



**St. Xavier's College
(Autonomous), Kolkata**

PROJECT REPORT

Image Steganography on Encrypted Messages

*Development of an Android Application for smartphones and tablets for:
Encryption of Entered Text Using Vigenère Cipher Inspired Algorithm
Using Steganography on the Encrypted Text by LSB Substitution Method*

Submitted By:

Sayudh Roy

Team Members:

Debayan Mitra

Chahat Gopalika

**Under the Guidance of Dr. Asoke Nath
Department of Computer Science
St. Xavier's College (Autonomous), Kolkata**

Submitted to the **Department of Computer Science**
in partial fulfilment of the requirements for the degree of
Bachelor of Science in Computer Science (Honours)
2014-2017

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled “**Image Steganography on Encrypted Messages**”, submitted to **Department of Computer Science, St. Xavier’s College (Autonomous), Kolkata**, in partial fulfilment of the requirement for the award of the degree of **Bachelor of Science (B.Sc.)** is an original work carried out by **Mr. Sayudh Roy**, Registration No. **A01-1112-0873-14**, under my guidance. The matter embodied in this project is authentic and is genuine work done by the student and has not been submitted whether to this College or to any other Institute for the fulfilment of the requirement of any course of study.

(SAYUDH ROY)

Dated:

Sayudh Roy

Flat: 4A, Acacia Apartments
12/1A Anil Maitra Road
Kolkata: 700 029

Registration No.:
A01-1112-0873-14

(DR. ASOKE NATH)

Dated:

Dr. Asoke Nath
Associate Professor

Dept. of Computer Science
St. Xavier’s College
(Autonomous), Kolkata

30, Mother Teresa Sarani
Kolkata: 700 016

ROLES AND RESPONSIBILITIES FORM

Name of the Project: Image Steganography on Encrypted Messages

Dated: 31st March, 2017

Sl. No.	Member	Role	Tasks And Responsibilities
1.	Sayudh Roy	GUI Designer, Steganography Coder	1. Programming in Java for the Steganography Function. 2. Designing the Android Interface on IntelliJ Idea JetBrains. 3. Documentation
2.	Debayan Mitra	Team Coordinator, Quality Manager	1. Debugging the codes. 2. Product testing with various input/outputs. 3. Documentation
3.	Chahat Gopalika	Encryption Coder, GUI Designer	1. Programming in Java for the Cipher Text Function. 2. Determining the Problem Definition and Requirements Analysis. 3. Documentation

Name and Signature of the **Project Team Members:**

Sayudh Roy

Debayan Mitra

Chahat Gopalika

Signature of the Professor

Dr. Asoke Nath

Dated

ABSTRACT

Although the fields of steganography and cryptography are associated with one another, there is a distinction to be made. Cryptography is the art of jumbling a message so that a would-be eavesdropper cannot interpret the message. Steganography, on the other hand, is the art of hiding a message so that a would-be eavesdropper is unaware of the message's presence.

Steganography is derived from the Greek word *steganographic* which means covered writing. It is the science of secret communication.

The goal of steganography is to hide the existence of the message from unauthorized party. The modern secure image steganography presents a task of transferring the embedded information to the destination without being detected by the attacker. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the internet. For hiding secret information in images, there exist a large variety of steganographic techniques some are more complex than others and all of them have respective strong and weak points.

In this project, an image based steganography has been performed that uses Least Significant Bits (LSB) technique on images to enhance the security of the communication. In the LSB approach, the basic idea is to replace the Least Significant Bits (LSB) of the cover image with the Bits of the messages to be hidden without destroying the property of the cover image significantly. The LSB-based technique is the most challenging one as it is difficult to differentiate between the cover-object and stego object if few LSB bits of the cover object are replaced.

Also, to enhance the application's portability and ease of use, the Steganography application has been developed Android platform.

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my hearty gratitude to my guide and supervisor **Dr. Asoke Nath**, Associate Professor of **Department of Computer Science, St. Xavier's College, Kolkata**, whose constant guidance, encouragement and support made the successful completion of my project possible.

I am obliged to all the professors of the Department of Computer Science in St. Xavier's College for instilling in me the basic knowledge about the field and the urge to keep learning and exploring new areas in this subject, that greatly benefitted me while carrying out the project and achieving this goal. The constant guidance and encouragement by the **Professors** and the co-operation of the **Laboratory Staff Members** has been extremely instrumental in helping our project reach successful completion.

Lastly, I am grateful to my project partners, Debayan Mitra and Chahat Gopalika, for their relentless support, constant brainstorming and cooperation without which, the completion of this project would have been virtually impossible.

With regards,

Sayudh Roy

TABLE OF CONTENTS

SERIAL NO.	TITLE	PAGE NO.
1.	Abstract	iv
2.	Acknowledgements	v
3.	Introduction	1
4.	Survey of Technologies	5
5.	Requirement and Analysis	11
6.	System Design	20
7.	Implementation and Testing	34
8.	Results and Discussions	46
9.	Conclusions	48
10.	References	50
11.	Glossary	52

TABLE OF FIGURES

SERIAL NO.	TITLE	PAGE NO.
1.	Figure 1: The Gantt Chart	17
2.	Figure 2: The Pert Chart	18
3.	Flowchart 1: Encryption	24
4.	Flowchart 2: Decryption	25
5.	Flowchart 3: Character to Bits (Array)	26
6.	Flowchart 4: Bits (Array) to Character	27
7.	Flowchart 5: Embedding of Text in Image	28
8.	Flowchart 6: Extraction of Text from Image	29
9.	Table 1: Observed Times for the Sample Inputs/Outputs	45

INTRODUCTION

Background

We live in a world where everything is increasingly digitized to the extent where computers, the internet and the digital platform has become an indispensable part of everyday life. However with the advent of this growth in the usage of digital services and exchange of messages and sensitive information digitally, the dangers of such information becoming vulnerable to cyber theft or hacking have also increased rapidly. For this reason, it is imperative that we devise and keep improving on ways to protect such sensitive information. Methods like encryption are used to achieve exactly that. In cryptography, encryption is the process of encoding a message or information in such a way that only authorized parties can access it. Encryption does not in itself prevent interference, but denies the intelligible content to a would-be interceptor. In an encryption scheme, the intended information or message, referred to as plaintext, is encrypted using an encryption algorithm, generating cipher text that can only be read if decrypted. Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The word *steganography* combines the Greek words *steganos*, meaning "covered, concealed, or protected", and *graphein* meaning "writing". The first recorded use of the term was in 1499 by Johannes Trithemiusin. His *Steganographia*, a treatise on cryptography and steganography, disguised as a book on magic. Generally, the hidden messages appear to be (or be part of) something else: images, articles, shopping lists, or some other *cover text*. For example, the hidden message may be in invisible ink between the visible lines of a private letter. Some implementations of steganography that lack a shared secret are forms of security through obscurity, whereas key-dependent steganographic schemes adhere to Kerckhoffs's principle.

Steganography is a special method of writing hidden messages in such a way that no-part from the sender and the receiver can even realizes that there is a hidden message. Today, the term Steganography includes the embedding of digital information within computer files. For example, the sender may embed a big text file in an image in such a way that there should not be any significant change in the image. We can embed even some voice or image or text in any host file which may be an image file or may be another sound file etc.

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the Internet. For hiding secret information in images, there exists a large variety of Steganographic techniques, some are more

complex than the others and all of them have respective strong and weak points. Different applicants have different requirements and hence different Steganography techniques are used. For example, some applications may require absolute invisibility of the secret information, while others require a larger secret message to be hidden.

The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages—no matter how unbreakable—arouse interest, and may in themselves be incriminating in countries where encryption is illegal. Thus, whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned with concealing the fact that a secret message is being sent, as well as concealing the contents of the message.

In this project, we have implemented the use of Steganography on Android devices. The operating system: Android, is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. With Android being a major player in the current world of information technology, and almost every individual having an Android device (be it a smartphone or tablet), we decided it would be best to design an application on the Android Platform to reach out to maximum number of consumers.

The Objectives

The aim of this project is to implement **Image Steganography** on any format of image files for Android devices. The levels of security provided to the users is increased by encrypting the messages to be embedded, using a polyalphabetic substitution cipher technique which has been derived from the Vigenère Cipher method.

Purpose, Scope and Applicability

Purpose: The purpose of embarking on this project is primarily two fold, namely:

- The dependence on the digital platform when it comes to exchange of messages and sensitive information is perpetually on the rise. However, along with it, the increasing threat of data theft and interception of data by an unwanted third party is also an undeniable part of modern day reality. Thus there is a constant need to upgrade our methods of encryption so as to protect the data that is present on the digital platform. This is why we think that a project on steganography is very relevant in terms of the complexities posed and the applications it has in keeping our data safe.

- Today a majority of the world population is getting connected via the android platform and the number of people who are using Android or has access to Android is always on the rise, So we believe that doing our image steganography project on the Android platform helps us reach out to a wider audience who may find use of this app to protect their data.

Scope: In this project, we have primarily used the following algorithms or methods for implementing each of the sub-sections:

- For the encryption of the plain text to the cipher text and vice-versa we have used an algorithm which has been inspired from the algorithm developed by French cryptographer, Blaise de Vigenère.
- The embedding of the cipher text into the image file is performed by a method known as the LSB Substitution method.

Applicability: The android application which we have designed to implement steganography will greatly contribute to users being able to embed encrypted messages into images due to the portability provided by the android operating system as it is widely available in smartphones and tablets and will reach out to maximum consumers. The application primarily serves to allow users to send messages to one and another without the contents of the actual message being exposed to a third-party. This purpose has been fulfilled by encrypting the said message using a polyalphabetic cipher algorithm and then embedding it in an image of the user's choice. Once the text has been embedded in the image, the output image (the one which will have the encrypted text embedded within it) will almost resemble the original image, with minor changes in its picture composition which is seldom noticeable to the naked eye. That image can now be sent the receiver using any networking medium, and the receiver can acquire the required message by using the same application's other module. Once the receiver supplies the correct key, the application will decrypt the hidden text and return it to the receiver.

