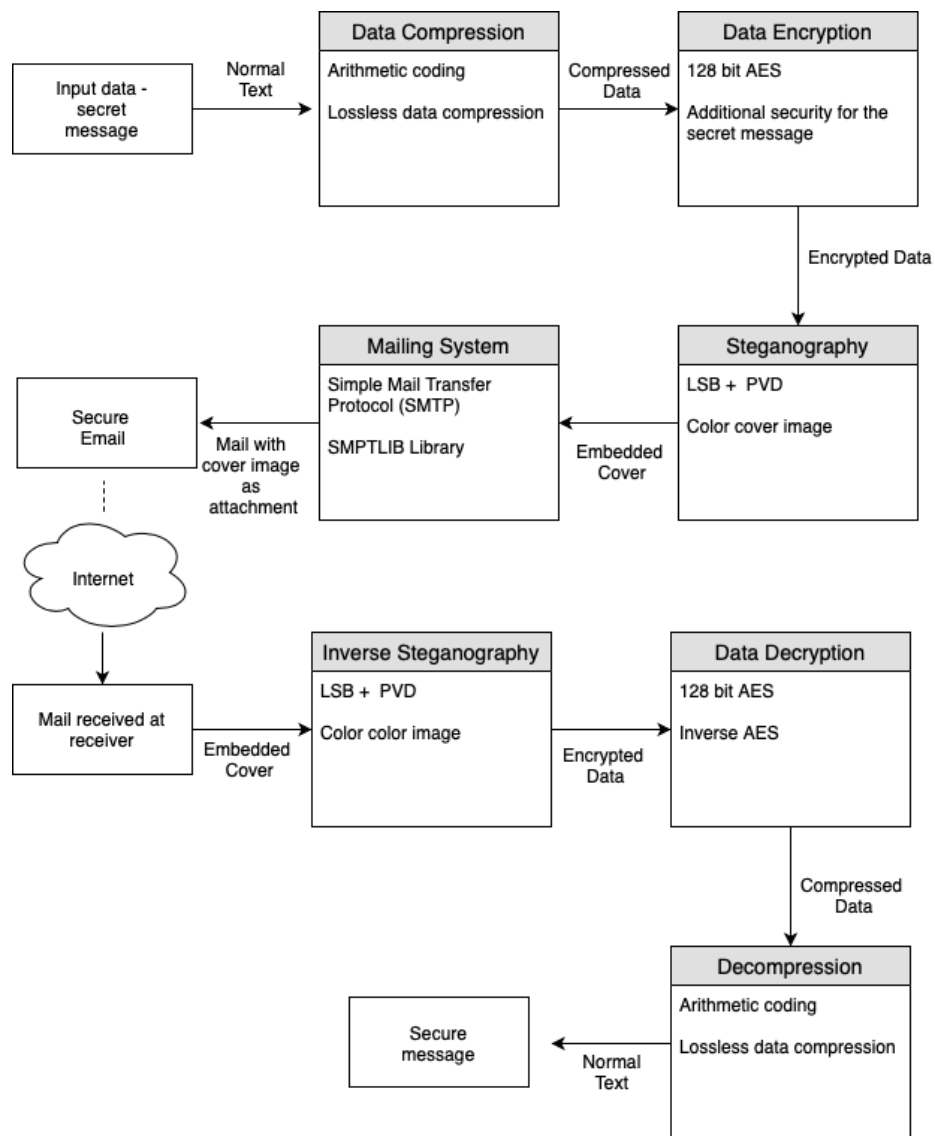


Figure 5.3: DFD Level 2

# Chapter 6

## Implementation

### 6.1 Algorithms

#### 6.1.1 Data Compression

Arithmetic coding is a common algorithm used in both lossless and lossy data compression algorithms. It is an entropy encoding technique, in which the frequently seen symbols are encoded with fewer bits than rarely seen symbols. It has some advantages over well-known techniques like Huffman coding.

---

**Algorithm 1:** Compress - Arithmetic Coding

---

1. ARITHMETIC ENCODING ALGORITHM.

Call encode-symbol repeatedly for each symbol in the message. Ensure that a distinguished "terminator" symbol is encoded last, then transmit any value in the range [low, high).

2. encode-symbol(symbol, cum-freq)

3. range = high - low

4. high = low + range \* cum-freq[symbol-1]

5. low = low + range\*cum-freq[symbol]

6. ARITHMETIC DECODING ALGORITHM.

"Value" is the number that has been received. Continue calling decode-symbol until the terminator symbol is returned.

7. decode-symbol(cum-freq)

8. find symbol such that cum-freq[symbol]  $\geq$  (value-low)/(high-low)  $\geq$  cum-freq[symbol-1]

This ensures that value lies within the new (low, high) range that will be calculated by the following lines of code.

9. range = high - low



- 
10.  $high = low + range * cum\_freq[symbol-1]$
  11.  $low = low + range * cum\_freq[symbol]$
  12. return symbol
- 

### 6.1.2 Encryption

AES is based on a design principle known as a substitutionpermutation network, and is efficient in both software and hardware. AES operates on a 4 × 4 column-major order array of bytes, termed the state. Most AES calculations are done in a particular finite field.

---

#### Algorithm 2: Encryption - AES

---

1. KeyExpansionround keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
  2. Initial round key addition:
  3. AddRoundKeyeach byte of the state is combined with a block of the round key using bitwise xor.
  4. 9, 11 or 13 rounds:
    - (a) SubBytesa non-linear substitution step where each byte is replaced with another according to a lookup table.
    - (b) ShiftRowsa transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    - (c) MixColumnsa linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
    - (d) AddRoundKey
  5. Final round (making 10, 12 or 14 rounds in total):
    - (a) SubBytes
    - (b) ShiftRows
    - (c) AddRoundKey
- 

---

#### Algorithm 3: Decryption - AES

---

1. Step 1: Inverse\_SubBytes()
2. Step 2: Inverse\_ShiftRows()

3. Step 3: Inverse\_MixColumns()
4. Step 4: Inverse\_AddRoundkey()
5. Step 3 is not performed for last round. All the encryption steps should be performed in reverse to achieve decryption.

