ABSTRACT

Steganography is the art of hiding information within other information in such a way that it is hard or even impossible to identify the existence of any hidden information. There are many different carriers for steganography. Of which, most popular ones are digital images. Due to recent developments in steganalysis, providing security to personal contents, messages, or digital images using steganography has become difficult. By using steganalysis, one can easily reveal existence of hidden information in carrier files. This project introduces a novel steganographic approach for covert communications between two private parties. The approach introduced in this project makes use of both steganographic as well as cryptographic techniques. The process involves converting a secret image into a text document, then encrypting the generated text into a ciphertext using a key (password) based encryption algorithm, and finally embedding the ciphertext on to a cover image. This embedding process is carried out using a threshold based scheme that inserts secret message bits into the cover image only in selected pixels. The security to maintain secrecy of message is achieved by making it infeasible for a third person to detect and retrieve the hidden message.
# TABLE OF CONTENTS

Abstract ................................................................................................................................. ii

Table of Contents .................................................................................................................. iii

List of Figures ......................................................................................................................... v

1. Introduction ......................................................................................................................... 1

   1.1 Traditional Approach ..................................................................................................... 2

       1.1.1 Monoalphabetic cipher ......................................................................................... 3

       1.1.2 MD5 Hashing ......................................................................................................... 4

       1.1.3 Data Encryption Standard .................................................................................... 4

       1.1.4 Least Significant Bit Substitution ......................................................................... 6

       1.1.5 Jsteg ..................................................................................................................... 6

       1.1.6 Embedding Process .............................................................................................. 7

   1.2 History ............................................................................................................................ 7

       1.2.1 Cryptanalysis ....................................................................................................... 7

       1.2.2 Stegoanalysis ...................................................................................................... 8

   1.3 Objective ......................................................................................................................... 9

   1.4 Rationale ........................................................................................................................ 10

2. Narrative ............................................................................................................................. 11

   2.1 Problem Statement ....................................................................................................... 11

   2.2 Motivation ...................................................................................................................... 11

   2.3 Scope ............................................................................................................................... 12

   2.4 Functionality ................................................................................................................ 12

3. Proposed System ................................................................................................................ 13
3.1 Implementation ...........................................................................................................................................13
  3.1.1 Base64 Conversion .................................................................................................................................13
  3.1.2 Password based encryption (PBE) ........................................................................................................16
  3.1.3 Embedding ...............................................................................................................................................16
  3.1.4 Embedding Process ...............................................................................................................................18
3.2 System Design ...............................................................................................................................................20
  3.2.1 Architecture ...........................................................................................................................................20
  3.2.2 Flow of Execution ..................................................................................................................................21
3.3 System and Resources .................................................................................................................................23
3.4 System Requirements ....................................................................................................................................23
3.5 User Interface Design ...................................................................................................................................24
  3.5.1 Main Window ..........................................................................................................................................24
  3.5.2 Encode Tab ............................................................................................................................................24
  3.5.3 Decode Tab ............................................................................................................................................24
3.6 Software Engineering Techniques .............................................................................................................25
  3.6.1 Requirements .........................................................................................................................................25
    3.6.1.1 Software Tools .................................................................................................................................25
    3.6.1.2 Functional and Non-Functional Requirements .............................................................................25
    3.6.1.3 Structured Requirements ...............................................................................................................25
3.7 Analysis .........................................................................................................................................................26
3.8 Design ..........................................................................................................................................................28
4. Functionality of Application ..........................................................................................................................29
  4.1. Modules Description ................................................................................................................................32
5. Evaluation .........................................................................................................................39
  5.1. First Experiment ........................................................................................................39
  5.2. Second Experiment .....................................................................................................40
6. Conclusion ..........................................................................................................................41
7. Future Work .......................................................................................................................42
8. References ..........................................................................................................................43
LIST OF FIGURES

Figure 1: Encryption circuit block diagram .................................................................5
Figure 2: Inverse circuit of encryption circuit ...............................................................5
Figure 3: Chart for base 64 encoding ...........................................................................14
Figure 4: System architecture ....................................................................................20
Figure 5: Flow of execution of sender’s operations ......................................................21
Figure 6: Flow of execution of receiver’s operations .....................................................22
Figure 7: Use case diagram of the system ..................................................................29
Figure 8: Sequence diagram of image to text conversion ..........................................30
Figure 9: Sequence diagram for embedding process ...............................................30
Figure 10: System component diagram .......................................................................31
Figure 11: Main window ..............................................................................................33
Figure 12: Operations available for a sender ...............................................................34
Figure 13: Operations available for a receiver .............................................................35
Figure 14: Available user operations in auto mode ......................................................36
Figure 15: Encoding window in auto mode .................................................................37
Figure 16: Decoding window in auto mode .................................................................38
Figure 17: Table 1 ........................................................................................................39
Figure 18: Table 2 ........................................................................................................41
Appendices ..................................................................................................................46
    Appendix A: Code ..................................................................................................46
Introduction

Lately, exponential growth of technology in every aspect of life is observed. Improvement of technology provides facilities to both users and hackers/intruders too. Advancement in technology that encourages hackers/intruders activities result in lack of security to user’s confidential data. The most common and popular techniques for data hiding that have been in use since long time are cryptography and steganography.

Cryptography:

There are many possible definitions for cryptography. One of which is, “The computerized encoding and decoding of information” to define cryptography [10]. This is a process of converting a message from a human readable or understandable form (plaintext) to non-understandable format (ciphertext) to enable secure sending and back to original format at other receiving end. The ciphertext in cryptography always reveals static information of plaintext. Many methodologies were introduced that follow their own strategy, but all the methodologies use some patterns. The underlying idea in pattern based approach is to decode the encoded message, that is, using a pattern of one’s own choice or a standard pattern, a sender encodes the message and thus generates a ciphertext. The receiver uses the same pattern and decodes the ciphertext to generate message (plaintext). Over a period, cryptographic approaches evolved over phases. It is suggested that a key should be used in the process of encoding and decoding a message. Based on this concept of keys, cryptography is further classified into two types, symmetric-key cryptography and public-key cryptography [10]. In case of symmetric key cryptography, same key has to be used by both sender and the receiver while encoding and decoding respectively. In contrast, in the case of public key cryptography, the keys used by the sender and the receiver are
different.