Achievements

By taking up this project on Steganography, we believe that we have definitely learnt a lot about various aspects of cryptography as well as having learnt a great deal about development of android applications. By developing the encryption and decryption of the plain text as entered by the user into cipher text, we had to review a great deal of existing literature on mono-alphabetic and polyalphabetic cipher algorithms. This has strengthened our base on encryption

processes and various methods applied for the same over the years. Next, we delved into core programming in the java programming language for the steganography process. This enabled us to learn about bit operations and manipulations and also file handling. The algorithm we used for the steganography made it possible for us to use even audio files for the process but we constrained ourselves into keeping it to just image files for the benefit of our android application. The development of the android application was perhaps the most challenging part in our entire project as it dealt with learning to create an application for an operating system we are so familiar with yet rather unacquainted with its inner nitty-gritties. But, we successfully managed to overcome the rather overwhelming experience of dealing with the Android operating system and managed to design and develop an application which was albeit rather basic to look at, but fully functional.

In our project, our original goals consisted of developing and using algorithms for the encryption and decryption of text and embedding and extraction of the same text into image files. It also comprised development of an android application which would do the above and then convert the encrypted image into a QR code for easier portability. This QR code would then be scanned by the application again for the decryption process.

We have successfully managed to achieve most of our targets for this project by successfully developing codes which transform the user's plain text into cipher text and vice versa and also the embedding of the same. The above have been achieved via coding in the Java programming language and the application development was done by using IntelliJ Idea JetBrains.

Although we had mentioned in our project synopsis that we were going to incorporate another module into our project which involved the transformation of the stego-image into a QR code and the android application would be able to scan and retrieve the image from the same, we were forced to opt out of it due to the lack of time and our limited scope in the field of android. Having said that, we have taken it upon ourselves to work on our limitation and try to achieve the original goal when we are able to provide more time.

Organization of Report

This report encapsulates the entire development process of the android application for the benefit of the reader. It outlines the various departments we had to research on and work on for the completion of this project. It gives the reader an essence of Steganography through the years, the review of literature that we embarked upon which inspired to take up this topic. It then will give you an essence of the processes involved in development and testing and finally the codes which were needed to implement the entire process.

SURVEY OF TECHNOLOGIES

The earliest recordings of Steganography were by the Greek historian Herodotus in his chronicles known as "Histories" and date back to around 440 BC. Herodotus recorded two stories of Steganographic techniques during this time in Greece. The first stated that King Darius of Susa shaved the head of one of his prisoners and wrote a secret message on his scalp. When the prisoner's hair grew back, he was sent to the King's son in law Aristogoras in Miletus undetected. The second story also came from Herodotus, which claims that a soldier named Demeratus needed to send a message to Sparta that Xerxes intended to invade Greece. Back then, the writing medium was text written on wax-covered tablets. Demeratus removed the wax from the tablet, wrote the secret message on the underlying wood, recovered the tablet with wax to make it appear as a blank tablet and finally sent the document without being detected. Romans used invisible inks, which were based on natural substances such as fruit juices and milk. This was accomplished by heating the hidden text, thus revealing its contents. Invisible inks have become much more advanced and are still in limited use today. During the 15th and 16th centuries, many writers including Johannes Trithemius (author of *Steganographia*) and Gaspari Schotti (author of *Steganographica*) wrote on Steganographic techniques such as coding techniques for text, invisible inks, and incorporating hidden messages in music. Between 1883 and 1907, further development can be attributed to the publications of Auguste Kerckhoff (author of *Cryptographic Militaire*) and Charles Briquet (author of *Les Filigranes*). These books were mostly about Cryptography, but both can be attributed to the foundation of some Steganographic systems and more significantly to watermarking techniques. During the times of WWI and WWII, significant advances in Steganography took place. Concepts such as null ciphers (taking the 3rd letter from each word in a harmless message to create a hidden message, etc), image substitution and microdot (taking data such as pictures and reducing it to the size of a large period on a piece of paper) were introduced and embraced as great Steganographic techniques. In the digital world of today, namely 1992 to present, Steganography is being used all over the world on computer systems. Many tools and technologies have been created that take advantage of old Steganographic techniques such as null ciphers, coding in images, audio, video and microdot. With the research this topic is now getting we will see a lot of great applications for Steganography in the near future.

As we saw, the application and usage of steganography has evolved through the ages and its importance and stature in the field of cryptography goes on increasing as we speak.

With advancements in digital communication technology and the growth of computer power and storage, the difficulties in ensuring individuals privacy become increasingly challenging. The degrees to which individuals appreciate privacy differ from one person to another. Various methods have been investigated and developed to protect personal privacy. Encryption is

probably the most obvious one, and then comes steganography. Encryption lends itself to noise and is generally observed while steganography is not observable. The term steganography refers to the art of covert communications. Steganography's aim is to make the secret communication undetectable, that is, to hide the presence of the secret message. It modifies the carrier in an imperceptible way only so that it reveals nothing neither the embedding of a message nor the embedded message itself. The recent development of the Internet has brought new attention to steganography. The interest in steganography has been enhanced recently by the emergence of commercial espionage and the growing concerns about homeland security due to terrorism.

Image steganography is the art of information hidden into cover image, is the process of hiding secret message within another message. The word steganography in Greek means “Covered Writing”. The information hiding process in a steganography with different techniques includes identifying cover mediums redundant bits. The embedding process creates a steganography medium by replacing the redundant bits with data from the hidden message. During the process of hiding the information three factors must be considered that are capacity it includes amount of information that can be hidden in the cover medium. Security implies to detect hidden information and Robustness to the amount of modification the steganography medium can withstand before an adversary can destroy hidden information.

Survey of Different Methods of Steganography

Least Significant Bit (LSB) Substitution Technique

In LSB steganography, the least significant bits of the cover media's digital data are used to conceal the message. The simplest of the LSB steganography techniques is LSB replacement. LSB replacement steganography flips the last bit of each of the data values to reflect the message that needs to be hidden.

Consider an 8-bit grayscale bitmap image where each pixel is stored as a byte representing a gray scale value. Suppose the first eight pixels of the original image have the following gray scale values:

11010010
01001010
10010111
10001100
00010101
01010111
00100110
01000011

To hide the letter C whose binary value is 10000011, we would replace the LSBs of these pixels to have the following new grayscale values:

11010011
01001010
10010110
10001100
00010100
01010110
00100111
01000011

Note that, on average, only half the LSBs need to change. The difference between the cover (i.e. original) image and the steganography image will be hardly noticeable to the human eye. However, one of its major limitations is small size of data which can be embedded in such type of images using only LSB. LSB is extremely vulnerable to attacks. LSB techniques implemented to 24 bit formats are difficult to detect contrary to 8 bit format.

Advantages of Spatial Domain LSB Technique

1. Degradation of the original image is not easy.
2. Hiding capacity is more i.e. more information can be stored in an image.

Disadvantages of LSB Technique

1. Robustness is low
2. Hidden data can be destroyed by simple attacks.

Transform Domain Technique

In the Frequency domain, the message is inserted into transformed coefficients of image giving more information hiding capacity and more robustness against attacks. Transform domain embedding can be termed as a domain of embedding techniques for which a number of algorithms have been suggested. Most of the strong steganographic systems today operate within the transform domain. Transform domain techniques have an advantage over LSB techniques as they hide information in areas of the image that are less exposed to compression, cropping, and image processing. Transform domain techniques are of different types.

1. Discrete Fourier transformation technique (DFT).
2. Discrete cosine transformation technique (DCT).

3. Discrete Wavelet transformation technique (DWT).

Discrete Cosine Transformation Technique (DCT)

DCT domain embedding techniques is the most popular one, mostly because of the fact that DCT based image format are widely available in public domain as well as the common output format of digital camera. JPEG image format for color components a discrete cosine transform (DCT) to transform successive $8 * 8$ pixel block of the image into 64 DCT coefficients each. The DCT coefficients $F(u, v)$ of an $8*8$ block of pixel $f(x, y)$ are given by

Where u = horizontal spatial frequency, v =vertical spatial frequency.

$C(x) = 1/\sqrt{2}$ when $x = 0$ and $C(x) = 1$ otherwise.

Embedding in DCT domain is simply done by altering the DCT coefficients. For example by changing the least significant of each coefficient. The modification of a single DCT coefficient affects all image pixels.

Discrete Wavelet Transform (DWT)

It is used to transform the image from its spatial domain into its frequency domain. We use DWT in the process of steganography so that we can clearly identify the high frequency and low frequency information of each pixel of the image. To obtain the DWT of the cover image, a filter pair called the Analysis Filter pair is used. First, the low pass filter is applied to each row of data in order to get the low frequency components of the row. Since the LPF is a half band filter, the Output data needs to be sub-sampled by two, so that the output Data now contains only half the original number of samples. Next, the high pass filter is applied for the same row of data, and similarly the high pass components are separated, and placed by the side of the low pass components. This procedure is done for all rows. Again filtering is done for each column of the intermediate data. The resulting two-dimensional array of coefficients contains four bands of data, each labelled as LL (Low-Low), HL (High-Low), LH (Low High) and HH (High-High). The LL band can be decomposed once again in the same manner, thereby producing even more sub-bands.

Hash-Least Significant Bits (Hash-LSB)

The Hash based Least Significant Bit (H-LSB) technique for steganography in which position of LSB for hiding the secret data is determined using hash function. Hash function finds the positions of least significant bit of each RGB pixel's and then message bits are embedded into these RGB pixel's independently. Then hash function returns hash values according to the least significant bits present in RGB pixel values. The cover image will be broken down or

fragmented into RGB format. Then the Hash LSB technique will use the values given by hash function to embed or conceal the data. In this technique the secret message is converted into binary form as binary bits; each 8 bits at a time are embedded in least significant bits of RGB pixel values of cover image in the order of 3, 3, and 2 respectively. According to this method 3 bits are embedded in red pixel LSB, 3 bits are embedded in green pixel LSB and 2 bits are embedded in blue pixel LSB. These 8 bits are inserted in this order because the chromatic influence of blue color to the human eye is more than red and green colors. Therefore the distribution pattern chooses the 2 bits to be hidden in blue pixel. Thus the quality of the image will be not sacrificed.

Masking and Filtering

Masking and Filtering is a steganography technique which can be used on grayscale images. Masking and filtering is similar to placing watermarks on a printed image. These techniques embed the information in the more significant areas than just hiding it into the noise level. Watermarking techniques can be applied without the fear of image destruction due to compression as they are more integrated into the image.

Advantages of Masking and Filtering Techniques

This method is much more robust than LSB replacement with respect to compression.