### 6.1.3 Steganography

After compression and encryption of secret message, modified approach of Khodaei and Faez's LSB+PVD method is applied to embed the message in cover image. In the proposed embedding method, the cover image is divided into non-overlapping pixel blocks of 3x3 or 3x3 + 2x2 pixel blocks as per the cardinality of the cover image.

#### Algorithm 4: Embedding

1. Step 1: Read the cover image I and divide it into 3x3 non-overlapping blocks. However, if the height and width of the image are not divisible by 3x3 blocks, the image is divided into 3x3 plus 2x2 blocks. For example, a 512x512 pixel image is partitioned into 28900 blocks of 3x3 pixels and 511 blocks of 2x2 pixels.
2. Step 2: Read block  $B_i$  and name the pixels as  $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9$ . For 2x2 pixel blocks, the pixels would be named as  $a_5, a_6, a_7, a_8$ .
3. Step 3: Select  $a_5$  as base pixel or reference pixel.
4. Step 4: For reference pixel, embed 3 bits directly into 3 LSBs of  $a_5$  to get  $a_5'$ .
5. Step 5: Calculate the difference  $d_i$  for all pixel values except for  $a_5$ . i.e.  $d_i = \text{mod}(a_5 - a_i)$
6. Step 6: Compute the ranges  $R_i$  to which  $d_i$  belongs. As stated in the range table, compute  $C_i$  i.e. number of bits to be hidden.
7. Step 7: Read the  $C_i$  bits continuously from S secret message as  $S_i$ .
8. Step 8: Replace  $C_i$  bits of  $a_i$  with  $S_i$  to obtain  $a_i'$ .

**Algorithm 5:** Inverse Steganography

---

1. Read the cover image  $I$  and divide it into  $3 \times 3$  non overlapping blocks. However, if the height and width of the image are not divisible by  $3 \times 3$  blocks, the image is partitioned into  $3 \times 3$  plus  $2 \times 2$  blocks. For example, a  $512 \times 512$  pixel image is partitioned into 28900 blocks of  $3 \times 3$  pixels and 511 blocks of  $2 \times 2$  pixels.
  2. Read block  $B_i$  and name the pixels as  $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9$ . For  $2 \times 2$  pixel the pixels would be named as  $a_1, a_2, a_3, a_4$ .
  3. Extract  $k$ - secret bits from  $k$ - LSBs of  $a$  and name it as  $S_c$ .
  4. Calculate the difference  $d_i$
  5. Compute the ranges  $R$  to which  $d$  belongs. As stated in range table obtain the  $C$ .
  6. Then extract  $C$ th rightmost LSBs of  $a_i$  and name it as  $S_i$  for all the pixels of  $B_i$ .
  7. Now, concatenate  $S_c$  and all  $S_i$  to obtain secret message.
-

## 6.2 Development Tools

### 6.2.1 Git

Git is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for software development, but it can be used to keep track of changes in any file. As a distributed revision control system it is aimed at speed, data integrity and support for distributed, non-linear workflows.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano.

As with most other distributed version control systems, and unlike most clientserver systems, every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, independent of network access or a central server.

Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2.

Usage of Git in this project helped in keeping track of code and issues. It also helped integrating the work during each development phase. Without Git the project would be messy and combining the work of all contributors would have been more time-consuming than necessary.

### 6.2.2 Visual Studio Code

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS.[7] It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences. The source code is free and open source and released under the permissive MIT License.[8] The compiled binaries are freeware and free for private or commercial use.[9]

Visual Studio Code is based on Electron, a framework which is used to deploy Node.js applications for the desktop running on the Blink layout engine. Although it uses the Electron framework,[10] the software does not use Atom and instead employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).[11]

In the Stack Overflow 2019 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 50.7% of 87,317 respondents claiming to use it.

Visual Studio Code was announced on April 29, 2015, by Microsoft at the 2015 Build conference. A Preview build was released shortly thereafter.

# Chapter 7

## Testing

### 7.1 Testing Methodologies

Software Testing Methodology is defined as strategies and testing types used to certify that the Application Under Test meets user expectations. Test Methodologies include functional and non-functional testing to validate the Application Under Test. Each testing methodology has a defined test objective, test strategy and deliverables. We performed the following testing: Unit testing, Integration testing, System testing.

### 7.2 Unit Testing

Unit testing finds problems early in the development cycle. This includes both bugs in the programmers implementation and flaws or missing parts of the specification for the unit. The process of writing a thorough set of tests forces the author to think through inputs, outputs, and error conditions, and thus more crisply define the units desired behaviour. The cost of finding a bug before coding begins or when the code is first written is considerably lower than the cost of detecting, identifying, and correcting the bug later; bugs may also cause problems for the end-users of the software.

We did unit testing on the following modules:-

- Compression Module - A text file is given as input to the module and compression is performed an average 50% compression is achieved.
- Encryption Module - The unit is tested by passing a text file as input and AES encrypted output file is obtained.
- Embedding Module - A cover image and text file is given as input and embedded cover image is obtained as output.
- Sending Mails - The module is tested by sending mails to the given ID's with given files as attachments.

### 7.3 Integration Testing

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These design items, i.e., assemblages (or group of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test whether all the components within assemblages interact correctly.

We integrated our modules one by one and tested the partially integrated system.

### 7.4 System Testing

System testing is performed on the entire system in the context of Functional Requirement Specifications (FRS) and/or System Requirement Specification (SRS). System testing tests not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond what is defined in the software/hardware requirements specification(s).

All modules were integrated at the end of integration testing and the entire system was tested by sending a textual message to a given mail ID using the mentioned processing steps.

## Chapter 8

# User Interface

### 8.1 GUI Overview

The main page of the graphical user interface consists of options for embedding and extracting data along with quitting the application. The GUI is developed using 'tkinter'. Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python. The name Tkinter comes from Tk interface.