Steganography:

It can be defined as "The art and science of communicating in a way which hides the existence of the communication" [10] [7]. A steganographic model facilitates hiding or embedding of sender’s secret message in a file (carrier) that does not give out a clue about the existence of secret message in it when viewed. For this, any media format or file format like .bmp, .doc, .gif, .jpeg, .mp3, .ppt, .txt and .wav is taken as a carrier that can act as cover for the sender’s message, that is, a message here is hidden in a carrier and that carrier is transmitted. The underlying operation of this methodology is both logical and technical. In general, a steganography algorithm takes a secret message and a carrier as input and gives a carrier message as output (in which the message is embedded). In the process of steganography, the carrier which hides the message in it will be sent to the receiver [7]. The carrier gives the receiver no information about the message but reveals it only after using the tool or algorithm that is used by the sender.

Both cryptography and steganography have found usage in many applications. For example, transmission of attack plans by military teams to hide information about their strategies. Many other applications of data hiding techniques other than its original objective, have gained importance, which include authentication and identification, watermarking and transmitting passwords etc. [10]

1.1 Traditional Approaches

In this section, we discuss several encoding techniques that are often used in day-to-day functions to ensure data security. Currently, there are many methods which could hide data. All the methods may be applied at any time irrespective of the content available.
Each of these methods when used to encode and decode, data has its own constraints that need to be considered. There are certain requirements that must be satisfied. These requirements can be: the format of the input file, the size of the input file and the encryption key.

In the following sections, both cryptographic and steganographic methods that provide data confidentiality are described.

1.1.1. Monoalphabetic Ciphers

*Caesar cipher* is the most popularly used substitution ciphering method. This was introduced by Suetonius in his biography [2] [3]. This involves a very simple substitution in which each alphabet will be replaced by third letter following it alphabetically. For example, if the plaintext has an alphabet ‘A’ it will be replaced with ‘D’ and the same method is applied for all the letters in plaintext to produce a cipher text. All the letters in the alphabet are considered circularly (‘Z’ will be replaced by ‘C’), i.e., position of a letter in alphabet will be shifted by 3 positions. This shifting do not have to be 3, it can be any variable ‘k’. Though this technique has a key space of 26, it is found to be easy to break.

Definitive cryptography says that using a key will make the breaking process difficult because retaining the substitution will be time consuming as well usage keys were introduced into substitution. Here a key is considered and starting letters of the alphabet are substituted with the letters of the key. Remaining letters will be substituted by the letters that are not included in the key in alphabetical order. If a same letter appears again, it will simply be discarded.
1.1.2. MD5 Hashing

MD5 stands for message digest algorithm 5. This is a hashing algorithm that can be used as a digital signature mechanism. This is a widely used hashing algorithm whose hash value is 128-bits. This algorithm takes in a variable length input but gives a fixed length output of 128 bits. The given input is first divided into individual blocks of 512 bits each. To make the blocks size divisible by 512, the last message block could be padded. While padding, first a single bit ‘1’ is added at the end and may be followed by many zeros until the blocks size can be divisible by 512. This algorithm uses four variables called state variables each of size 32-bits [5] [9]. These state variables are initially stored with some default hexadecimal values. This algorithm also has four predefined functions that works on AND, OR, XOR and NOT operations. These functions use the state variables and message as input and convert the state variables from their original form to message digest. The generated digest will be stored in state variables. To get the final message digest, hexadecimal value of each state variable were taken as output. This is also called one-way hashing.

1.1.3. Data encryption standard (DES)

DES is an encryption standard where encryption is done in individual blocks called block cipher. The block size used here is 64 bits. The core idea behind this standard is Fiestel network. This standard involves 16 identical stages in its process. Each block of 64 bits is divided into two blocks; left and right of 32-bits. The right part is given as input to a Fiestel function. An XOR operation is applied between the output of Fiestel function and the left part and the resultant is considered as the right part of the second stage. For the left part in the second stage, the right part from the previous stage is simply copied [11] [6].
The same procedure is contained for 16 iterations to get a final output of the 64-bit block. This is how each block of 64-bits is encrypted with DES.

![Encryption circuit block diagram](image)

**Figure 1: Encryption circuit block diagram [6].**

Decryption on the other hand, is just a reverse process to encryption which can also be called as inversion where an XOR operation is applied between the right part and the output of the Fiestel function. The resultant is considered as the left part to the next stage. And the left part is just copied as the right part of the next stage. This process is carried out through all the 16 stages.

![Inverse circuit of encryption circuit](image)

**Figure 2: Inverse circuit of encryption circuit [6].**

**Fiestel Function:**

This function takes 32-bits of a block as input and submits to an expansion function. The task of the expansion function is to expand the input; therefore this expansion function takes 32-bit information as input and gives 48-bit output. The Fiestel function takes another input (key) which is of 48-bits length between the key value and 48-bit output from expansion function, an XOR operation is performed. From the resultant 48-bits, every 6 bits were given as input to an S-Box and a total of 8 S-boxes were used [11]. Each s-box
gives 4 bits of information as output; the output file is 32-bits in size.

1.1.4. Least Significant Bit Substitution (LSB)

LSB substitution is a popular technique to embed data on to digital images. We know that an image will be stored in the form of bytes. In this kind of encoding, by using the LSB of each byte, 1-bit information can be stored in the image as secret message [8]. Accordingly 1-bit per byte can be stored in 8-bit images while 3-bits can be stored in 24-bit images for every 24-bits. Depending upon the color palette of a cover image, a secret message can be stored in two LSB’s which cannot be identified by human visual system (HVS) [8] [9]. But the main drawback of this encoding method is that images after encoding can be intercepted easily i.e, information can be changed or image format can be changed.