Disadvantages:

Techniques can be applied only to gray scale images and restricted to 24 bits.

Image Steganalysis

Steganalysis is the breaking of steganography and is the science of detecting hidden information. The main objective of steganalysis is to break steganography and the detection of stego image. Almost all steganalysis algorithms depend on steganographic algorithms introducing statistical differences between cover and stego image.

Steganalysis are of three different types:

1. Visual attacks

It discovered the hidden information, which helps to separate the image into bit planes for further more analysis.

2. Statistical attacks

Statistical attacks may be passive or active. Passive attacks involves with identifying presence or absence of a secret message or embedding algorithm used. Active attacks is used to investigate embedded message length or hidden message location or secret key used in embedding. Structural attacks the format of the data files changes as the data to be hidden is embedded, identifying this characteristic structure changes can help us to find the presence of image/text file.

REQUIREMENTS AND ANALYSIS

Problem Definition

The project is designed to implement Image Steganography on encrypted messages, which is implemented in an android environment. The receiver will be able to decode the image and obtain the encrypted message. On entering the correct key which was also used in the encryption process, the receiver will obtain the decrypted message.

This problem can be seen as an amalgamation of the following segments:

1. Encryption for the message – the sender is prompted to enter the message to be transmitted and the key which will be used for encrypting the message using the modified Vigenère Cipher algorithm.

Encryption algorithm - The key can comprise of any of the 256 characters that are recognized by the ASCII standards. In case the length of the key is shorter than the length of the input string, it is appended with itself 'n' times until the key and the string are of equal lengths. In case the key contains excess characters after being appended, the excess characters are dropped from the key. For each character of the message, the character obtained from the modular 256 operation on the sum of the ASCII values of the original message and corresponding key characters is the corresponding character in the encrypted text.

Plain Text: DISPLAY MESSAGE

Key: 12%&

D	I	S	P	L	A	Y		M	E	S	S	A	G	E
1	2	%	&	1	2	%	&	1	2	%	&	1	2	%
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
U	{	x	V	}	s	~	F	~	W	x	y	r	y	j

Cipher Text: U{xv}s~F~wxyryj

2. Embedding the message – the sender is asked to select an image into which the desired message has to be embedded. Embedding of the message is done by using the LSB (Least Significant Bit) substitution method. Here, the first 5120 bytes of the image are ignored to prevent the image file from getting corrupted. Thereafter, each character of the encrypted message is converted to bytes. For each byte of the image after the first 5120 bytes, the LSB of the image byte is substituted with a bit from the character byte, starting with the LSB. When all the bits for one character have been substituted, we move to the next character in the message. 16 null characters will be appended at the end of the message to act as an end of message indicator that helps increase the efficiency of the extraction process.

Let us consider an example. For an input image as given below, let us consider that the image is represented by bytes as:



Bytes					
10110011	11010000	10111001	10011011	11010011	10111111
01101100	11110011	11010111	10101101	00111010	01111010

Let the first character of the encrypted message be 'i'. The byte representation is 11001100. The image bytes after applying LSB substitution will be:

Bytes					
1011001 0	1101000 0	1011100 1	1001101 1	1101001 0	1011111 0
0110110 1	1111001 1	11010111	10101101	00111010	01111010

The remaining bits of the image remain unchanged. The image thus obtained is stored in a default location of the device which is then easily transferred.

3. Extracting the message – the user uploads the stego image to extract the message. After skipping the first 5120 bytes, the LSB of each byte is read to obtain the character byte of the message, moving from the LSB to the MSB of the byte. This byte returns a character for the encrypted message. The process is repeated until the sequence of 16 null character is encountered which acts as a terminator for the extraction process. The encrypted message is now given to the user.

Let us consider the above example.

Bytes					
10110010	11010000	10111001	10011011	11010010	10111110
01101101	11110011	11010111	10101101	00111010	01111010

When the byte 11001100 is obtained, the corresponding character value is stored as the next encrypted message character. On encountering the <> character, the message 'i' is returned.

4. Decryption of the message – the user is required to enter the key that was used in the encryption process. This acts a check to ensure that only an authorized user has access to the message. The key is be appended as in the encryption process. The modular 256 operation on the difference between the ASCII values of the encrypted character and the key gives the corresponding character of the original message.

u	{	x	V	}	s	~	F	~	W	x	y	r	y	j
1	2	%	&	1	2	%	&	1	2	%	&	1	2	%
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
D	I	S	P	L	A	Y		M	E	S	S	A	G	E

Requirement Specification

A software is developed when a need for the same arises. In a technologically advanced society where hacking, online forgery, message snooping have become commonly reported methods where a person's privacy has been invaded. Therefore, in such situations a person may want to transfer a message to a second person where it is not visible to any other person. This software has been developed keeping the above requirements under consideration. The software has been designed to implement the following purposes:

- The user should be able to enter the text and a desired key for encrypting the message. The key should not be pre-defined. This method of encryption will significantly reduce the chances of an unauthorized perpetrator to be able to crack the key as it can be repeatedly altered every time it is used. Thus this helps increase the dynamism of this method appreciably.

- Security mechanisms should be available so as to ensure that the key that is entered is not visible to a third, unauthorized user. The entire legitimacy of this method falls apart if an unauthorized user is able to crack the key. Hence one of the most important features have to be the strong safeguards which must be kept in place in order to ensure that no unauthorized user can gain access to the key.
- Separate portals to implement the encoding and decoding phase. Both processes may not take place sequentially on the same device. This is very important to ensure the viability and dynamism of this method. The method loses some of its usefulness if the message encoded by the sender in one device cannot be decoded by the receiver in a different device if the receiver has been intimated the cipher key by the sender.
- The method to ensure that the extracted, original message is only available to the authorized user. Thus, security levels should be included in a manner that in an event where the unauthorized user does gain access to the embedded image, he cannot view the original message. He only gets access to the encrypted message.

Planning and Scheduling

Planning and Scheduling of the processes that are involved in the development of a software system is of utmost importance. It is necessary to find an approximate plan which will be followed to develop the software. This would ensure efficient use of all resources including manpower, finance, system resources. Also it ensures that the software is developed in time and not after the requirement of the software has changed significantly or may have even become obsolete.

The planning and scheduling of our project helped us segregate the project into various segments. The gradual completion of each segment has in achieving our goal like every step that helps us move up the ladder. Our project development phase comprised of the following phases:

1. **REQUIREMENT SPECIFICATION:** Once the basic concept of our project was decided upon, it became imperative that we clearly identify what our software will do. This not only includes identifying and framing the overall problem but also identifying each sub-problem of our project which would ultimately be of vital importance. This phase recognized the important tasks of our project as :
 - i) Devising an algorithm to implement secured encryption of messages
 - ii) Transferring the key to the destination user to allow the decryption of the embedded message

- iii) The selection of an algorithm to allow image steganography on a wide range of messages
 - iv) The development of an Android mobile application to increase the scope of availability and use of the software being developed.
2. **SRS DOCUMENTATION:** It is not only important that we identify the requirements of a software but also ensure that the integrity of the software requirements is maintained. For this purpose, a formal document was prepared and submitted that clearly stated each requirement that the software has to fulfill. This phase helped us eliminate obsolete requirements. Transferring the key through the project would compromise the levels of security provided to a user in case of an attack by an unauthorized user. Thus, this requirement was considered as obsolete and removed during this phase.
3. **STRUCTURE DESIGN OF THE SOFTWARE:** The structure design of the software helped us get a better idea about how the requirements should be tackled. By creating a very basic structure of what we would want the final software to look like, we were given a better idea about what the scope of the code would be required to be. It also helped us identify intricate details such as hiding the key during input, displaying a message to indicate the completion of the steganography process and displaying the name of the output image files which would make the application easier for use by naïve users.
4. **LOGICAL DESIGN OF THE SOFTWARE:** This phase included the analysis of the various working algorithms available to achieve the requirements of our project. It also gave us which helped us modify the existing algorithms and devise our own to achieve the desired tasks. At the end of this phase the following algorithms were identified to be cordial for our system :
- i) A modified Vigenère Cipher algorithm which allows the user to encrypt messages using a polyalphabetic substitution technique on the input message.
 - ii) The Least Significant Bit substitution technique for image steganography. This method allows the user to embed the desired message into an image with minimal or no distortion.
 - iii) Corresponding algorithms that would help in extracting the embedded message from an image file and the decryption of the cipher text.

5. **DEVELOPMENT OF CODE:** This was a rather simple phase where the actual code was developed for the system. Each algorithm was treated as a different function and developed independently in the Java Eclipse environment.
6. **UNIT TESTING:** Each independently coded function was tested against a large set of test values. In case of discrepancies, the code was modified so as to ensure that the scope of use of the function was widened for the users. This phase made the debugging of the code easier as it is always easier to debug a small code segment instead of a complete integrated program code.
7. **CODE OPTIMISATION AND DOCUMENTATION:** After ensuring that each function achieves the purpose it has been designed for, the code was optimized to increase the efficiency of the code, optimize the resource requirements, including system resources and execution time. Also the length of the code was optimized by combining or removing redundant lines of code and redundant use of variables in the code. In order to enhance the readability of the code, comment lines were added to clearly explain the purpose of each function and also various code segments within a function. In this manner, the importance of the code segments are illustrated within the code itself, thereby making the code self-sufficient. A separate documentation would not be required to be included with the code only to explain the functionality.
8. **APPLICATION DEVELOPMENT:** In this phase, the different code modules that were developed in Java Eclipse and integrated together to develop the Android application. This phase was executed in the IntelliJ IDEA application which has been designed to help programmers develop Android applications. The codes written in Java were suitably modified so that they could run in the Android environment while ensuring that the basic algorithm that has been used to design each function was not lost. This phase integrated the use of different applications and storage spaces like Gallery, the Internal Storage, and File Commander that would be needed to run the application on a device. This phase produced a working prototype of the application which was running in Android devices.
9. **SYSTEM TESTING:** In this phase, the application was tested against a wide range of input test values. These input values were selected to judge if the application resulted in the distortion of images after steganography, if the application ran for all image formats in any size, if the length of the message affected the functioning of the application, does the application fail to respond when certain image sizes or message lengths were selected and was the security levels included to protect the cipher key values functioning as desired in the application. The application was progressively modified until us as the developers of the software were satisfied with the end product. Also, it became a method of reviewing the system to ensure all requirements, design specifications were included in the software to the best of our capabilities.