The main options included in the home page of the application are:

- Compose Mail
- Extract data from Cover Image

The main usage of the application is shown in Figure.



Figure 8.1: Basic UI



## 8.2 Main GUI Components

### 8.2.1 Compose Mail

In this phase the textual data to be sent is given to the application as text in the email content field along with recipient address and subject.

There is alert boxes to alert user about ignored or incorrect entries to the input fields.

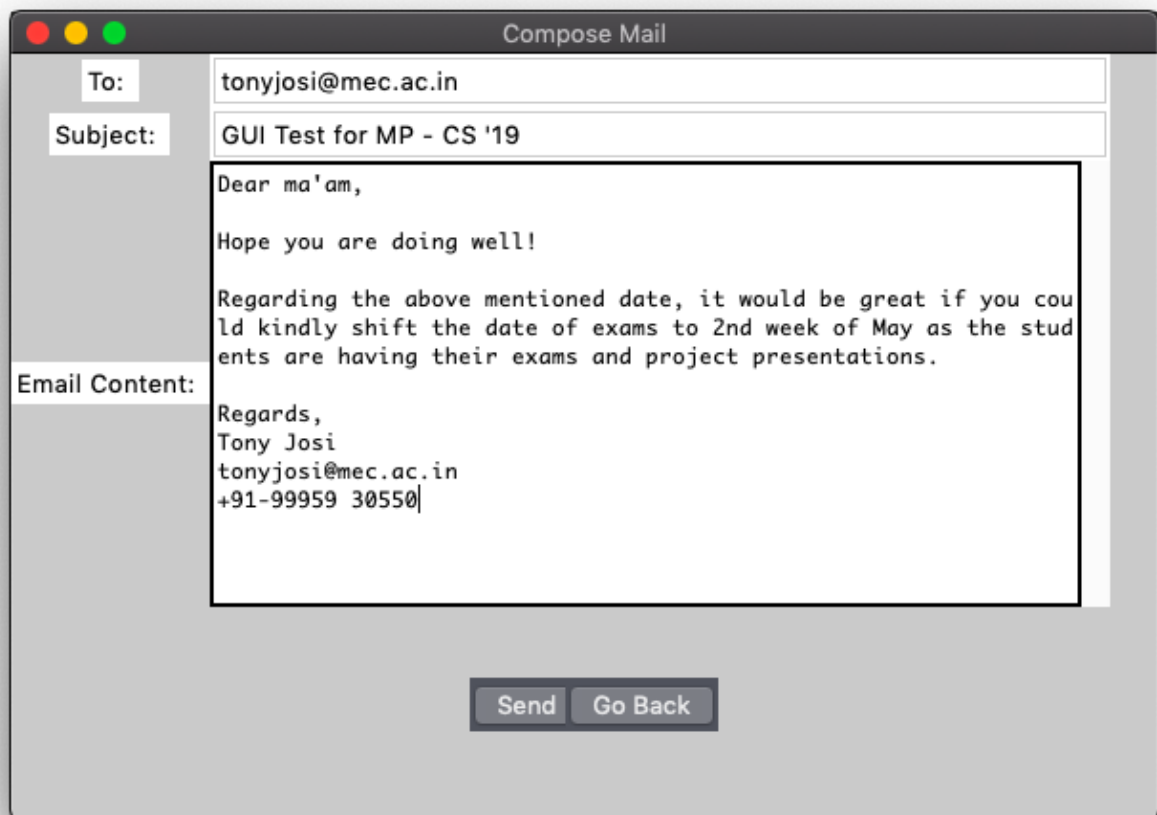


Figure 8.2: Compose Mail

### 8.2.2 Alert Boxes for Compose Mail Window

There is alert boxes to alert user about ignored or incorrect entries to the input fields. During display of alert boxes the state of the compose mail window is saved for the user to correct the errors in the input.