1.1.5. Jsteg

This was the first publicly available steganographic system for JPEG images. This encoding technique is similar to that of the LSB technique. This technique uses the concept of discrete cosine transformation (DCT) [11] [13]. The JPEG image format uses a discrete DCT to transform successive 8 × 8 pixel blocks of the image into 64 DCT coefficients each. Here, encoding is done by sequentially replacing the LSB of DCT coefficients with message’s data. Andreas Westfeld and Andreas Pfitzmann noticed that steganographic systems that change least-significant bits sequentially cause distortions detectable by steganalysis [1] [8] [11]. The disadvantage with this system is, embedding step changes the LSB of colors in an image, that is, embedding uniformly distributed message bits reduces
the frequency difference between adjacent colors.

1.1.6. Embedding Process

Interpreting technically, an image is stored in the form of an array of numbers on a computer that represents a collection of pixels, representing light intensity at various points of the image. Typically, a pixel could carry 8-bits or 24-bits of data depending upon the image quality. An 8-bit image is smaller in size which can be dependable, but one disadvantage is that, it exhibits only 256 colors. This may affect adversely while encoding. Hence, while handling an 8-bit image, gray scale color palette is used. Use of 24-bit images increases efficiency because it can exhibit many colors (more than 16 million) [10] [12]. A file with bigger size (usually in Mega Bytes) can make it more suspicious when transmitted on the Internet.

1.2. History

1.2.1. Cryptanalysis

Apart from encoding techniques, there are some universal tricks that are capable of breaking ciphertext into plaintext without any clue of keys or what the message has. This is process of decoding a ciphertext is called cryptanalysis, that is a counter technique to cryptographic methods. The base idea behind cryptanalysis is repetition of letters. Every language will have many words to exchange information [3]. For example English alphabet has only 26 letters but combination of those letters can give many words: enough to communicate. This idea gave birth to ‘frequency analysis’ technique in cryptanalysis.
In the process of cryptanalysis, an analysis will check for the distribution of frequency of the letters in the cipher text and compare it with the distribution of the frequency of normal alphabet. Looking at the comparison results, the analyst will substitute appropriate letters to get plaintext. The letter with more frequency in English is ‘e’. There are many tools that work on character substitution techniques, CRANK is the popular attacking tool for this techniques [3] [4].

1.2.2. Steganalysis

Attacking techniques for steganography were also developed. When compared to cryptanalysis, steganalysis is a more challenging task for the analyst, because ciphertext can easily be identified by looking at it. Whereas a message hidden in an image cannot be determined or even look suspicious to the eye of the analyst. There are three attacks possible which are considered as steganalysis: Visual attack, Structural attack and Statistical attack [1] [3].

The simplest form of steganalysis is visual attack, in which the analyst’s attack is by observing subjected file with naked eyes. The first rule of good steganography is to keep the stego-image unchanged. Despite of that, removing some parts of the images that were not altered and just focus on the altered parts make the visual attack a successful one. Therefore the main idea of this attack is to identify which parts of the stego-image are to be considered. The study of image says that, in an image, the bits with value 1 are almost equal to bits with value 0. That is there are approximately equal number of 0’s and 1’s in an image. Embedding of some plain data will disprove this fundamental rule; but if the embedded data is encrypted, then the fundamental rule will remain the same.
Structural attacks mainly focus on the high level properties of a particular method or algorithm. However, when we remove parts of the image that are not altered as a result of embedding a message, and instead concentrate on the likely areas of embedding in isolation, it is usually possible to observe signs of manipulation. The ‘Hide & Seek’ steganographic algorithm can be applied only to images whose size is $320 \times 480$ pixels, which is an example of this attack. In this case, an analyst can consider all the images satisfying those size specifications as a suspicious image to perform his attack.

Statistical attacks are mathematical approaches that involve some statistical approach. In a dataset the presence of some peculiar data which is random, can be identified using Statistics. With the help of this strategy the analyst could break the stego-image into two sets, one is image data and the other will be information data. All the information regarding the image that can be seen, which is fully related to pixel values and the colors of those pixels are included in image data. Information about the hidden message comes under information data.

1.3. Objective

Using a multilevel approach, the main objective of this project is to provide high level security to sensitive digital data against steganalysis techniques, while keeping the operational time low. This objective can be achieved by encrypting and authenticating the image that has user’s sensitive data and then embedding the resultant document on to a carrier.
1.4. Rationale

Initially, there were not many tools that could block the cryptanalysis or steganalysis attempts at various levels to secure/protect the hidden data in text or carrier media respectively. There were many situations this tool was needed. For, example attacks on military secrets and plans were occurred many times in many countries. Data hiding is a very important viewpoint of any user who transmits confidential information through networks. Sometimes trade secrets may have to be shared among different industries. In the same way, there can be many situations that require data hiding.

Though there were many methods introduced for data hiding, still there has been a challenge. Many of them were designed in a way that follows the approach which shows a small change to previously introduced approaches. Each of them has their own disadvantages. Few of which are discussed in this document.

Digital images are being used as most common carriers to hide secret messages because the human visual system (HVS) has limitations. That is, the human visual system has no function like abstracting the illusion effects on what it looks [1] [12]. Steganography took advantage of using digital images so that small visual changes to an image cannot be suspected. Essentially any media such as plaintext, ciphertext, can be hidden in a digital image that can be converted into a bit stream.

The new tool proposer here is a combination of both cryptography and steganography. This tool uses a digital image as a carrier to take advantage over HVS. This tool would try blocking an analyst through multiple levels up to maximum extent. The mechanism of this new tool is discussed below along with some screen shots of the tool while running it.
2. Narrative

2.1. Problem statement

Currently many cryptography and steganography techniques have come into existence. Encoding of plaintext is achieved using DES, AES, Triple DES, RSA and many other algorithms. Any individual can use his/her own approach as encryption method.