10. **PEER REVIEW:** After having developed and tested the application, we approached a group of our peers to help us in improving the software. We ran the application on their devices to see the limitations on the hardware and software requirements of our application. This approach allowed us to get feedback about the usability of the software and the ease with which a person not involved in the development process can operate the software.
11. **FORMAL DOCUMENTATION:** The formal documentation was an ongoing process that continued simultaneously during the latter phase of the project. The documentation included a complete overview and also the intricate description of the tasks involved in the project development, a description on the problem of the application for which this software has been developed, a short overview about the project topic itself, algorithms used to design the application, an extensive description about each algorithm, achievements, scope and limitations of the project, the intended future scope of implementation of the project, the scope of future improvements and upgrades for our project.

Figure 1: The Gantt Chart

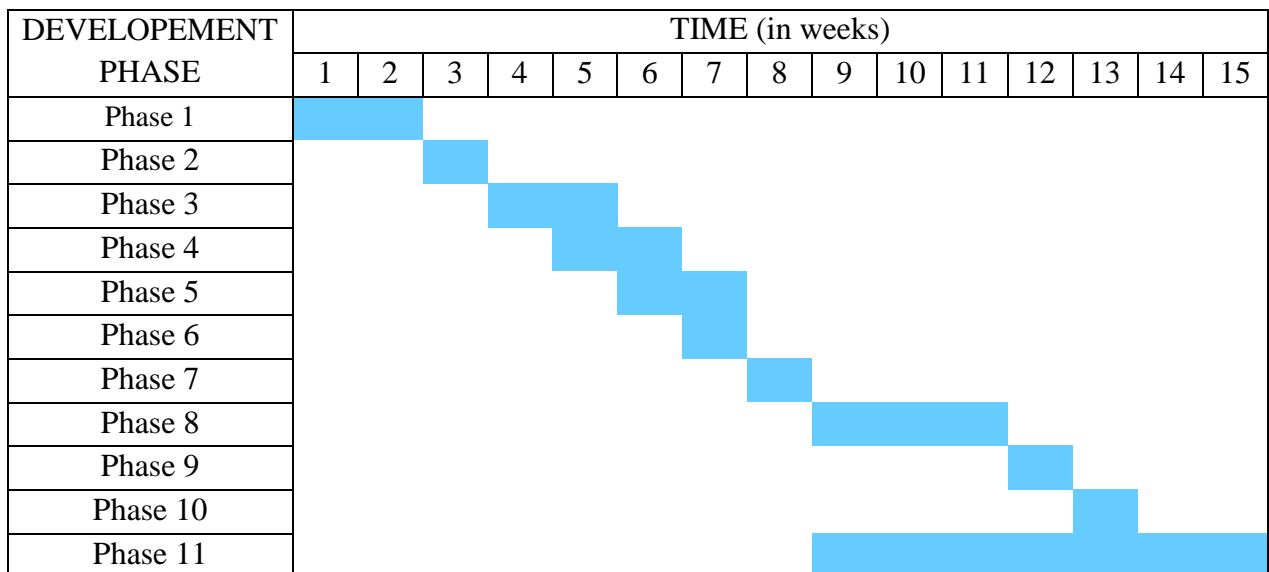
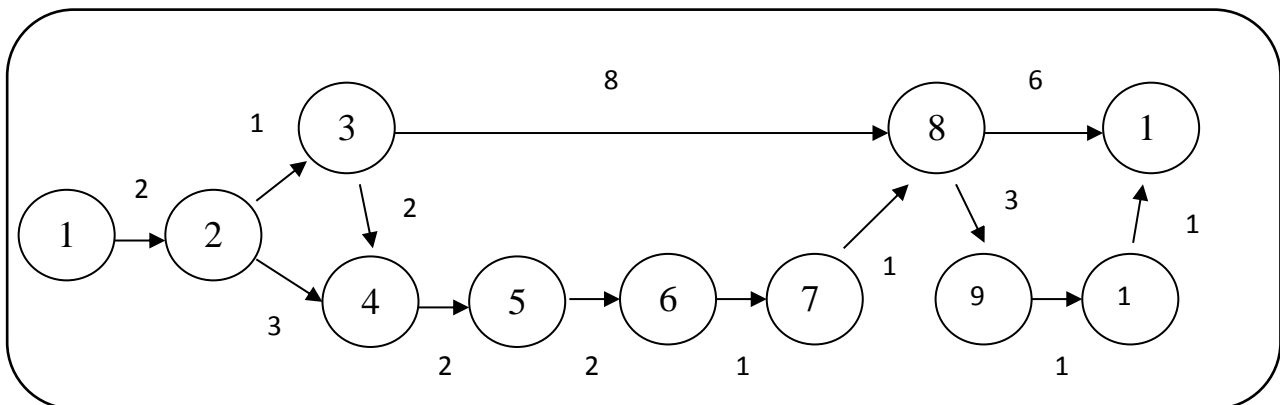


Figure 2: The PERT Chart



Software and Hardware Requirements

Software Requirements (for Development)

Java Development Kit (JDK)	Version 8.0
IntelliJ Idea Jetbrains	Android Development Software
Eclipse	Neon.3

Hardware Requirements (for Development)

Processor	Pentium Core2 Duo 2.6 GHz Processor Or Above
Random Access Memory (RAM)	3 GB (Minimum), 6 GB (Recommended)
Disk Space	500 MB for IntelliJ Idea Jetbrains 1.5 GB (Minimum) for Android SDK
Screen Resolution	1280 X 800 (Minimum)

Software Requirements (for Use)

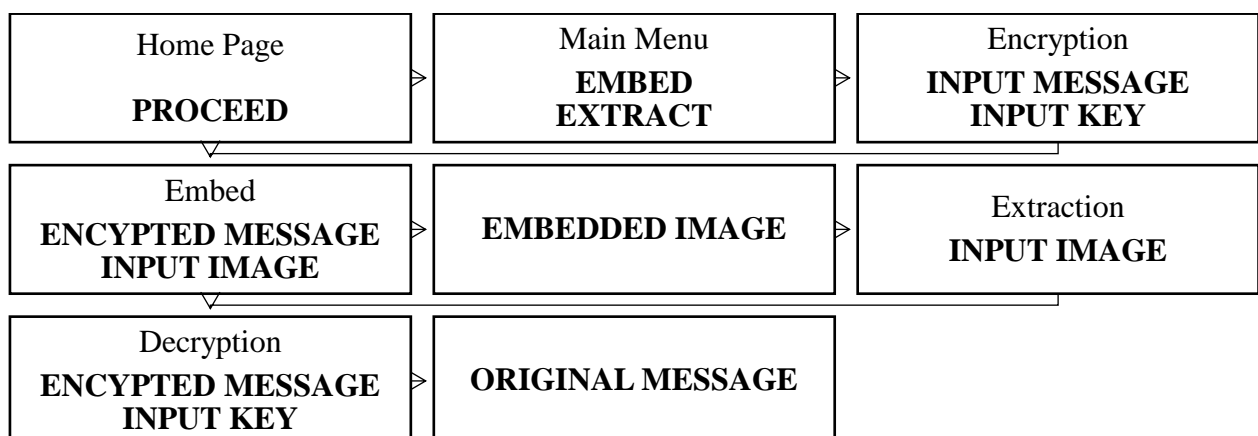
Android Operating System	Version 4.4 KitKat (Minimum)
--------------------------	------------------------------

Preliminary Product Description

The android application which has been developed gives the user options on its Main Menu to proceed to either; the Encryption and Embedding of Text in an Image, or the Extraction and Decryption of the Text from an Image. The user is prompted to select his or her preferred option and thus the system takes you to the chosen page. On the Encryption and Embedding of Text page, the user is first prompted to enter the message to be encrypted in a text field. Next, he or she has to input the password for encryption (formally known as the encryption key) and finally the user is allowed to select any image present in the smartphone or tablet's internal or external storage. Once these fields are completed, the user may choose to embed the text in the image. The system takes a while to complete the process after which it generates a message saying that the image has been successfully encoded.

The other option on the Main Menu will take the user to another page where the user is prompted to input the image to decrypt. Once the image is entered in the specified image box, the user may click on the 'Extract the Message' button, which extracts the embedded information from the image. This information although has not yet been decrypted for the user to read its original contents. The user is then prompted to input the password for encryption. If this password matches the password entered during the process of encryption, then the system decrypts the extracted encrypted message to reveal its original contents.

Conceptual Models



SYSTEM DESIGN

This chapter will encapsulate the physical design of the application developed in this project. One will be informed about how the requirements which were specified in the previous chapter has been met by the system. In short, the requirement of the application was to create an android application to implement image steganography, and to be able to do that on an android device (smartphones and tablets specifically).

For the implementation of the image steganography through the Least Significant Bit (LSB) method, we first need to understand what the LSB method really is. This method has its origins dating back to the seventeenth century when publishers of logarithmic tables used to introduce errors intentionally in the least significant digits. In the LSB insertion method, the binary representation of eight hidden information bytes is used to overwrite the least significant bits in the cover image. In steganography, as mentioned before, the ultimate goal is to ensure that the message is **invisible** to the third party who is trying to intercept it. The LSB Method exploits this idea. In this method, the data/information that is hidden is inserted into the least significant bits of the pixel information. According to an article on Academia, “an increase or decrease of value by changing the least significant bit does not change the appearance of the image, such that the resulted stego-image looks exactly same as the cover image”.

When using the LSB Method for the insertion of a message, the message is nearly undetectable to the naked eye. There are various LSB schemes, but most achieve low perceptibility, high payloads, and removable resistance. These terms mean users are most interested in how invisible the text is and how the cover image has been manipulated, or how large the capacity of the encoding method’s data hiding scheme is and the amount of secret data that can be hidden in the image.

From the following simple example we will try to understand exactly how the method of LSB substitution has been used in the steganography process.

Image Data (Say): **ABCDEFGH**

Message Data (Say): **a**

Hence, in bits the message data could be translated into the following bit array:

0	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

ASCII Code of ‘a’: 97

Now, to begin the LSB substitution method, we will replace each of the least significant bits of the image data bytes with a bit from the message data, starting from the last position to the first position (from LSB to MSB of the message data).