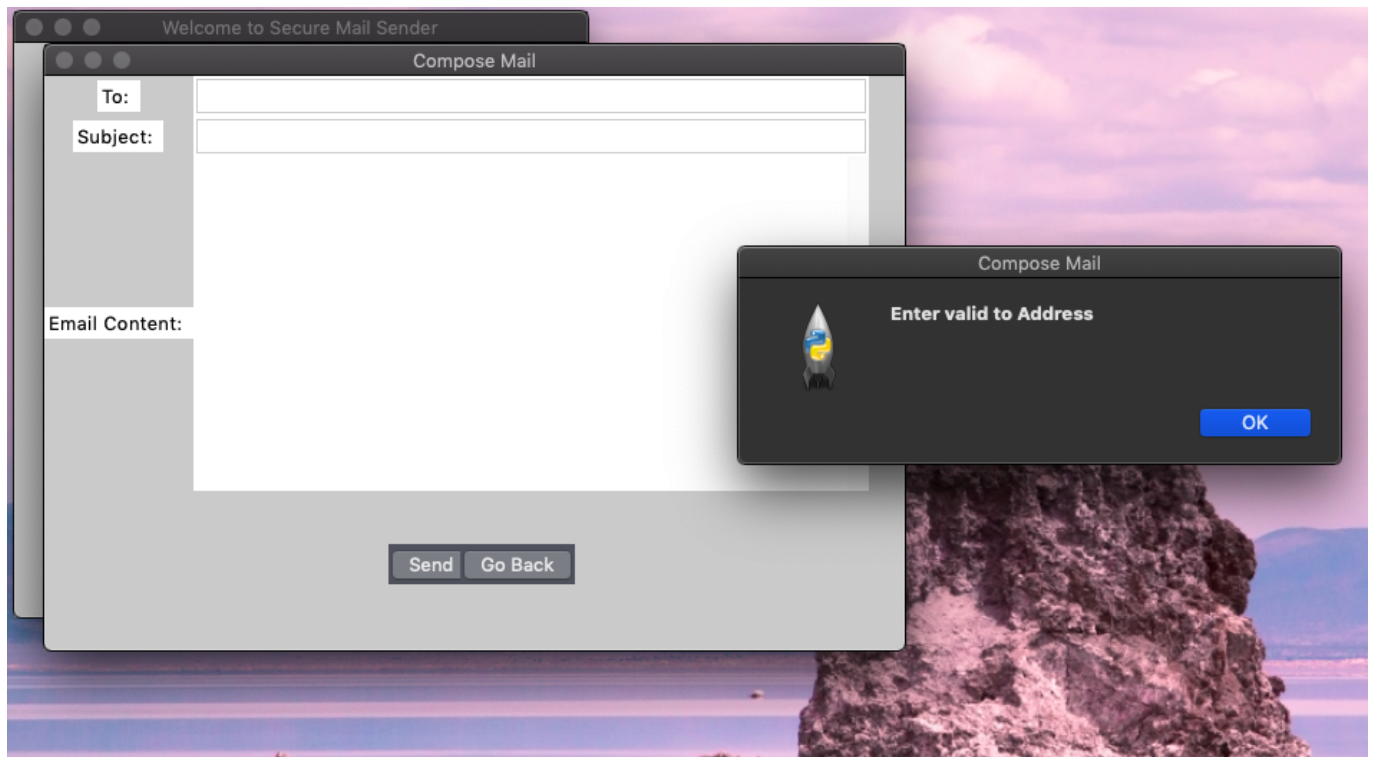


Figure 8.3: Alert Boxes for Compose Mail Window

### 8.2.3 Extracting Encrypted data from Cover Image

The window prompts for path location of the cover image. After pasting the path and pressing extract button, extraction starts. This phase extracts the data from cover image and saves it in new file then the content of the file is displayed as email content in the GUI

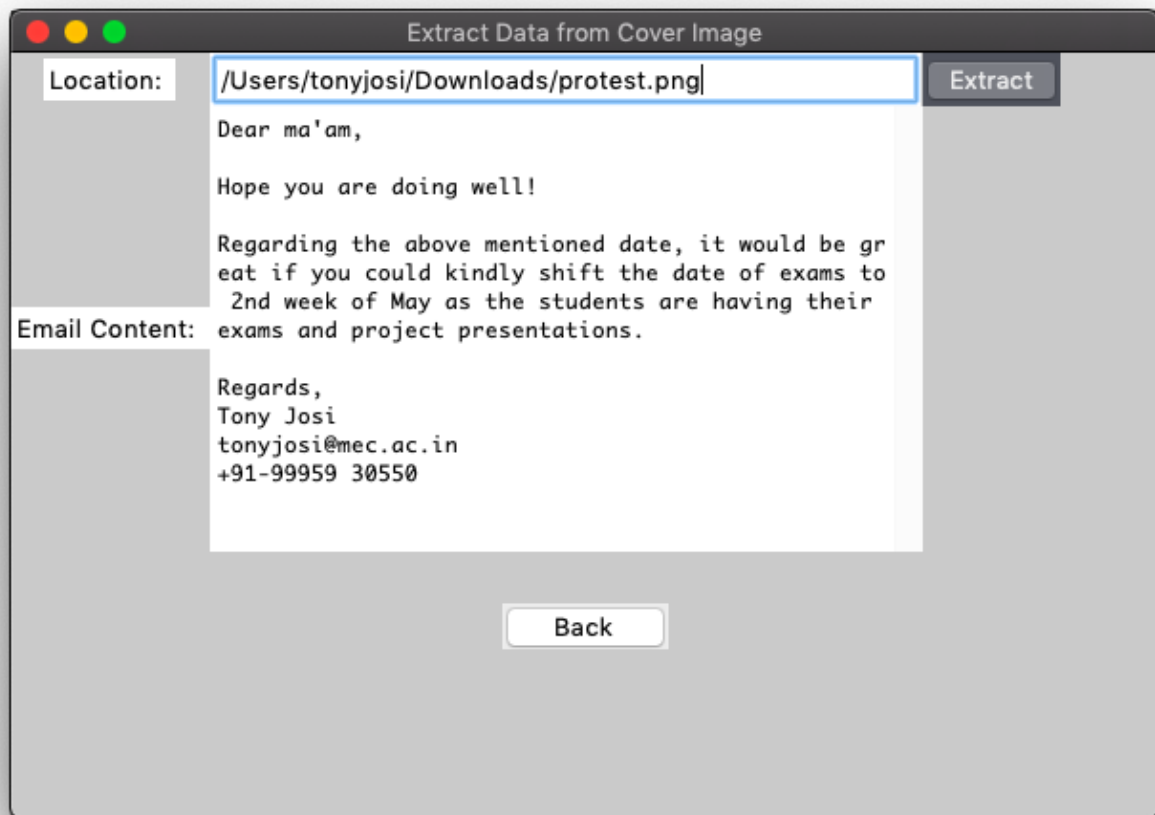


Figure 8.4: Extracting Encrypted data from Cover Image

### 8.2.4 Alert Boxes for Extract Data Window

There is alert boxes to alert user about ignored or incorrect entries to the input fields. If the path name given is empty or incorrect the alert is presented.

