Chapter 3: Steganography Techniques

To understand how steganography works and to bring the insight about the known embedding techniques this chapter, we are to discuss about some of the most documented steganographic approaches, splitting them into appropriate groups to illustrate clearly how they differ from each other. Some are easier to implement than others, and some are more robust than others. This as we will see later on in this phase of the research - is one of the biggest trade-offs with steganography. The question steganographers usually face is: "how well do you want to hide the existence of a secret message?" If the message just needs to be hidden from one person with little computing experience, then a simple algorithm may be preferred. However, terrorist cells would be looking to hide the existence of the message from government defense agencies typically comprised of some of the most highly skilled people in computing; in that case a clever algorithm is likely to be preferred.

3.1 Classification of Existing techniques

Image steganography techniques can be divided into two groups...

A. Image Domain

B. Transform Domain [5].

Image – also known as spatial – domain techniques embed messages in the intensity of the pixels directly, while for transform – also known as frequency – domain, images are first transformed and then the message is embedded in the image [42].

Image domain techniques encompass bit-wise methods that apply bit insertion and noise manipulation and are sometimes characterized as “simple systems” [23, 25]. The
image formats that are most suitable for image domain steganography are lossless and the techniques are typically dependent on the image format [44].

Steganography in the transform domain involves the manipulation of algorithms and image transforms [25]. These methods hide messages in more significant areas of the cover image, making it more robust [43]. Many transform domain methods are independent of the image format and the embedded message may survive conversion between lossy and lossless compression [44].

3.2 Spatial Domain techniques

3.2.1 Fundamentals of LSB insertion

While converting an analog image to digital format, usually a choice is made among three different ways of representing colors:

- 24-bit color: every pixel can have one in \(2^{24}\) colors, and these are represented as different quantities of three basic colors: red (R), green (G), blue (B), given by 8 bits (256 values) each.
- 8-bit color: every pixel can have one in \(2^8\) colors, chosen from a palette, or a table of colors.
- 8-bit gray-scale: every pixel can have one in \(2^8\) shades of gray.

Following example show how LSB insertion modifies the LSBs of each color in 24-bit images, or the LSBs of the 8-bit value for 8-bit images.

**Example:**

The letter 'C' has an ASCII code of 67 (in decimal), which is equivalent to 1000011 in binary. To store 7 bit ASCII value we need 7 pixels. In 24-bit images, it will need three successive pixels of each plane to store letter 'K'.

Let's say that the pixels before the insertion are:

```
01111101, 01111101, 10000101, ------
10001001, 10001001, 10001001, ------
11100010, 11100010, 11011111, ------
```

"Implementation & Analysis of Covert Data Hiding Techniques" 33
Then their values after the insertion of 'C' will be:

\[
\begin{array}{c}
01111100, 01111101, 10000101, \\
10001000, 10001001, \\
11000111, 11100010, 11011111
\end{array}
\]

(The values in bold are the ones that were modified by the transformation)

The same example for an 8-bit image would have needed 8 pixels:

\[
\begin{array}{c}
10100010, 10100001, 10100000, \\
10011111, 10011111, 10011111, \\
10011111, 10100000, 10100010, \\
10100010
\end{array}
\]

Then their values after the insertion of letter 'C' would have been:

\[
\begin{array}{c}
10100011, 10100000, 10100000, \\
10011110, 10011110, 10011111, \\
10011111, 10100000, 10100010, \\
10100010
\end{array}
\]

(Again, the values in bold are the ones that were modified by the transformation)

From these examples we can infer that 1-LSB insertion usually has a 50% chance to change a LSB every 8 bits, thus adding very little noise to the original picture.

For 24-bit images the modification can be extended sometimes to the second or even the third LSBs without being visible. 8-bit images instead have a much more limited space where to choose colors, so it's usually possible to change only the LSBs without the modification being detectable.
3.2.2 Data Rate (Capacity)

The most basic of LSBs insertion for 24-bit pictures inserts 3 bits/pixel. Since every pixel is 24 bits, we can hide

\[ \frac{3 \text{ hidden bits/pixel}}{24 \text{ data bits/pixel}} = \frac{1}{8} \text{ hidden bits/data bits} \]

So for this case we hide 1 bit of the embedded message for every 8 bits of the cover image. If we pushed the insertion to include the second LSBs, the formula would change to:

\[ \frac{6 \text{ hidden bits/pixel}}{24 \text{ data bits/pixel}} = \frac{2}{8} \text{ hidden bits/data bits} \]

And we would hide 2 bits of the embedded message for every 8 bits of the cover image. Adding a third-bit insertion, we would get:

\[ \frac{9 \text{ hidden bits/pixel}}{24 \text{ data bits/pixel}} = \frac{3}{8} \text{ hidden bits/data bits} \]

Acquiring a data rate of 3 embedded bits every 8 bits of the image.