Many algorithms such as JSteg, JPHide and JPSeek, OutGuess, F3, F4 and F5 were invented for the purpose of embedding images. These algorithms follow a certain principle to embed and retrieve hidden contents. All the existing approaches have their own disadvantages as they can easily be compromised using steganalysis. It means that one way or another, an intruder can figure out the existence of hidden data which results in him/her compromising sensitive data. Currently, no integrated cryptography and steganography approach in one application exists for image based information security. There are encryption and embedding approaches present that work with plaintext only.

2.2. Motivation

As described above all the available techniques used in early tools are old and follow some specified process with some improvements to previously proposed techniques. This makes the intruders work easy. The intruder may try a counter attack by making some changes to counter existing techniques. None of the existing techniques offers protection through multiple levels. That is one of the reasons why an intruder is able to view/obtain hidden data with just one or two attacks.
2.3. Scope

The primary idea behind developing this project is to protect confidential data from an intruder’s counter-attacks and to block the intruder through various levels in his/her attacks. A new tool has been developed with a combination of cryptographic encryption and steganographic encryption for its implementation. The developed steganographic tool has a sender’s segment that can take a message, a password and a cover image as input and give a stego-image as output that has message embedded in it. On the other hand, it also has a receiver’s segment where the receiver inputs the stego-image and the same password is used by the sender as input to get the sender’s message as output. The project is tested with various inputs and made sure that the generated stego-image has no noise are data loss.

2.4. Functionality

The developed steganographic tool is a very useful to any user who shares confidential data through a network. The developed model has a customized access that gives more freedom to users. An interface has been developed that helps the user to interact with the tool. The interface is very user-friendly with different modules implemented to encode and decode the secret message. The developed tool was tested for various input conditions.
3. Description of the system

3.1. Implementation

The disadvantage with existing steganographic systems is their simple data embedding procedure. Because of this single level encryption of information the counter attack is being done at one level only and by applying all possible counter techniques blindly it may result in compromising the data. Our system is intended to encrypt data through multiple levels.

3.1.1. Base64 Conversion

As part of multi-level approach the base64 conversion is taken as the first step. The intended purpose of base64 mechanism is to convert any unreadable data into a readable format. As mentioned earlier, an image to a computer is just a stream of bytes. Each byte is composed of 8-bits, and each bit is capable of storing one value, either ‘0’ or ‘1’ which we all know as ASCII standard. The ASCII standard is an 8-bit character set which has 256 ($2^8$) characters that form text. Among those 256 characters, only few are readable and printable, but the idea here is to generate a readable text, which could be done by decreasing the number of character set [6] [11]. To reduce the character set, the Base64 mechanism reads only 6-bits as a character instead of 8-bits. Now we will have only 64 ($2^6$) characters which are represented by upper case alphabet (26), lower case alphabet (26), numbers 0-9 (10) and symbols ‘+’,’/’ that is a total of 64 characters. But every byte has 8-bits, so reading just 6-bits would be troublesome. Using 3 bytes 24-bits are formed and these 24-bits are chopped into 4 equal parts of 6-bits each.
Encoding:

To reduce the number of characters, 8-bits are reduced to 6-bits by considering 3 characters at a time. The actual translation follows mapping of values from 0 to 63 characters (A-Z, a-z, 0-9, +, /). For this, a chart is used as shown in figure 3.

<table>
<thead>
<tr>
<th>Value:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64:</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>Value:</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>base64:</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>Value:</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>base64:</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
</tr>
<tr>
<td>Value:</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>base64:</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>+</td>
<td>/</td>
</tr>
</tbody>
</table>

Figure 3: Character for base 64 [6].

For example, consider a string “Hello World!” Base64 interpretation for given string is as follows [6]:

1. Convert the characters to binary.

   "Hel" is 01001000 01100101 01101100 in binary. (Notice that there are 24 bits).

2. Convert the 24 bits from three 8 bit groups to four 6 bit groups.

   01001000 01100101 01101100 becomes 010010 000110 010101 101100.

3. Convert each of the four 6 bit groups into decimal.

   010010 = 18

   000110 = 6

   010101 = 21

   101100 = 44

4. Use each of the four decimals to look up the base64 character code.
18 = 'S'
6 = 'G'
21 = 'V'
44 = 's'

6. You now have your first three ASCII characters ("Hel") encoded as base64 ("SGVs").

**Decoding:**

The decoding process is relatively simple process. The most important thing here is that while encoding 8-bits were used for each character. But now for each character only 6-bits should be used, so to decode the information 4 characters are to be considered at a time [6].

For example decoding of string “YmFz” is as follows:

1. Convert the base64 characters to binary. (Remember to use 6 bit binary!)

"YmFz" is 011000 100110 000101 110011 in binary.

2. Convert the 24 bits from four 6 bit groups to three 8 bit groups.

011000 100110 000101 110011 becomes 01100010 01100001 01110011.

3. Convert each of the three 8 bit groups into decimal.

01100010 = 98
01100001 = 97
01110011 = 115

4. Use each of the three decimals to look up the ASCII character for that value.

98 = 'b'
97 = 'a'
115 = 's'

We now have the first four base64 characters ("YmFz") decoded as ASCII ("bas").

In the same manner, data of any format can be encoded and decoded on the basis of 64 \((2^6)\) characters.

3.1.2. **Password based encryption (PBE)**

PBE with MD5 and DES is a secure cipher class provided by JAVA itself. This project makes use of the service provided by JAVA. A small function call for encryption and decryption returns the encrypted and decrypted file by directing the input respectively.

Here PBE is used with MD5 and DES. An 8 character password is given as input to this algorithm. The MD5 hashing generates a 128 bit stream with the given input (password) [5] [6] [13]. This 128 bit stream is used as key input which is necessary for the DES encryption as explained in literature review. Therefore, encryption is carried out in the form of blocks which is called block ciphering based upon a password.

3.1.3. **Embedding**

Embedding is a process that inserts the bits of information into the byte array stream of cover image. The following describes the process.