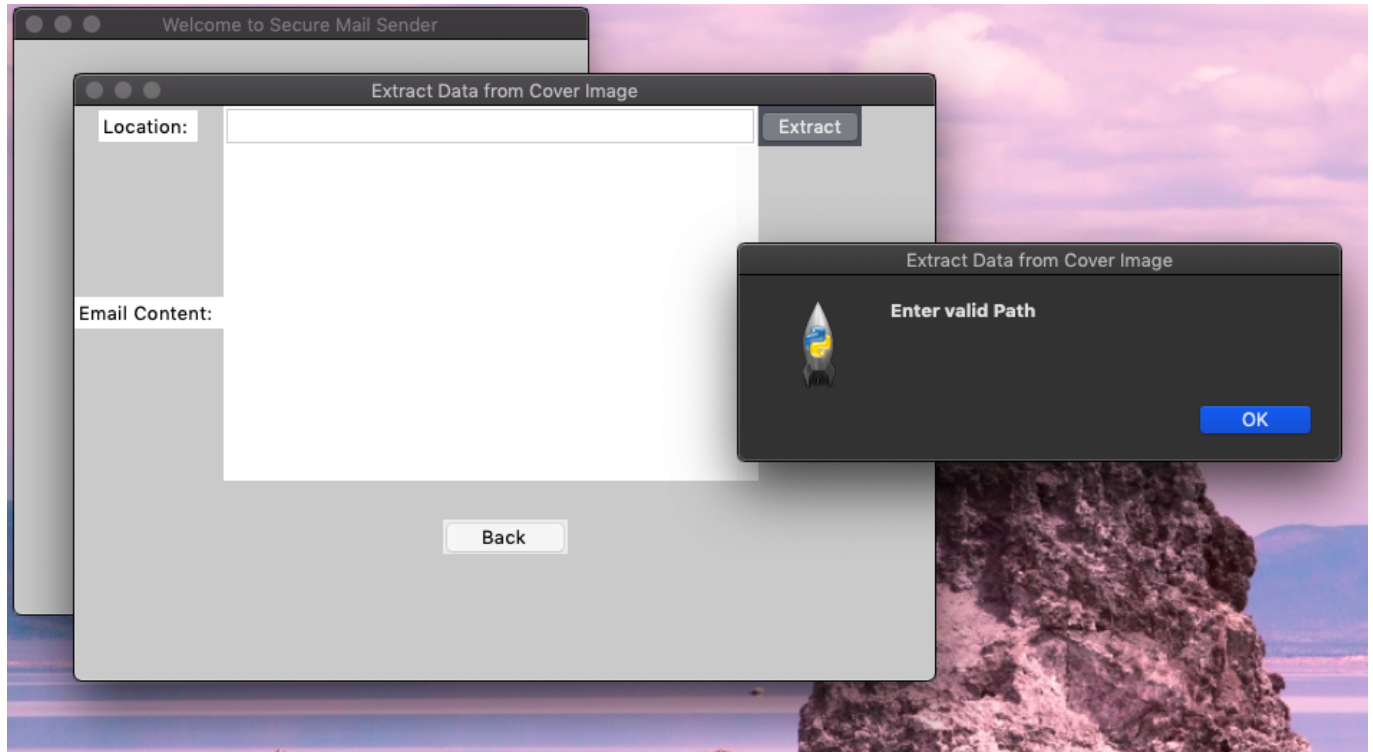


Figure 8.5: Alert Boxes for Extract Data Window

### 8.2.5 Alert Boxes for Successfully Sending of Mail

Alerts if mail is sent successfully.

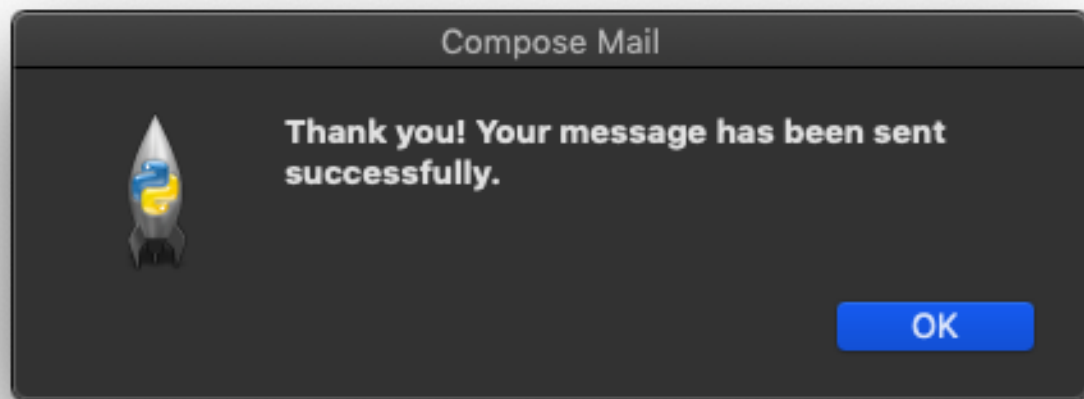


Figure 8.6: Alert Boxes for Successfully Sending of Mail

## Chapter 9

# Results

### 9.0.1 Image Quality Comparison

#### Test 1



Figure 9.1: Test Image 1 – Original



Figure 9.2: Test Image 1 – Embedded

- **Mean Square Error:** 0.1996641975308642
- **Peak Signal to Noise Ratio:** 55.1278016384 dB
- **Mean Structural Similarity Index:** 0.9992565858419387

## Test 2



Figure 9.3: Test Image 2 – Original



Figure 9.4: Test Image 2 – Embedded

- **Mean Square Error:** 0.4947555555555556
- **Peak Signal to Noise Ratio:** 51.1868968132 dB
- **Mean Structural Similarity Index:** 0.9996629898989822

### Test 3



Figure 9.5: Test Image 3 – Original





Figure 9.6: Test Image 3 – Embedded

- **Mean Square Error:** 0.40630123456790124
- **Peak Signal to Noise Ratio:** 52.0423221891 dB
- **Mean Structural Similarity Index:** 0.9994692877039093