The data rate for insertion in 8-bit images is analogous to the 1 LSB insertions in 24-bit images, or 1 embedded bit every 8 cover bits.

We can see the problem in another light, and ask how many cover bytes are needed to send an embedded byte. We need 8 bits (i.e. 1 byte) of cover image to hide 1 bit of message. We are converting every message character to its 7 bit ASCII value. So we need 7 bytes of cover image to hide single character of message.

3.2.3 Robustness

LSB insertion is very susceptible to a lot of transformations, even the most risk-free and standard ones.

Lossy compression, e.g. JPEG, is very likely to destroy it completely. The problem is that the "holes" in the Human Visual System that LSB insertion tries to exploit, a little sensitivity to added noise, are the similar to what lossy compression algorithms rely on to be able to reduce the data rate of images.

Geometrical transformations, moving the pixels around and especially displacing them from the original grid, are likely to destroy the embedded message, and the only one that could allow recovery is a simple translation.

Any other kind of picture transformation, like blurring or other effects, usually will destroy the hidden data. All in all, LSB insertion is a very little robust technique for data hiding.
**3.2.4 Peak signal to noise ratio**

PSNR is the peak signal-to-noise ratio (PSNR) between original and stego image. The PSNR computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is often used as a quality measurement between the original and a stego image. The higher the PSNR, the better the quality of the stego image [117].

To compute the PSNR, we need to first calculate the mean-squared error using the following equation:

\[
MSE = \frac{\sum [I_1(m,n) - I_2(m,n)]^2}{M \times N} \tag{3.1}
\]

In the above equation, \( M \) and \( N \) are the number of rows and columns in the input image, respectively. Then the block computes the PSNR using the following equation:

\[
PSNR = 10 \log_{10} \left( \frac{R^2}{MSE} \right) \tag{3.2}
\]

Where, \( R \) is the maximum fluctuation in the input image data type. For example, if the input image has a double-precision floating-point data type, then \( R \) is 1. If it has an 8-bit unsigned integer data type, \( R \) is 255, etc.

**3.2.5 Problems and possible solutions**

Having stated that LSB insertion is good for steganography, we can try to improve one of its major drawbacks: the ease of extraction. We don’t want that a malicious attacker be able to read everything we are sending.

This is usually accomplished with two complementary techniques:

- Encryption of the message, so that even if message is extracted, it does not make sense for them without decrypting the same.

- Randomizing the placement of the bits using random function (scattering), so that it’s almost impossible to rebuild the message without knowing the key used for the random placement.

In this way, the message is protected by two different keys and hence acquiring better confidentiality than before. This approach also protects the integrity of the message, being much more difficult to counterfeit the message. Anyway, since we don’t want our
message only to be an encrypted and scrambled message, but also have the purpose of making the communication hidden.

The two most important issues in this problem are:

- the choice of images
- the choice of the format (24-bit or 8-bit)

The cover image first of all must seem casual, so it must be chosen between a set of subjects that can have a reason to be exchanged between the source and the receiver. Then it must have quite varying colors, it must be "noisy", so that the added noise is going to be covered by the already present one.

Second, there is a problem with the file size, which involves the choice of the format. Unusually big files exchanged between two peers, in fact, are likely to arise suspicion.

For instance, what could be the size of a most common picture of 512x512 (262144 pixels) available on internet, with the different color representations:

- 24-bit color: 262144 pixels x 24 bits/pixel / 8 bits/byte = 786432 Bytes ~= 768 KB
- 8-bit color / grayscale (the occupancy is the same): 262144 pixels x 8 bits/pixel / 8 bits/byte = 262144 bytes ~= 256 KB

Looking at the size, we can see that a 24-bit uncompressed picture is of a quite uncommon size, because it's very strange that the sender didn't compress it, a practice that's widely used and wouldn't have worsened the image quality so much.

To solve this problem, it has been studied a modification to the JPEG algorithm that inserts LSBs in some of the lossless stages or pilots the rounding of the coefficients of the DCT used to compress the image to encode the bits.

Since we need to have small image file sizes, we should resort in using 8-bit images if we want to communicate using LSB insertion, because their size is more likely to be considered as normal. Most of the experts, anyway, advise to use 8-bit gray scale images, since their palette is much less varying than the color one, so LSB insertion is going to be very hard to detect by the human eye.