The following legend will be followed in the demonstration stated below for the benefit of the reader:

Message Bit Concerned
 Old Image Bit
 New Image Bit

Image Data Byte: 'A'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	0	0	1
After	0	1	0	0	0	0	0	1

ASCII Code of 'A': 65

Image Data Byte: 'B'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	0	1	0
After	0	1	0	0	0	0	1	0

ASCII Code of 'B': 66

Image Data Byte: 'C'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	0	1	1
After	0	1	0	0	0	0	1	0

ASCII Code of 'C': 67

Image Data Byte: 'D'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	1	0	0
After	0	1	0	0	0	1	0	0

ASCII Code of 'D': 68

Image Data Byte: 'E'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	1	0	1
After	0	1	0	0	0	1	0	0

ASCII Code of 'E': 69

Image Data Byte: 'F'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	1	1	0
After	0	1	0	0	0	1	1	1

ASCII Code of 'F': 70

Image Data Byte: 'G'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	0	1	1	1
After	0	1	0	0	0	1	1	1

ASCII Code of 'G': 71

Image Data Byte: 'H'

Message	0	1	1	0	0	0	0	1
Before	0	1	0	0	1	0	0	0
After	0	1	0	0	1	0	0	0

ASCII Code of 'H': 72

Therefore, due to the LSB substitution method, we witness the following:

Image Data (Before LSB Substitution): **ABCDEFGH**

Message Data: **a**

Image Data (After LSB Substitution): **ABDDGGH**

Basic Modules

Now that we have understood and also implemented basic LSB substitution method, we will now proceed towards the basic modules that this project comprises. They can be categorically classified as:

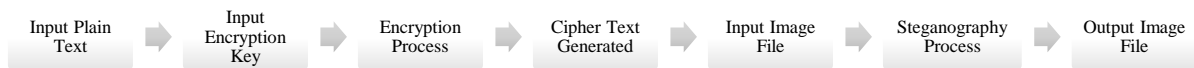
1. Encryption of the Plain Text: This module is basically a code segment which takes as input from the user a message of the user's choice in text format, thus including a full character set with both the ASCII character set [0-127] and the Extended ASCII character set [128-255] and also an encryption key. This text is now encrypted by using the Vigenère Cipher method.
2. Embedding of the Cipher Text in the Image: This module consists of the part where the steganography process is carried out, by using the LSB Substitution method to encode the cipher text received from the aforementioned function into the image file chosen by the user.
3. Extraction of the Cipher Text from the Image: This module consists of the part where the reverse steganography process is carried out, by using the reverse LSB Substitution method to uncover and retrieve the cipher text stored in the image from the previous function.
4. Decryption of the Cipher Text: This module is a function which takes as parameters the cipher text received by running the previous method and also a user input of the

decryption (same as encryption) key. This cipher text is now decrypted by using the reverse Vigenère Cipher method to attain the original plain text.

Data Design

In this project, data flow can occur in two ways, one from the end of encryption and the other from the end of decryption:

For the Encryption and Embedding Process:



For the Extraction and Decryption Process:

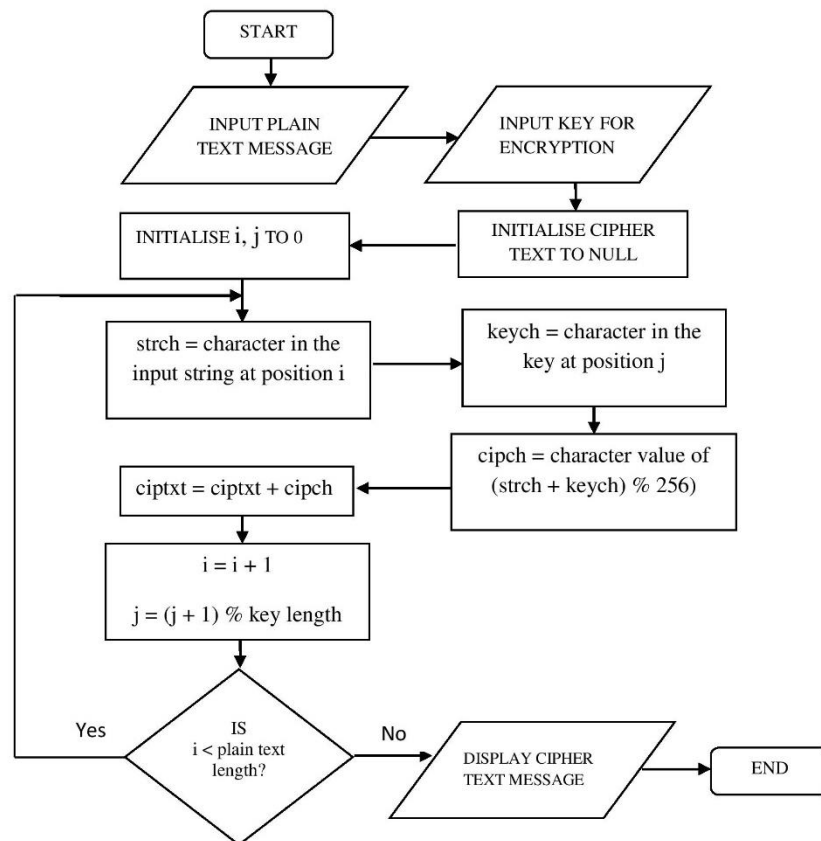


Procedural Design

This section of the chapter will show how each of the processes in our project have been carried out and what each of their functions pertain to:

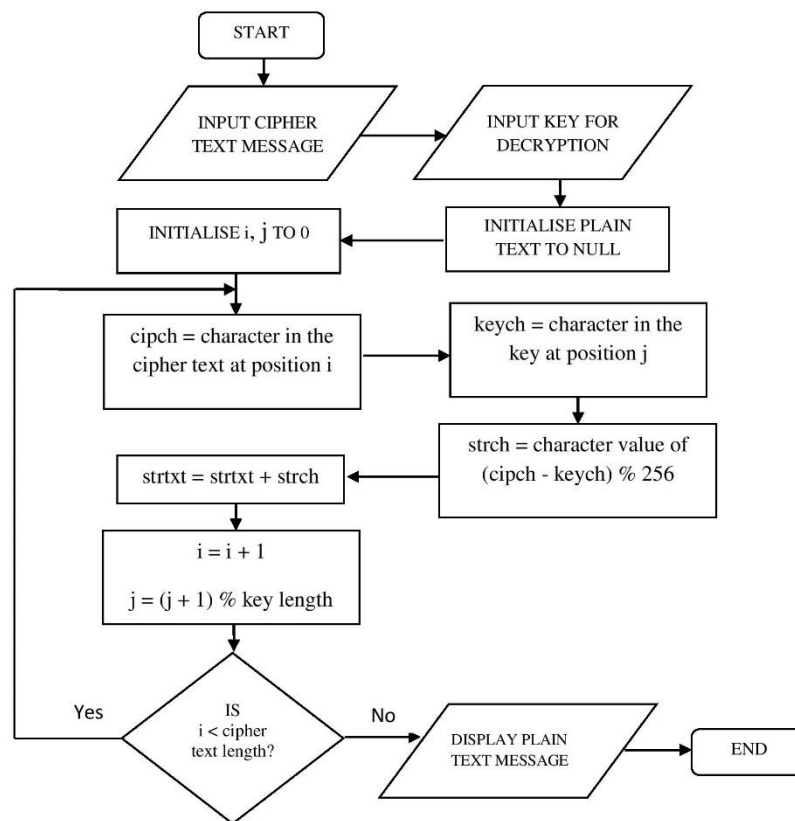
1. Encryption of the Plain Text (as entered by the user) to a Cipher Text, using an encryption key (also entered by the user): Flowchart 1

Flowchart 1: Encryption



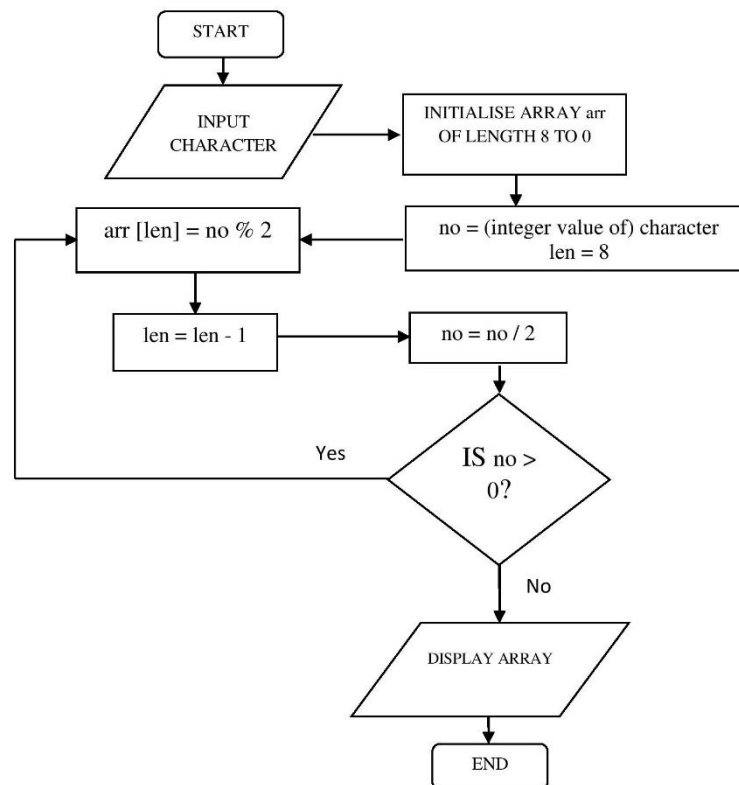
2. Decryption of the Cipher Text (as recovered from the Image) to the Plain Text using the decryption (same as encryption) key (as entered by the user): Flowchart 2