Insertion of data should not change the information. That means, after inserting information into the cover image, and when the information is retrieved by receiver from
image, the retrieved information should be same as the inserted information [7] [8] [10].
The information to be inserted is encrypted twice before insertion, so the double encrypted information is processed first in order to keep the information consistent. The main problem could be only with the characters followed by symbol ‘\’. Some of the characters followed by back slash ‘\’ could form an escape sequence character. To remove this affect a single backslash is replace by multiple backslashes.

The cover image is processed and interpreted in the form of bytes. That is, information of each pixel of the cover image is stored in the form of a byte for corresponding RGB values [8], [10]. The last bit of each interpreted byte of all the pixels is replaced with one bit of information that is to be inserted. In the same way, each and every bit of information that are be hidden is inserted into available pixels of the cover image till the end of information.

Threshold:

The insertion of bits of information is not straight-forward, but is done by selecting appropriate pixels to insert. That means, a threshold value is given as input by the user. And the pixel that satisfies the threshold (pixels value is greater than the threshold) is eligible for insertion. Before checking the threshold value of a pixel, its position is also considered to be even [7] in terms of row and the column index. Finally any pixel that is in an even position and whose value is greater than the user supplied threshold value is used for data to be inserted in it.
3.1.4. Embedding Process

The following steps describe the embedding process:

a) Process the information that has to be inserted to remove escape sequences. The code written below exhibits this operation:

   \[
   \text{information} = \text{"\" + information.replaceAll("\\\", "\\\\\\\\\") + "\\"};
   \]

b) The cover image is processed in the form of bytes by the colors of pixels and stored in the form of an array. This process is implemented with following code:

   \[
   \text{for (int } i = 0, k = 0, l = 0; i < image.getWidth(); i++) \{

   \text{for (int } j = 0; j < image.getHeight(); j++, k += 3, l++) \{

   \text{rgb = image.getRGB(i, j);} \\
   \text{bytes[k] = (rgb >> 16) & 0xFF;} \\
   \text{bytes[k + 1] = (rgb >> 8) & 0xFF;} \\
   \text{bytes[k + 2] = (rgb >> 0) & 0xFF;} \\
   \text{alpha[l] = (rgb >> 24) & 0xff;} \\
   \}
   \}
   \]

c) Check the available size of the cover image to insert information and throw an exception if there is no enough space available. To make it work, following lines are used in code:

   \[
   \text{if ((bytes.length - 2) / 8 < information.length()) \{} \\
   \text{System.out.println("Secret Message exceed the threshold");} \\
   \text{throw new NotEnoughSpaceException("Information to long for Picture"); \};}
   \]

Now information and cover image both are in bytes form, so filtering of pixels and
insertion of information can be done.

d) To insert a value into a last bit of byte, the pixel should be in even position both in row and column. And also the value of the pixel should be greater than the input threshold. The following lines of code give a brief look at the internal operation:

```java
for (int i = 0, k = 0; i < information.length(); i++) {
    int cur = information.charAt(i);
    for (byte j = 0; j < 8; j++, k+=2) // filter by position (even position only)
    {
    }

    if (bytes[k] > ThresholdBean.getThreshold())
        // threshold should be satisfied
        bytes[k] = setLastBit(bytes[k],((cur & 1 << 7 - j) >> 7 - j) == 1);
}
```

e) This insertion is done until the end of information or till the end of last pixel of cover image.

f) The process of retrieving information is done in similar manner. The last bit of a pixel satisfying both position and threshold conditions is read into a variable. The following code describes the process:

```java
for (int i = 0; i < bytes.length;) {
    cur = 0;
    for (int j = 0; j < 8 & i < bytes.length; j++, i+=2) // i value double increment
    {
    }
```
if (bytes[i] > ThresholdBean.getThreshold())
  cur |= (bytes[i] & 1) << 7 - j;
}

3.2. System Architecture

3.2.1. Architecture

Figure 4 represents the architecture that is implemented. The modules of the steganographic tool are also included in the architecture. The user can either be the sender or the receiver.

Figure 4: System Architecture

A user is able to do all the operations as shown in figure 4 manually or automatically by using auto-mode. Both the manual and auto-mode follows the same system flow.
3.2.2. Flow of Execution

Figure 5 shows the flow chart with the encryption part of the tool. The purpose of this project is to hide an image in another image, so a secret image would be an input. At first, the secret image is converted to a text file using Base64 conversion. Then the generated text file is encrypted with a password-based encryption algorithm to generate an encrypted text file called ciphertext. Using a customized embedding algorithm, ciphertext is embedded onto a cover image. The output is the stegogramme (a cover image with a secret message embedded in it).

Figure 5: Flow of execution of sender’s operations.
Figure 6 shows a flow chart with the decryption part of the project. To read the hidden message (secret image), the stegogramme has to be decrypted. So, the stegogramme is used as input to the retrieving algorithm. If the retrieving algorithm is not the same as the embedding algorithm, there is no way that the correct output can be obtained. The correct output from retrieving algorithm is the ciphertext is used as input to the decryption algorithm. This decryption algorithm is the same as the encryption algorithm; otherwise the secret message cannot be determined. And also the decryption algorithm takes a key (password) to generate plaintext.

Figure 6: Flow of execution of receiver’s operations.
The correct password has to be used to get plaintext. Then, that plaintext is given as input to the Base64 converter to reconstruct the secret image for the plaintext.

### 3.3. System Resources

JAVA is used to implement the proposed steganographic application and as expected, the application can be used for any platform that supports JAVA v1.6. For the sake of development and testing, the following hardware and software systems are used.

The hardware and software involved are as follows.

- Intel i3 core
- Windows 7
- JDK v1.7
- NetBeans IDE7.1.2

### 3.4. System Requirement

Processor should contain 2.2 GHz speed for better performance and System required 40GB hard disk space for the Software and Operating System and 128MB free RAM space for the application; however RAM space varies because of the image operations and image size.
3.5. User Interface Design

3.5.1. Main Window

It is written in java swings. It is used to open the operational window or frames to perform set of operations on images using the specified techniques. It consists of basic window operations like closing and minimizing but restoring or rescaling is not possible. This main window has two tabs encode and decode which allows the user to perform his task.