#### Test 4



Figure 9.7: Test Image 4 – Orginal

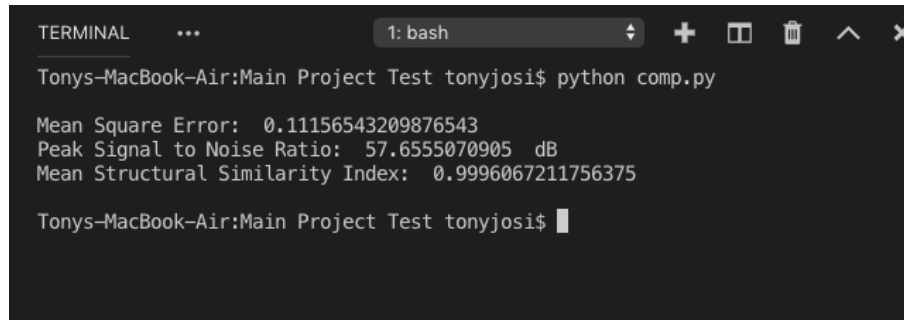


Figure 9.8: Test Image 4 – Embedded

- **Mean Square Error:** 0.16094814814814815
- **Peak Signal to Noise Ratio:** 56.0639437676 dB
- **Mean Structural Similarity Index:** 0.9988305404933313

### 9.0.2 Image Quality Comparison with Varying Input File Size done on Same Cover Image

#### Test 1 - Input Size: 644 Bytes



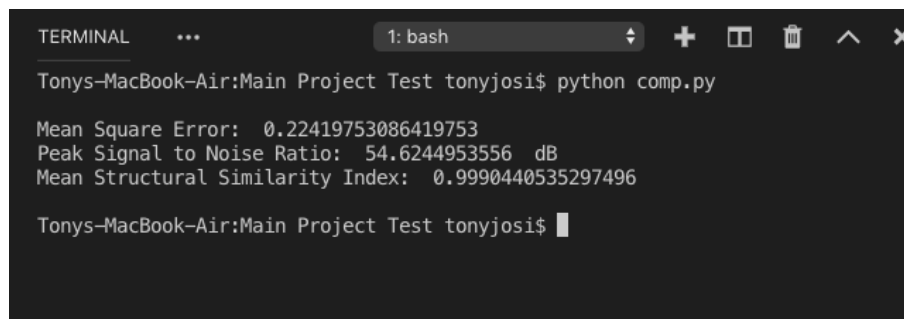
```
TERMINAL  ...  1: bash
Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error: 0.11156543209876543
Peak Signal to Noise Ratio: 57.6555070905 dB
Mean Structural Similarity Index: 0.9996067211756375

Tonys-MacBook-Air:Main Project Test tonyjosi$
```

Figure 9.9: Test 1 - Input Size: 644 Bytes

#### Test 2 - Input Size: 1289 Bytes



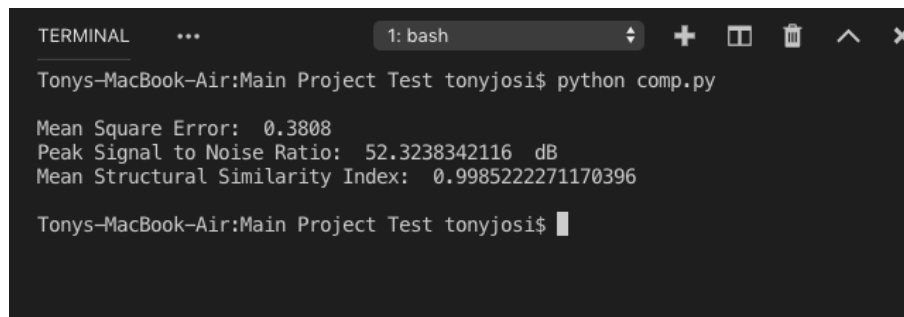
```
TERMINAL  ...  1: bash
Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error: 0.22419753086419753
Peak Signal to Noise Ratio: 54.6244953556 dB
Mean Structural Similarity Index: 0.9990440535297496

Tonys-MacBook-Air:Main Project Test tonyjosi$
```

Figure 9.10: Test 1 - Input Size: 1289 Bytes

#### Test 3 - Input Size: 1934 Bytes

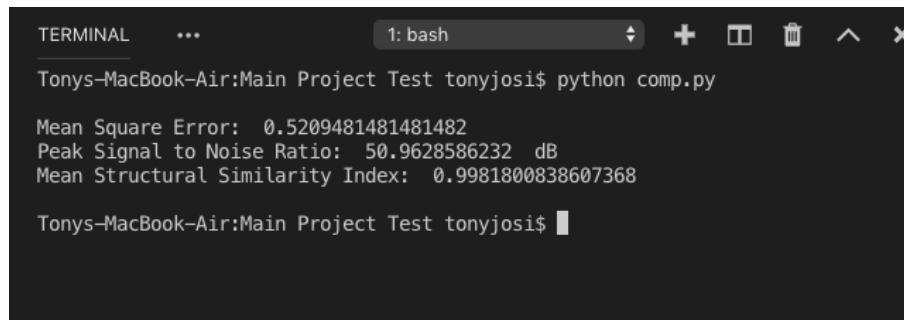


```
TERMINAL  ...  1: bash
Tonys-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error: 0.3808
Peak Signal to Noise Ratio: 52.3238342116 dB
Mean Structural Similarity Index: 0.9985222271170396

Tonys-MacBook-Air:Main Project Test tonyjosi$
```

Figure 9.11: Test 1 - Input Size: 1934 Bytes

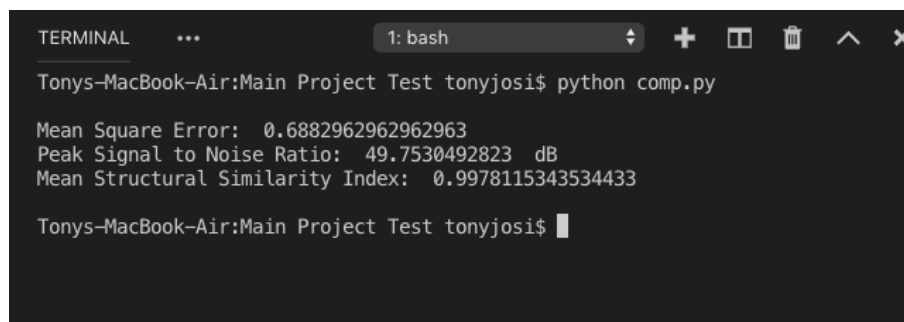
**Test 4 - Input Size: 2579 Bytes**A terminal window titled '1: bash' showing the execution of 'python comp.py'. The output displays three metrics: Mean Square Error, Peak Signal to Noise Ratio, and Mean Structural Similarity Index.

```
Terminal 1: bash
Tony's-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error: 0.5209481481481482
Peak Signal to Noise Ratio: 50.9628586232 dB
Mean Structural Similarity Index: 0.9981800838607368

Tony's-MacBook-Air:Main Project Test tonyjosi$
```