Flowchart 2: Decryption



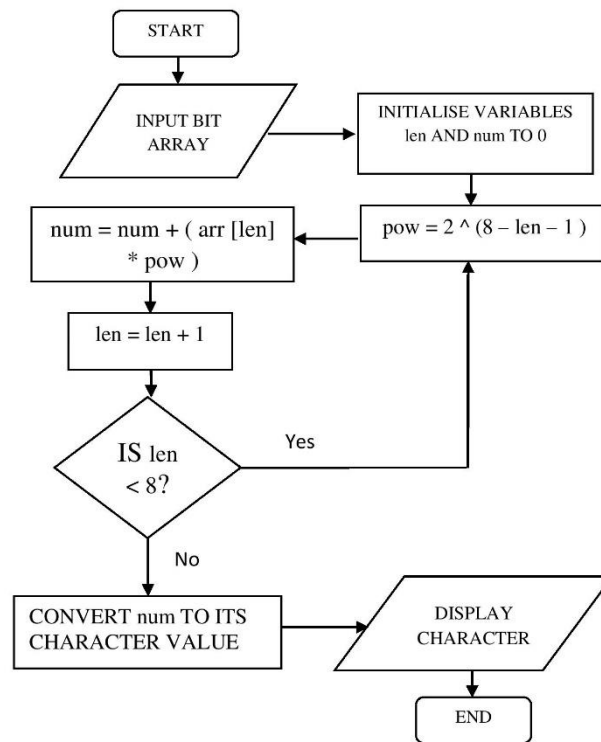
3. Conversion of a Character to an Integer type array of length 8, with the bits describing the binary value of the ASCII code of the input character: Flowchart 3

Flowchart 3: Character to Bits (Array)



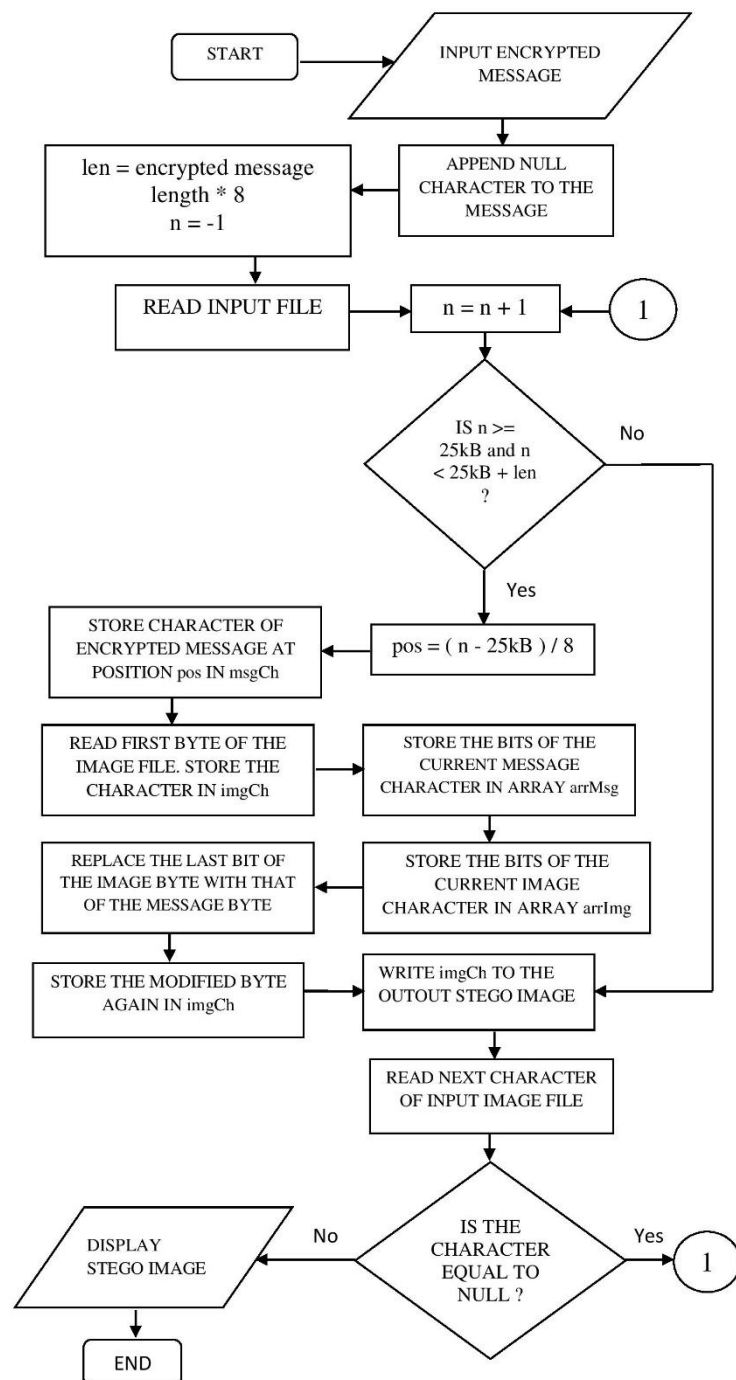
4. Conversion of an Integer type array of length 8, composing binary bits to a character of the same ASCII code built from the binary bits: Flowchart 4

Flowchart 4: Bits (Array) to Character



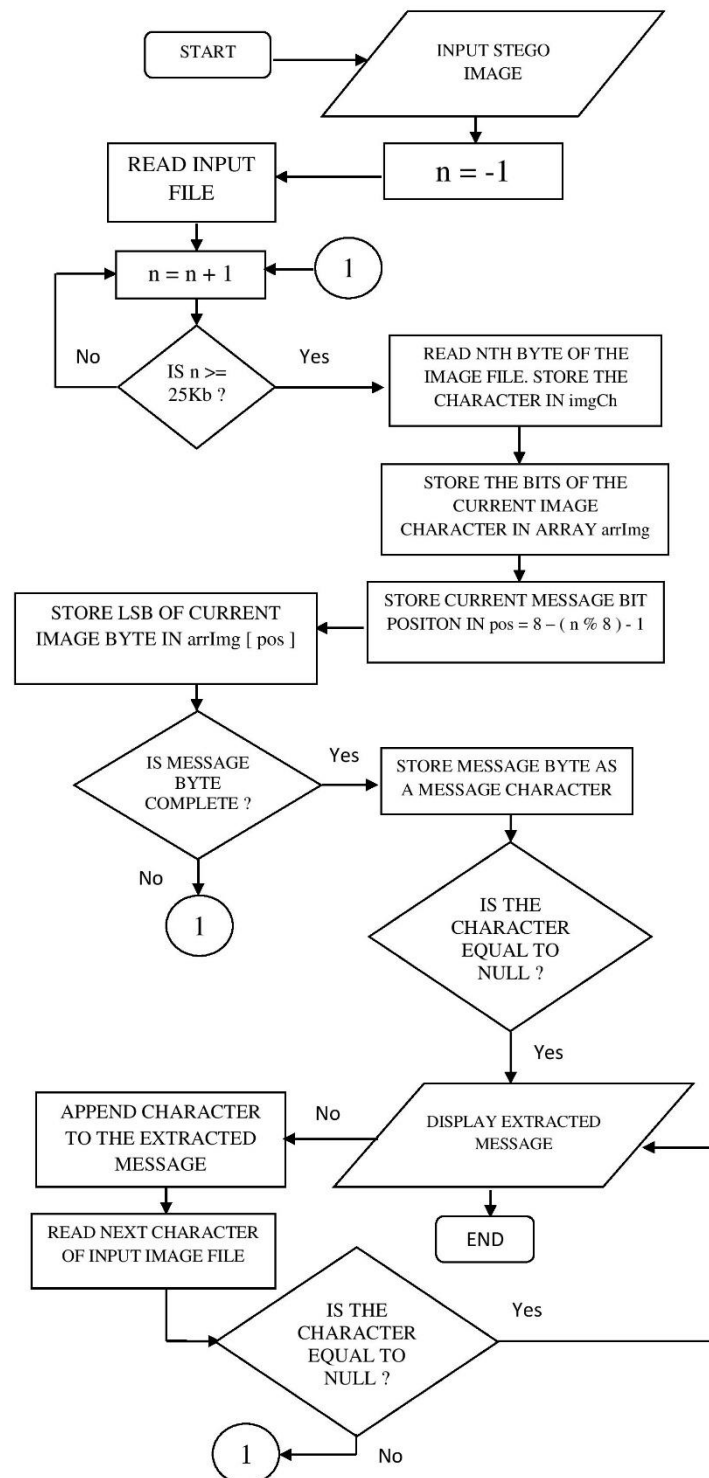
5. Embedding of the Cipher Text generated from Flowchart 1 into an image file (entered by the user). This function uses the Flowcharts 3 and 4 for execution: Flowchart 5

Flowchart 5: Embedding of Text in Image



6. Extraction of the Cipher Text from the image generated in Flowchart 5. This function also uses Flowcharts 3 and 4 for execution. The output of the function is sent to Flowchart 2: Flowchart 6

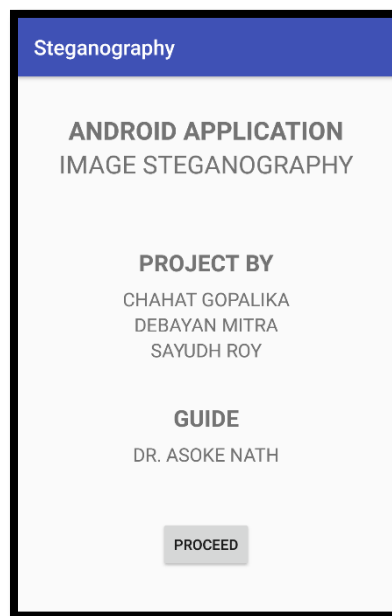
Flowchart 6: Extraction of Text from Image



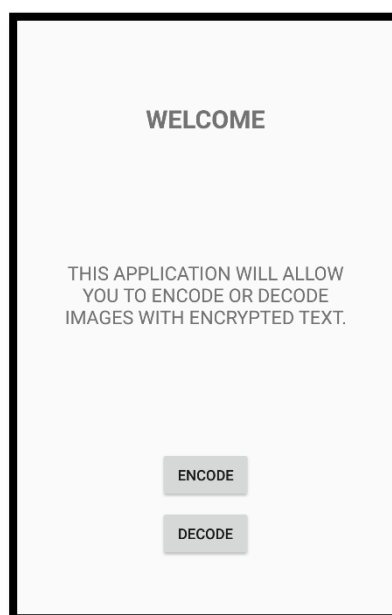
User-Interface Design

The following are rough pictorial views of the Android Application we have developed to implement Steganography on Android devices. This is the first draft of the interface that we have designed. However, we are working on the interface continuously and thus hope to present an upgraded interface on the day of the final presentation.