3.5.2. Encode Tab

The encode tab directs the user through three windows: Base64, Encrypt and Embed while all these windows can be executed individually. These windows were developed using Applet as well as Swing framework. Use of all these three windows is not mandatory; they can be used based upon user’s security requirements. As the name of this tab says, windows involved in this tab are used in the process of encoding an input. A user can perform base64 conversion of a message, password based encryption of a message and embedding a message into a cover image.

This window consists of all kinds of operations related to encoding, which are required for the developed steganographic tool.

3.5.3. Decode Tab

It is purely written in java swings. Similar to “encode” tab, this tab has three windows: “base64”, “decrypt” and “retrieve” which are the respective complementary decoding methods for encoding operations on “encode” tab.
Operations under this tab are to be used by the receiver to read/view the secret message.

3.5.4. Auto mode Tab

This tab is for automatic functioning of both “encode” and “decode” tabs. That is, all the three operational windows of “encode” and “decode” tabs are integrated into a single window “encode” window and another single “decode” window respectively.

3.6. Software Engineering Techniques

3.6.1. Requirements

3.6.1.1. Software Tools

An Integrated Development Environment tool such as Netbeans IDE is used in this project. Because of this tool, construction of user interface becomes easier and we can represent interface in Rich Graphical representation. It gives feasibility for the inbuilt classpath, path settings, and resource links. We can view the interface design without running the code. And modifications can be done easily by using this tool.

3.6.1.2. Functional and Non-Functional Requirements

The ‘functional requirements’ satisfied by the tool are as follows:

i. Main window is the start of the application. Using this, a user can perform individual operations like encoding and decoding applicable at various levels in separate manner or individually.

ii. Encoding tab leads the user to three windows that are used to perform
cryptographic and steganographic operations on images. A user can transform the secret image into a text file and encrypt the text file based upon a password and he can also embed the encrypted file into a cover image. The entire file generated by the user can be saved on desired location in a computer on which the tool is running.

iii. Decode tab is the tab where a receiver can generate his secret message. The embedded image can be processed here. With this tool a user can retrieve the information embedded in the image. The user can also decrypt an encrypted file with the help of a password. Finally he will also be able to generate the secret image from a text file.

Among many ‘non-functional requirements’, the project must be very user-friendly. People will likely use such software tools with just a little more knowledge in using computers other than for everyday tasks.

The developed steganographic tool satisfies all the above requirements.

3.6.1.3. Structural Requirements

The tool satisfies the following structural requirements.

The project is to provide user-friendly operational steganographic tool for the multi-level protection. This tool allows users to operate easily on images and files with different formats like .txt, .png, .enc etc.

a) It gives Self-explanative user interface.

b) On clicking encode tab, all the encoding options appear.

c) On clicking decode tab, all the decoding options should appear.
d) All the buttons that are used to instruct the tool should appear.

When a user closes the main window all operational windows must stop their action and close.

Encode tab:

a) A user can generate a text format of an image file.

b) After generating a text file, a user can encrypt the text file with a password.

c) Finally a user can embed the encrypted file into a cover image, with format of the image being PNG so that it will be easy to transfer in any network.

d) A user is allowed to save all the output files in the hard disk.

Decode tab:

a) A user is allowed to load stego-image to retrieve hidden information.

b) A user is allowed to decrypt the encrypted file using same password that was used at the time of encryption.

c) At the end, a user can generate an image from the retrieved information.

Auto mode tab:

a) The proposed operations, either encoding or decoding can be performed automatically without going through all the steps.

b) All the encoding and decoding operations are integrated into two individual windows i.e., encode and decode respectively.
c) All the inputs are given at a time both for encoding and decoding operations.

3.7. Analysis

The important task of this project is to embed an image into an image, not directly but by providing security through multiple levels. With the proposed system code, the requirement is fulfilled with any digital format of input. All the inputs can be transformed to appropriate format and any information can be embedded into a cover image. After embedding, they are automatically stored in portable network format for sharing purpose on network. Hence the input image can be of any format (jpg, jpeg, png, gif) for encoding and gives out a cover image with input embedded in it. The receiver’s side of this tool takes the cover image as input. By applying appropriate operations on the cover image, the secret digital image is given as an output.
3.8. Design

Users and System:

![Use case diagram of the system.](image)

**Figure 7: Use case diagram of the system.**

Users can interact with the system through interface. A user can either be a sender or a receiver; therefore, all the conversion, cryptographic and steganographic operations can be performed by the user.

Base64 encoding:

First the user runs the tool; then selects the base64 window using the interface via “encode” tab. Next user selects the image that is to be encoded and sets the destination location for the output (text file). To do so the user has to interact with the system through the interface. The “encode” tab gives all the encoding functions to the user, where the user selects base64 encoding to convert an image to a text. Figure 8 shows the sequence diagram for base64 encoding process.
Figure 8: Sequence diagram for image to text conversion.

Embed Sequence Diagram:

Figure 9: Sequence diagram for embedding process.
Figure 9 is a sequence diagram showing the flow of instructions in the process of embedding. A user interacts with the system to embed the message in cover image. The embed function invoked by the user’s operation processes the input message in the form of bits information. This information in the form of a bit stream is inserted into the bytes of the cover image by processing the cover image.

System Component Diagram:

![System Component Diagram](image)

**Figure 10: System component diagram.**

The system consists of four main components namely “Base64 conversions”, “encrypt and decrypt” operations, “embed and retrieval” of information and finally the “interface”. All the components can be run individually without the interference of other components. All functional components depend on the interface to run.
4. Functionality of Application

In this section, all the modules of the developed tool are discussed. This is done with the help of screen shots.