Figure 9.12: Test 1 - Input Size: 2579 Bytes

**Test 5 - Input Size: 3224 Bytes**A terminal window titled '1: bash' showing the execution of 'python comp.py'. The output displays three metrics: Mean Square Error, Peak Signal to Noise Ratio, and Mean Structural Similarity Index.

```
Terminal 1: bash
Tony's-MacBook-Air:Main Project Test tonyjosi$ python comp.py

Mean Square Error: 0.6882962962962963
Peak Signal to Noise Ratio: 49.7530492823 dB
Mean Structural Similarity Index: 0.9978115343534433

Tony's-MacBook-Air:Main Project Test tonyjosi$
```

Figure 9.13: Test 1 - Input Size: 3224 Bytes

### 9.0.3 Plots for Image Quality Comparison

#### Mean Square Error (MSE)

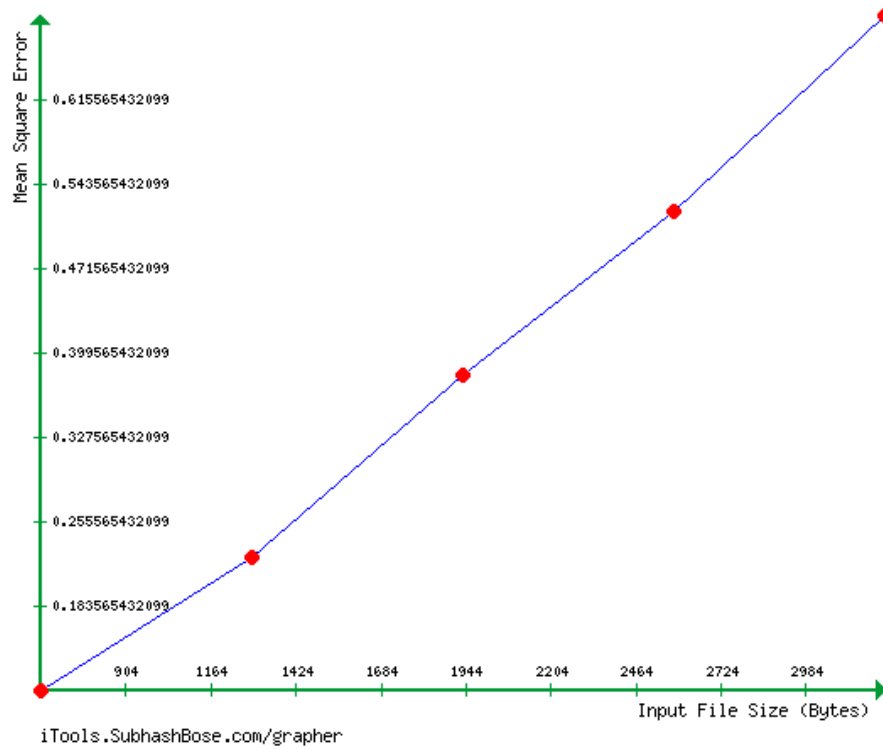


Figure 9.14: Mean Square Error (MSE)

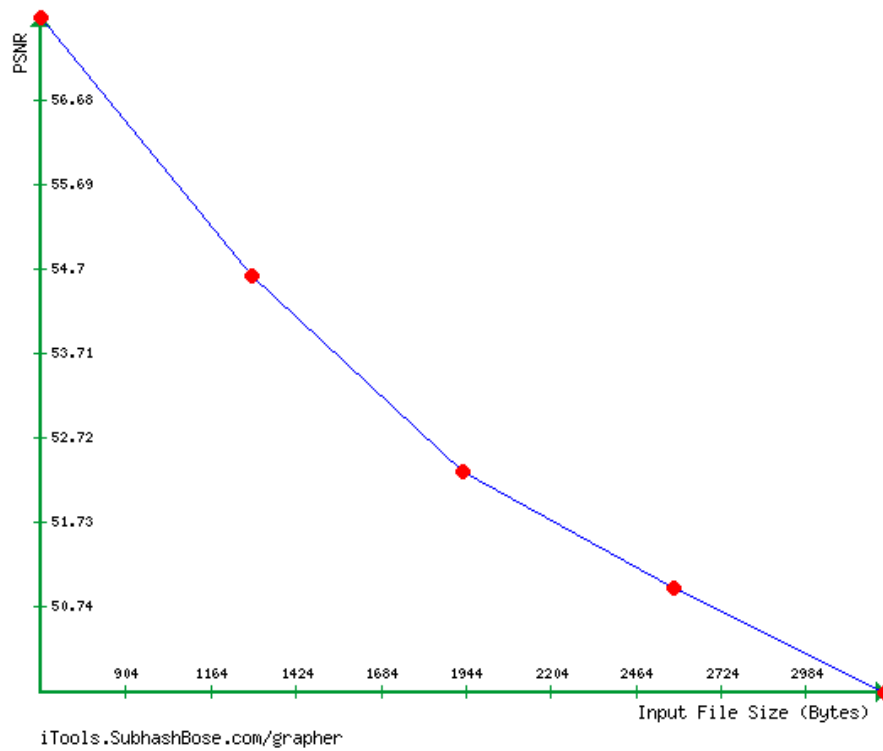
**Peak Signal to Noise Ratio (PSNR)**

Figure 9.15: Peak Signal to Noise Ratio (PSNR)