Home Page



Main Menu



Text Encryption & Embedding

IMAGE ENCRYPTION

Enter The Plain Text To Be Encrypted

Enter The Encryption Key

Enter The Image To Be Encrypted

ENCRYPT

IMAGE ENCRYPTION


Enter The Plain Text To Be Encrypted

Our Project Guide is Dr. Asoke Nath.

Enter The Encryption Key

....|

Enter The Image To Be Encrypted



ENCRYPT

IMAGE ENCRYPTION

Enter The Plain Text To Be Encrypted

Our Project Guide is Dr. Asoke Nath.

Enter The Encryption Key

....

Enter The Image To Be Encrypted




Image Has Been Encoded Successfully


Filename: Output.jpg

Text Extraction

IMAGE DECRYPTION
Enter The Image To Be Decrypted

EXTRACT MESSAGE

IMAGE DECRYPTION
Enter The Image To Be Decrypted



EXTRACT MESSAGE

Text Decryption

The Extracted Message

•aa•BaEEc•aUÄE•ÜÖ•±â•@æDlÖ•~ÄäÜ•

Enter The Encryption Key

DECRYPT

The Decrypted Text From The Image

1 2 3 4 5 6 7 8 9 0

q w e r t y u i o p

a s d f g h j k l

↑ z x c v b n m ↵

? ! @ , . : ; ' " & * % ^ & _ { } | ~ `

The Encrypted Message Has Been
Extracted Successfully

The Extracted Message

•aa•BaEEc•aUÄE•ÜÖ•±â•@æDlÖ•~ÄäÜ•

Enter The Encryption Key

• • • •

DECRYPT

The Decrypted Text From The Image

Our Project Guide is Dr. Asoke Nath.

1 2 3 4 5 6 7 8 9 0

q w e r t y u i o p

a s d f g h j k l

↑ z x c v b n m ↵

? ! @ , . : ; ' " & * % ^ & _ { } | ~ `

IMPLEMENTATION AND TESTING

Program Listing

Masking: To Break a Character into Bits

```
private static int[] toBits(char ch) {
    int arr[] = { 0, 0, 0, 0, 0, 0, 0, 0 }, no = (int) ch, len
= 8;
    while (no > 0) {
        arr[--len] = no % 2;
        no /= 2;
    }
    return arr;
}
```

Sample Input / Output

Example 1:

Parameter - ch = 'A'

Returned Integer Array - arr[] = [0, 1, 0, 0, 0, 0, 0, 1]

Example 2:

Parameter - ch = 'z'

Returned Integer Array - arr[] = [0, 1, 1, 1, 1, 0, 1, 0]

Unmasking: To Get Back a Character from its Bits

```
private static char toCharacter(int arr[]) {
    int len, num = 0;
    for (len = 0; len < 8; len++)
        num += arr[len] * Math.pow(2, 8 - len - 1);
    return (char) num;
}
```

Sample Input / Output

Example 1:

Parameter - Integer Array - arr[] = [0, 0, 1, 1, 0, 0, 1, 0]

Returned Character - ch = '2'

Example 2:

Parameter - Integer Array - arr[] = [0, 1, 1, 0, 0, 0, 1, 1]

Returned Character - ch = 'c'

Encryption: To Encrypt the Plain Text to Cipher Text

```
public String encrypt(String text, String key){
    String res = "";
    for (int i = 0, j = 0; i < text.length(); i++) {
        res += (char) ((text.charAt(i) + key.charAt(j)) %
256);
        j = ++j % key.length();
    }
    return res;
}
```

Sample Input / Output

Example 1:

Parameter - String text = "Debayan", String key = "Chahat"

Returned String res = "?ÍÃÉÚÕ±"

Example 2:

Parameter - String text = "Asoke Nath", String key = "Sir"

Returned String res = "?Üá¾Î?;Êæ»"

Decryption: To Decrypt the Cipher Text to Plain Text

```
public String decrypt() {
    String text=Data.extractedMsg, key=Data.key;
    String res = "";
    for (int i = 0, j = 0; i < text.length(); i++) {
        res += (char) ((text.charAt(i) - key.charAt(j)) %
256);
        j = ++j % key.length();
    }
    return res;
}
```

Sample Input / Output

Example 1:

String text = "?ÍÃÉÚÕ±", String key = "Chahat"

Returned String res = "Debayan"

Example 2:

Parameter - String text = "?Üá¾Î?;Êæ»", String key = "Sir"

Returned String res = "Asoke Nath"

Embed: To Embed the Cipher Text into the Image

```
public void embed(String msg, String path) {
    String dirName =
Environment.getExternalStorageDirectory().getPath()+
"/output.jpg";

    FileInputStream fin = null;
    FileOutputStream fout=null;
    try {

        File fp=new File(dirName);
        fp.createNewFile();

        fout= new FileOutputStream(fp);
        fin = new FileInputStream(path);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    int i, n = -1, arrImg[], arrMsg[], len;
    msg += '\0';
    len = msg.length() * 8;
    char imgCh, msgCh;
    try {

        while ((i = fin.read()) != -1) {
            n++;
            imgCh = ((char) i);
            if (n >= (25*1024) && n < (25*1024) + len) {
                msgCh = msg.charAt((n - (25*1024)) / 8);
                arrMsg = toBits(msgCh);
                arrImg = toBits(imgCh);
                arrImg[7] = n % 8 == 0 ? arrMsg[7] : arrMsg[8
- (n % 8) - 1];
                imgCh = toCharacter(arrImg);
            }
            fout.write(imgCh);
        }
        fin.close();
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Extract: To Extract the Cipher Text from the Image

```
public String extract(String path) throws IOException {
    FileInputStream fin = new FileInputStream(path);
    String msg = "";
    int i, n = -1, arrImg[], arrMsg[] = new int[8];
    char imgCh, msgCh;
    try {

        while ((i = fin.read()) != -1) {
            n++;
            if (n >= (25*1024)) {
                imgCh = ((char) i);
                arrImg = toBits(imgCh);
                arrMsg[8 - (n % 8) - 1] = arrImg[7];
                if ((n + 1) % 8 == 0) {
                    msgCh = toCharacter(arrMsg);
                    if ((int) msgCh == 0)
                        break;
                    msg += msgCh;
                }
            }
        }
        fin.close();
        System.out.print("\nMessage successfully extracted.");
    } catch (Exception e) {
        System.out.println(e);
    }
    return msg;
}
```

Android Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.steganography">
    <uses-permission
android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
android:name="android.permission.READ_INTERNAL_STORAGE"/>
    <uses-permission
android:name="android.permission.READ_INTERNAL_STORAGE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```

        android:supportsRtl="true"
        android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action
android:name="android.intent.action.MAIN"/>

            <category
android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <activity android:name=".Menu"></activity>
    <activity android:name=".Encode"></activity>
    <activity android:name=".Decode"></activity>
    <activity android:name=".Decrypt"></activity>
</application>

</manifest>

```

Integrated Testing

CASE 1

Input File: input1.jpg



Encryption

Enter the Secret Message: at the end, just one line.

Enter the Key: !@#\$\$%

The Encrypted Message: ?`C???`???M`????`???A-???O

Output File: outputJPEG1.jpg



The Extracted Message: ?'C???`???M`????`???A-???O
Enter the Key: !@\$%
Decrypted Message: at the end, just one line.

CASE 2

Input File: input2.jpg



Enter the Secret Message: this is a display message.
Enter the Key: qwerty
The Encrypted Message: âßÎâ?âä?Æ?ØâäçÑÓí?ÞÜØâÕàÖŸ

Output Image: outputJPEG2.jpg



The Extracted Message: âßÎâ?âä?Æ?ØâäçÑÓí?ÞÜØâÕàÖŸ
Enter the Key: qwerty
Decrypted Message: this is a display message.

CASE 3

Input File: input3.jpg



Enter the Secret Message: These are basic modules.

Enter the Key: hash

The Encrypted Message: ¼ÉØÚÍ?ÔÚÍ?ÕÉÛÊÖ?ÕÐ×ÝÔÆæ?

Output Image: outputJPEG3.jpg



The Extracted Message:

We see in the above example, the images which have a size less than 25kb will not support image steganography. In the algorithm we have skipped the first 25kb to prevent the header of any image file from getting altered. This will ensure that the image file does not get corrupted using the steganography embed process. Thus, in a scenario where an input file has a size less than 25 kb, the embed function does not accomplish its task even though no error will be raised by the application. This is because the complete image will be read under the first 25kb which serve to prevent any significant distortion of the image.

CASE 4

Input File: input4.jpg



Enter the Secret Message: a gray cat is walking on a gray wall.

Enter the Key: colour

The Encrypted Message: Ä?ÓáÖë?ÒÍã?ÛÖ?ãÐáÝÌÝÓ?ää?Ð?ÖçÓÛ?ãÐáÐ?

Output Image: outputJPEG4.jpg



The Extracted Message: Ä?ÓáÖë?Òíã?ÛÖ?ãĐáÝÌÝÓ?ää?Đ?ÖçÓÜ?ãĐáĐ?
 Enter the Key: colour
 Decrypted Message: a gray cat is walking on a gray wall.

CASE 5

Input File: input5.jpg



Enter the Secret Message: test message
 Enter the Key: 123
 The Encrypted Message: ¥?¥R ?¥!???

Output Image: outputJPEG.jpg



The Extracted Message: ¥?¥R ?¥!???
 Enter the Key: 123
 Decrypted Message: test message

CASE 6

Input File: input6.jpg



Enter the Secret Message: this is an invalid test.

Enter the Key: run

The Encrypted Message: æÝ×â?×â?Ïà?×àëÏpPÒ?éÓâé?

Output Image: outputJPEG6.jpg



The Extracted Message: æÝ×â?×â?Ïà?×àëÏpPÒ?éÓâé?

Enter the key: running

Decrypted message: thiw,i~#Zr'nr?]ipd){lst.

This is an invalid example. Here, we see that the key entered during decryption is different from the key used during encryption. In the cipher algorithm, the key entered by the user is used to generate the cipher text. The cipher text is therefore related only to the input message and the entered key. In the event where the user enters an invalid key, the decrypted message which is returned to the user is inaccurate. This is imperative as this algorithm provides the required security to the user. An unauthorized user is expected to not have the valid key needed to decrypted one message. This means that when the unauthorized user enters an erroneous key, the message returned to the user will be unintelligible to the third party.