4.1. Modules Description

The modules were categorized into two types, based upon the type of the user. One is the sender’s side and the other is the receiver’s side.

Sender’s side:
- Base64
- Encrypt
- Embed
- Automode

Receiver’s side
- Retrieve
- Decrypt
- Base64
- Automode

The developed tool does not use any backend database for its operations. This can work on the local disk’s memory. All the modules are executed independently. Working of the tool is explained in this section using some screen shots as follows.
Figure 11, shows the home screen or main window of the developed tool. It shows three tabs; one is encode that works on sender’s side, one is decode tab that works on receiver’s side. The third tab is auto mode tab that works for both the sender and the receiver. Using this tab all the sender’s operations can be done through one window as well as the receiver’s tab also. Communication begins from the sender’s side only when a sender hides his secret image in the cover image, and transmits the stego-image to the receiver.
Figure 12: operations available for a sender.

Figure 12, shows the three operations that a sender can perform. According to the proposed model, the sender starts with base64 operation where a user is supposed to give his/her secret image as input. At the end of this operation user will have a text file as an output (image converted into text). The second option is the encryption operation, here text file is given as input and the expected output is an encrypted file. Finally the encrypted file is embedded in the cover image using the Embed option and the user (here sender) will end up with a stego-image as outcome.
Figure 13: Operations available for a receiver.

Figure 13 shows the available operations for the receiver. According to the proposed model, the receiver starts with the retrieve option. This tab shows the operation that works in a quite reverse manner to sender’s operations. The receiver uses a stego-image as his first input and extracts the embedded information using the “retrieve” operation. According to the proposed model, the retrieved information is encrypted information. A user is able to decrypt the encrypted file using decrypt operation and gets a text file as output. Finally the generated text file is converted to an image using base64 option that is a base64 decoding is done on the text file to generate the secret image as output.
Figure 14: Available user operations in auto mode.

Figure 14 shows the operations available to the user. The encode option performs all the operations under the encode tab, that is, base64 encoding, encryption and finally embedding. These are done automatically by taking the required input at once to give the stego-image as output. The decode option performs all the operations under the decode tab, that is, retrieve, decrypt and finally base64 decoding. These three operations are also done automatically by taking all the inputs at a time. Finally the user (here receiver) will have the secret image as output.
Figure 15 shows the encoding window with auto mode. Here the user has to give his secret message as input. User will be able to browse through the computer to select his/her secret image using ‘select source’ button. To hide the secret image user requires a cover image, he/she will be able to select the cover image by browsing the computer using the ‘select source’ button. The final location to save the output file should also be selected by the user using ‘select destination’ button. As the encryption here is a password based encryption, a password with a minimum of 8 characters should also be given as input. A final stego-image will be generated after giving the threshold value and clicking the ‘embed secret’ button.
Figure 16: Decoding window in auto mode.

Figure 16 shows the decryption process window for the user in auto-mode, where the reverse process to the sender’s operations is done automatically. The input here should be a stego-image which can be given as input by browsing the computer. The destination location for the output file to save should also be given by the user. The password used and the threshold value should be similar to that of the password and threshold used by the sender respectively. Final image will be generated by clicking ‘retrieve secret’ button.
5. Evaluation

In order to determine the effectiveness of the tool in hiding a secret message, we carried out two experiments. One experiment deals with structural attack that counts the number of zeros and ones in an image. We also conducted a survey for determining mean opinion score on the effectiveness of the tool. The following summarizes the results of the experiments.

5.1. First Experiment

<table>
<thead>
<tr>
<th>Size of secret message</th>
<th>Applied Threshold</th>
<th>Cover Image Number of 0’s</th>
<th>Cover Image Number of 1’s</th>
<th>Difference between 0’s and 1’s</th>
<th>Stego Image Number of 0’s</th>
<th>Stego Image Number of 1’s</th>
<th>Difference between 0’s and 1’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 KB</td>
<td>20</td>
<td>52.72% (982433)</td>
<td>47.27% (880960)</td>
<td>5.44% (101473)</td>
<td>47.54% (6319893)</td>
<td>52.45% (6973644)</td>
<td>4.91%</td>
</tr>
<tr>
<td>20 KB</td>
<td>16</td>
<td>52.72% (982433)</td>
<td>47.27% (880960)</td>
<td>5.44% (101473)</td>
<td>51.94% (7689637)</td>
<td>48.05% (7112764)</td>
<td>3.89%</td>
</tr>
<tr>
<td>30 KB</td>
<td>12</td>
<td>52.72% (982433)</td>
<td>47.27% (880960)</td>
<td>5.44% (101473)</td>
<td>48.36% (7062084)</td>
<td>51.63% (7540057)</td>
<td>3.27%</td>
</tr>
<tr>
<td>40 KB</td>
<td>8</td>
<td>52.72% (982433)</td>
<td>47.27% (880960)</td>
<td>5.44% (101473)</td>
<td>51.67% (8259114)</td>
<td>48.32% (7723763)</td>
<td>3.35%</td>
</tr>
<tr>
<td>50 KB</td>
<td>4</td>
<td>52.72% (982433)</td>
<td>47.27% (880960)</td>
<td>5.44% (101473)</td>
<td>51.68% (8541652)</td>
<td>48.31% (7985963)</td>
<td>3.37%</td>
</tr>
</tbody>
</table>
Table 1 shows some simulation results that can describe the properties of the stego-image, generated using the proposed tool. Five different images with different sizes were selected. Five stego-images were generated by embedding the selected images in to a cover image. All the stego-images were generated using the same cover image. Same cover image is used to have a single reference. The table has the values of number of 1’s and 0’s in the cover image and stego-image and also the difference between 1’s and 0’s for both the cover image and the stego-image. The comparisons are shown in the form of percentages for convenience. Looking at the results it is proven that the number of 1’s and 0’s are approximately equal in all the stego-images that are generated using proposed tool. That is the embedding process involved in with this tool preserves the image properties. This keeps the analyst attacks in steganalysis a step away. This can be taken as an advantage.