### 3.3 Frequency domain steganography

In frequency domain, images are first transformed and then the message is embedded in the image. When the data is embedded in frequency domain, the hidden data exist in more robust areas, spread across the entire image, and provides better resistance
against statistical attacks. There are many techniques used to transform image from spatial domain to frequency domain. The most common frequency domain method usually used in image processing is the 2D discrete cosine transform. In this technique the image is divided into 8×8 blocks and DCT transformation on each block is performed. DCT arranged the pixel of image according to their frequency value. The data bits are embedded in the low frequency coefficients of DCT.

As mentioned before, the DCT transforms an image from spatial domain to frequency domain. It decomposes the image signal into spectral sub-bands, each having different importance with respect to the image's visual quality. As a roughview, as shown in figure 3.1, DCT decomposes the image signal into low, middle, and high frequency components [45-50].

![Figure 3.1: Low, Middle, and High frequency distribution in a DCT block.](image)

The high-frequency components are better places to hide the secret data in than low-frequency components [50]. The first reason is that the high-frequency components often become zeros after quantization, and there is no need to change the values of the coefficients if the data to be embedded is zero. And the second reason is that high-frequency components are more visually resistant to noises than low-frequency components. Therefore, by following this method, we can reduce the quality degradation of the stego-image.
The human vision system is much more sensitive to the values in low-frequency components than those in the higher frequencies. Thus, distortion in high-frequency components is visually acceptable and perceptible. Therefore, the upper left values in the quantization table are small enough to avoid large alteration. In contrast, the lower right values in the table are large and can be altered.

3.3.1. Discrete Cosine Transform (DCT) and Quantization

A more complex way of hiding a secret message inside an image comes with the use and modifications of discrete cosine transformations. Discrete cosine transformations (DCT), are used by the JPEG compression algorithm to transform successive 8 x 8 pixel blocks of the image, into 64 DCT coefficients each. The plausibly of steganographic target formats increases with the amount of data transmitted in the respective format. JPEG images are widely used over internet and therefore they are an ideal target format for steganography. Each DCT coefficient \( F(u, v) \) of an 8 x 8 block of image pixels \( f(x, y) \) is given by:

\[
F(u, v) = \frac{1}{4} C(u) C(v) \left[ \sum_{x=0}^{7} \sum_{y=0}^{7} f(x, y) \cos \left( \frac{2x+1}{16} \pi u \right) \cos \left( \frac{2y+1}{16} \pi v \right) \right] \tag{3.3}
\]

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

After calculating the coefficients, the following quantizing operation is performed:

\[
F^Q(u, v) = \frac{F(u, v)}{Q(u, v)} \tag{3.4}
\]

where \( Q(u, v) \) is 64 (8 x 8) element quantization table.

The JPEG image format uses a discrete cosine transform (DCT) to transform successive 8x8 pixel blocks of the image into 64 DCT coefficients each. Equation 3.4 gives quantized DCT coefficients \( F^Q(u, v) \) for 8x8 pixel block.

The least significant bits (LSBs) of the quantized DCT coefficients are used as redundant bits into which the hidden message is embedded. In some image formats, like GIF, the visual structure of an image exists to some degree in all bit layers of the image. Steganographic systems that modify least significant bits of these image formats are often susceptible to visual attacks.
This is not true for the JPEG format. The modification of a single DCT coefficient affects all 64 image pixels. For that reason, there are no known visual attacks against the JPEG image format.

The DCT packs energy in the low frequency regions. Therefore, some of the high frequency content can be discarded without significant quality degradation. Such a quantization scheme causes further reduction in the entropy (or average number of bits per pixel). The attributes of the DCT have led to its wide spread deployment in virtually every image processing standard of the last decade, for example, JPEG [110].

### 3.3.2. Basic steps involve in DCT base algorithm

**Embedding Process:**
- As shown in figure 3.2, the cover image is divided into 8x8 blocks, then DCT is applied to each one & find DCT coefficient using equation 3.3.
- The DCT coefficients are quantized using the equation 3.4.
- The secret message (text, image, and audio) that we wish to embed within the cover image should be converted into a binary sequence, or more specifically, into bits sequence. This binary sequence is embedded in the middle and high frequency coefficients area of cover image block to get a stego-block.
- For each stego-block the inverse DCT will be taken to get the output blocks. Finally, output blocks will be put together to establish stego image.

![Diagram showing DCT base algorithm](image)

*Figure 3.2: Basic step of DCT base algorithm*
3.4 Implemented work of Steganography in this research work

Figure 3.3: Types of Steganography