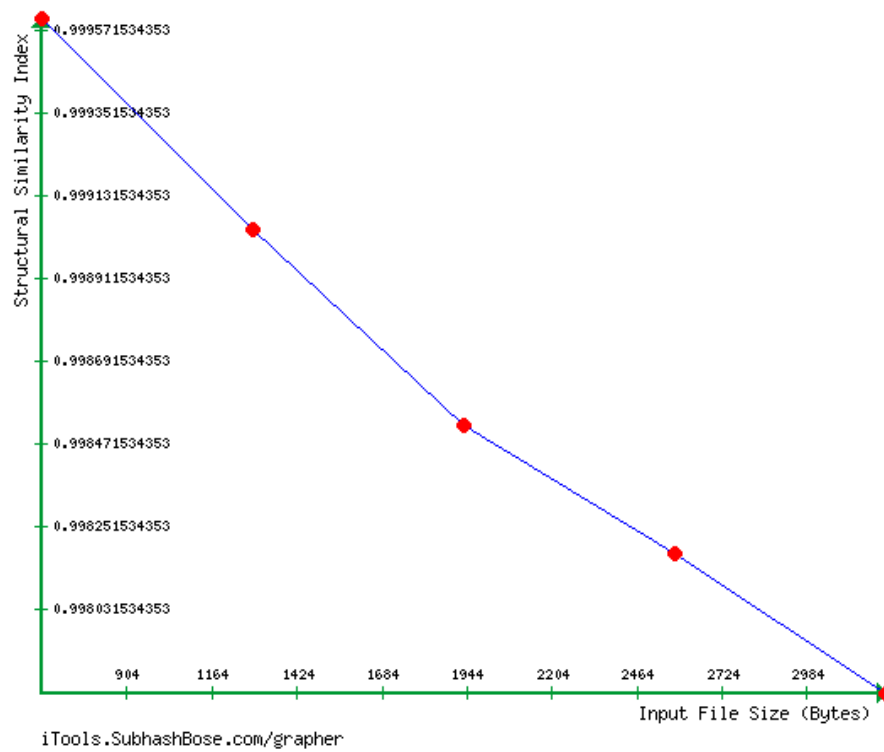
**Structural Similarity**

Figure 9.16: Structural Similarity

## Chapter 10

# Conclusion

A fully functional methodology for sending highly confidential information via e-mails using cryptographic and stenographic methods has been developed. This mode of secure transmission combines Compression, Encryption and Pixel Value Differencing. The data is first compressed to improve efficiency by decreasing size. Arithmetic coding was applied on secret message for lossless compression, which provided 22% higher embedding capacity. Then the data is undergone encryption using AES which provides higher security in the cases of steganalysis attacks.

The encrypted data is then embedded to a cover image using a new steganographic method - PVD (Pixel Value Differencing). Enhanced hiding capacity is obtained by combining LSB and PVD. The PVD and LSB embedding methods are applied on smooth and edge areas of the cover image respectively, after partitioning the cover image into smooth and edge areas. The cover image is partitioned into 1x3 non-overlapping pixel blocks and second pixel of each block is selected as base pixel. The base pixel is subjected to k-bit LSB and first and third pixels of the block are subjected to PVD.

The embedded cover image is sent to the receiver using simple mail transfer protocol library (smtplib), email (pyhton library) and MIME protocols. The cover image along with the embedded message is retrieved by the receiver by using the same SMTPLIB - library the cover image is downloaded to the receiver machine.

The cover image is then undergone inverse steganography, AES decryption and Decompression to obtain the original secret message sent by the sender. The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order Add round key, Mix columns, Shift rows and Byte substitution. Also inverse arithmetic coding is applied to decompress the compressed message after the decryption phase of AES.

The proposed project is a modification of the original mailing platform used for sending emails by improving the security and efficiency. Data compression involved in this project compresses the total data that needs to be send across and thus increases the efficiency of further processes. AES and Steganography further adds the security of the current system.



## Chapter 11

# Future Scope

- In this project, we have considered both the message and the key to be of 128 bits. The strength of the AES algorithm may be enhanced by increasing the key length from 128 bits to 512 bits. As a result, the number of rounds is increased in order to provide a stronger encryption method for secure communication.
- Enhancements in image based steganography can be done using soft computing techniques such as Neural based steganography, Fuzzy and Genetic algorithms based approaches.
- A graphical user interface can be developed to enhance the process of obtaining the data from the user.
- Research can be done to develop features like retrieving mails automatically from the receiver.

# References

- [1] Awdhesh K. Shukla, Akanksha Singh, Balvinder Singh, Amod Kumar "A Secure and High-capacity Data-hiding Method Using Compression, Encryption and Optimized Pixel Value Differencing"
- [2] Sriram, S, Karthikeyan, B, Vaithyanathan, Anishin Raj, M. M "An Approach of Cryptography and Steganography using Rotor cipher for secure Transmission."
- [3] Adrian Kapczyski, Arkadiusz Banasik "Biometric Logical Access Control Enhanced by Use of Steganography Over Secured Transmission Channel"
- [4] Rina Mishra, Atish Mishra, Praveen Bhanodiya "An Edge Based Image Steganography with Compression and Encryption"
- [5] Amritpal Singh, Harpal Singh "An improved LSB based image steganography technique for RGB images"
- [6] Ryan A. Subong, Arnel C. Fajardo, Yoon Joong Kim "LSB Rotation and Inversion Scoring Approach to Image Steganography"
- [7] C. Balasubramanian & S. Selvakumar & S. Geetha "High payload image steganography with reduced distortion using octonary pixel pairing scheme"
- [8] Jitha Raj.T, E.T Sivadasan " Secure transmission of data by splitting image "
- [9] A. SaiKrishna Shankar Parimi, G.Manikandan & Sairam.N. "A clustering based steganographic approach for secure data communication."
- [10] Jayanti Bhadra, M.K. Banga, M. Vinayaka Murthy "Securing Data Using Elliptic Curve Cryptography and Least Significant Bit Steganography"