CASE 7

Input File: input7.jpg



Enter the Secret Message: this is the testing phase

Enter the Key: check

The Encrypted Message: ×ĐÎÖ?ÌÛ?×ÓÈ?ÙÈ×ÑÓÊ?ÓÐÆÖÐ

Output File: outputJPEG7.jpg



The Extracted Message: ×ĐÎÖ?ÌÛ?×ÓÈ?ÙÈ×ÑÓÊ?ÓÐÆÖÐ

Enter the Key: check

Decrypted Message: this is the testing phase

CASE 8

Input File: input8.bmp



Enter the Secret Message: this one is bitmap.

Enter the Key: cats

The Encrypted Message: ×ÉÝæ?ĐâØ?Êç?ÂÊèàÃÑç

Output File (in BMP): output.bmp



Output File: outputJPEG8.jpg



The Extracted Message: ×ÉÝæ?ĐâØ?Êç?ÂÊèàÃÑç

Enter the Key: cats

Decrypted Message: this one is bitmap.

The following were the times observed for the processes:

Table 1: Observed Times for the Sample Inputs/Outputs

Sl. No.	Cover File		Secret Message	Time	
	Name	Size		Embed	Extract
1.	input1.jpg	127 KB	at the end, just one line.	3 sec	0.5 sec
2.	input2.jpg	187 KB	This is a display message.	5 sec	1 sec
3.	input3.jpg	9 KB	These are basic modules.	-	-
4.	input4.jpg	93 KB	a gray cat is walking on a gray wall.	7 sec	2 sec
5.	input5.jpg	193 KB	test message	5 sec	1 sec
6.	input6.jpg	3000 KB	this is an invalid test.	2 min 2sec	2 min 2 sec
7.	input7.jpg	157 KB	this is the testing phase	9 sec	0.5 sec
8.	input8.jpg	147 KB	this one is bitmap.	3 sec	0.5 sec

As we see in the above example, the images which have a size less than 25kb will not support image steganography. In the algorithm we have skipped the first 25kb to prevent the header of any image file from getting altered. This will ensure that the image file does not get corrupted using the embed process. Thus, in a scenario where the input file has a size less than 25 kb, the embed function does not accomplish its task even though no error will be raised by the application. This is because the complete image will be read under the first 25kb which serve to prevent any significant distortion of the image.

RESULTS AND DISCUSSIONS

The project on image steganography has been fairly successful and has been running smoothly in most cases. However, there are certain minor technical glitches that this project runs into in certain situations which throw up certain anomalies which have not been resolved yet. However as we go forward with this project, we intend to introduce certain modifications that will take care of the anomalies.

Our results indicate that the LSB insertion using random key is better than simple LSB insertion in case of lossless compression. The image resolution doesn't change much and is negligible when we embed the message into the image and the image is protected with the personal key as long as the message to be embedded is not too big. So, it is not possible to damage the data by unauthorized personnel. However we have observed that in the scenario where the length of the message embedded is very long as in comparison to the size of the image being used where the message is to be embedded.

Furthermore, to maintain the integrity of application, we have had to place a condition where we need to leave a space of 25 kilobytes (kb) in every image where a message needs to be embedded. This condition goes on to ensure that we can embed the message in the image without a problem. However keeping this condition has created a slight problem where we cannot use an image whose size is less than or equal to 25 kilobytes (kb), otherwise the encrypted code will not be embedded. So we need to always ensure that the images we use are greater than 25 kb, and realistically, it needs to be greater than at least 50 kb to ensure that the quality of the image does not deteriorate too much once the message is embedded. We shall try to ensure that we can devise a method where we can use images of all sizes where messages can be embedded.

The final drawback of this project we have noticed is that if we click on the extract image option or any other option without first selecting and inputting the image where the message will be embedded and the steganography will be implemented, then the application stops functioning and must be restarted. We are trying to overcome this glitch and ensure that the application does not crash if there is a scenario where we select one of the options without having had selected the message where the message is to be embedded.

The preceding paragraphs document the few modifications that we need to undertake to make this application more or less completely free of glitches. However there is one more improvement that we would like to incorporate in this project as we go forward with it so as to make the application more user friendly. As we mentioned in the project synopsis, we intend to keep working on this project and upgrade it where the embedded message is converted into a QR code instead of a jpeg file. This QR code is to be sent to the receiver who can now scan

the QR code to decrypt the sent message. This improvement also goes a long way in further ensuring that the protective measures present in this application is remarkably increased and the chances of data theft or interception is made more or less negligible.

CONCLUSION

Steganography is an effective way to hide sensitive information and the relevance of such information hiding techniques is perpetually on the rise in this modern digitized world where the threat of data theft and interception are part of everyday reality.

In this paper we seek to address this constant threat of data theft by making the information that we send across digital networks more secure and we seek to do that with the help of image steganography. As we know, there are different methods to implement the process of encryption via image steganography. In this project, we have used the Least Significant Bit (LSB) Technique on images to obtain secure stego-image. In this method, we take the least significant bit of the image and replace it with the least significant bit of the encrypted message, and this process progressively continues with higher bits.

We have also kept in mind the easy availability and widespread use of the Android platform due to which we have created an application based in Android which we can use to encrypt and decrypt messages via image steganography.

Limitations of the System

Our results indicate that the LSB insertion using random key is better than simple LSB insertion in case of lossless compression. The image resolution doesn't change much and is negligible when we embed the message into the image and the image is protected with the personal key as long as the message to be embedded is not too big. So, it is not possible to damage the data by unauthorized personnel. However we have observed some slight change in picture resolution in the case of comparatively sizeable embedded messages and this is a shortcoming we intend to look into and rectify in future projects.

Furthermore, to maintain the integrity of application, we have to leave a space of 25 kilobytes (kb) in every image. This will ensure that we can embed the message in the image without hassle. However this leads to a slight exception where we cannot encrypt a message in an image whose size is less than or equal to 25 kilobytes (kb), otherwise the encrypted code will not be embedded. So preferably, it is helpful if we take images which are at least more than or equal to 50kb in size.

Another drawback of this project is that if we click on the extract image option in the window without actually selecting the image where we want to embed the encrypted message, then the application fails.

Future Scope of the Project

We have mentioned in the preceding paragraphs some of the minor unresolved technical glitches that we ran into while embarking on this project. We intend to continue working on these errors and iron out these glitches in the future. Moreover, as we mentioned in the project synopsis, we intend to keep working on this project and upgrade it where the embedded message is converted into a QR code instead of a jpeg file. This QR code is to be sent to the receiver who can now scan the QR code to decrypt the sent message.

This project has been an enriching experience for all three of us where we learnt massively and we have understood a lot about how modern day encryption through steganography can be implemented to further protect and maintain the integrity of messages and data sent through the digital platform.

REFERENCES

- 1) New Data Hiding Algorithm in MATLAB using Encrypted secret message, Agniswar Dutta, Neeraj Khanna, Abhirup Kumar Sen and Joel James under the guidance of Dr. Asoke Nath, April 2011.
- 2) A Secure Image Steganography using LSB Technique and Pseudo Random encoding technique, Kshetrimayum Jenita Devi under Dr. Sanjay Kumar Jena (professor), Dept. of Computer Science and Engineering, National Institute of Technology, Rourkela, May 2013.
- 3) Survey on Different Method of Image Steganography by Palak R Patel, Department of Computer Engineering, Parul Institute of Engineering (PIET), Limda, India, Assistant Professor, Department of I.T., Parul Institute of Technology (PIET), Limda, India, 2014. Image Based Steganography Using LSB Method and Java Based Encryption, Deepika and Dr. Sanjay Kumar, Jaipur National University, Jaipur, International Journal of Engineering Trends and Applications (IJETA) – Volume 2 Issue 5, Sep-Oct 2015.
- 4) Research paper titled Data Structures : Image Steganography, by Nick Nabavian, Nov. 28, 2007.
- 5) www.stackoverflow.com – to find clarity on technical issues faced during the development of the application.
- 6) <http://thinktobits.blogspot.in/2012/12/ImageIO-Convert-Bitmap-to-JPEG-in-Java-Example.html> - reference to find clarity on to under how we can change the image formats in a Java program.
- 7) Advanced Steganographic approach hiding encrypted secret message in LSB, LSB+1, LSB+2 and LSB+3 bits in non-standard cover files, Joyshree Nath, Sankar Das, Shalabh Agarwal and Asoke Nath, International Journal of Computer Applications (0975-8887) Vol 14-No7, Feb 2011.
- 8) Data Hiding and Retrieval: Asoke Nath, Sankar Das, Amlan Chakraborti, published in IEEE “Proceedings of International Conference on Computational Intelligence and Communication Networks (CICN 2010)” held from 26-28 NOV’2010 at Bhupal, Page: 392-397(2010).

- 9) Advanced Steganography Algorithm using encrypted secret message: Joyshree Nath and Asoke Nath, International Journal of Advanced Computer Science and Applications, Vol-2, No-3, Page-19-24, March (2011).
- 10) Sankar Das, Asoke Nath, Arijit Samanta, Abhishek Roy, Saptarshi Bhattacharyya, “A Secure Approach for data Hiding using Visual Cryptography”, International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE), ISSN(Online):2320-9801,ISSN(Print):2320-9798, Vol.4,Issue 6, Page 10274-10282, June 30, 2016.

GLOSSARY

1) **Steganography**

The practice of concealing messages or information within other non-secret text or data.

2) **Polyalphabetic Substitution Cipher**

A substitution cipher technique where multiple substitution characters are used to generate the cipher text from the plain text.

3) **Plain Text**

The message which needs to be encrypted using an encryption algorithm to secure its access by unauthorized users.

4) **Cipher Text**

The message generated after applying an encryption algorithm on a plain text.

5) **Key**

The combination of characters entered by the user to encrypted an entered message.

6) **Cover Image**

The image file into which the text will be embedded. This is the image on which the image steganography algorithm will be implemented.

7) **Android**

An open-source operating system used for smartphones and tablet computers

8) **PERT Chart**

A project management tool, known as Program Evaluation Review Technique. This provides a graphical representation of a project's timeline that allows the tasks in a particular project to be analyzed.

9) **Gantt Chart**

A chart developed to describe the project development schedule.