5.2. Second Experiment

Table 2 shows the opinion of five different evaluators who operated the tool and gave their feedback about the results obtained. Looking at the mean opinion of those five evaluators it is clear that, the stego-image generated is no way different or noisy that cover image. The encoding process will not change the quality of the stego-image, which is a first and foremost rule of steganography.
Table 2: Mean Opinion Score

<table>
<thead>
<tr>
<th>User</th>
<th>Observed difference between cover image and Stego-image</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>No Difference</td>
<td>Nil</td>
</tr>
<tr>
<td>User 2</td>
<td>No Difference</td>
<td>Nil</td>
</tr>
<tr>
<td>User 3</td>
<td>No Difference</td>
<td>Nil</td>
</tr>
<tr>
<td>User 4</td>
<td>No Difference</td>
<td>Nil</td>
</tr>
<tr>
<td>User 5</td>
<td>No Difference</td>
<td>Nil</td>
</tr>
</tbody>
</table>

6. Conclusion

The developed steganographic tool is used to encrypt and decrypt the image. In this project, security to confidential data is achieved through multiple levels with the combination of both cryptographic and steganographic strategies. In the process of embedding information into the cover image, a successful threshold strategy is used. A bit of information is inserted into a pixel only if the pixel satisfies threshold value and position constraint. The embedding image can be of any format (jpeg, pjg, gif, png). The generated stego-image is in .png format because the image quality of this format is reasonable with the file size. All the operations are done with user-friendly interface. Any user, either a sender or a receiver can operate the tool without any basic knowledge just by clicking a few buttons.
Evaluations for both the structural and visual attacks were performed which show positive results. This demonstrates that, using this tool the data confidentiality of the hidden message can be achieved as needed.

7. Future Work

As a part of security, the pixels of the cover image are filtered both according to their position and the threshold limit. Because of this, the space availability of data insertion could become very less. Therefore, the embedding information should be small for successful embedding. New ideas could be developed on increasing the space availability in the cover image to insert as much data as possible.
8. References


International Conference on computer vision 2007.


Appendix A: Code

Manipulation of Bits:

```java
package embed;

public class BManipulation {
    public static int[] manipulateBytes(int[] bytes, String information)throws NotEnoughSpaceException {
        int threshold = information.length();
        System.out.println("Threshold is :" + threshold);
        if ((bytes.length - 2) / 8 < threshold) {
            System.out.println("Secret Message exceed the threshold");
            throw new NotEnoughSpaceException("Information to long for Picture");
        }
        for (int i = 0, k = 0; i < information.length(); i++) {
            int cur = information.charAt(i);
            for (byte j = 0; j < 8; j++, k++) // k value double increment
            {
                if (bytes[k] > ThresholdBean.getThreshold())
                    bytes[k] = setLastBit(bytes[k],((cur & 1 << 7 - j) >> 7 - j) == 1);
            }
            return bytes;
    }

    public static String getInformation(int[] bytes, boolean all) {
        StringBuilder information = new StringBuilder();
        int cur = 0;
        for (int i = 0; i < bytes.length;) {
            cur = 0;
            for (int j = 0; j < 8 && i < bytes.length; j++, i++) // i value double increment
            {
                if (bytes[i] > ThresholdBean.getThreshold())
                    cur |= (bytes[i] & 1) << 7 - j;
            }
            information.append((char) cur);
            if (!all && information.length() > 2
                && information.charAt(information.length() - 2) == \n```
&& information.charAt(information.length() - 1) != \\
&& information.charAt(information.length() - 3) != \\
} {
    return information.substring(1, information.length() - 3)
        .replaceAll("\\\\\\", "\\\\");
} else if (!all && & information.charAt(0) != \) {
    return null;
}
return information.length() > 2 ? information.substring(1,
    information.length() - 3).replaceAll("\\\\\\", "\\\\")
    : information.toString().replaceAll("\\\\\\", "\\\\");

private static int setLastBit(int zahl, boolean flag) {
    zahl >>= 1;
    zahl <<= 1;
    if (flag) {
        zahl |= 1;
    }
    return zahl;
}

Image conversion:

package embed;

import java.awt.Color;
import java.awt.image.BufferedImage;

public class IConvert {

    private int[] alpha = null;

    public int[] getBytes(BufferedImage image) {
        int rgb = 0;
        int[] bytes = new int[image.getWidth() * image.getHeight() * 3 + 2];
        this.alpha = new int[image.getWidth() * image.getHeight()];
        for (int i = 0, k = 0, l = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++, k += 3, l++) {
                rgb = image.getRGB(i, j);
                bytes[k] = rgb >> 16 & 0xff;
                bytes[k + 1] = rgb >> 8 & 0xff;
                bytes[k + 2] = rgb & 0xff;
            }
            bytes[k + 2] = 0;
        }
        return bytes;
    }
}
bytes[k] = (rgb >> 16) & 0xFF;
bytes[k + 1] = (rgb >> 8) & 0xFF;
bytes[k + 2] = (rgb >> 0) & 0xFF;
alpha[l] = (rgb >> 24) & 0xff;

bytes[bytes.length - 2] = image.getWidth();
bytes[bytes.length - 1] = image.getHeight();
return bytes;

public BufferedImage getImage(int[] bytes) {
    BufferedImage lastImage = new BufferedImage(bytes[bytes.length - 2],
    bytes[bytes.length - 1], BufferedImage.TYPE_INT_ARGB);
    for (int i = 0, k = 0, l = 0; i < lastImage.getWidth(); i++) {
        for (int j = 0; j < lastImage.getHeight(); j++, k += 3, l++) {
            lastImage.setRGB(i, j, new Color(bytes[k], bytes[k + 1],
            bytes[k + 2], alpha[l]).getRGB());
        }
    }
    return lastImage;
}

public int[] getLastReadAlphaChannel() {
    return alpha;
}

public void setAlphaChannel(int[] alpha) {
    this.alpha = alpha;
}