CHAPTER 1

1. INTRODUCTION

1.1 Steganography

Steganography is an art of secret communications. Its main purpose is to hide the occurrence of communication over a public channel. In contrast to cryptography, steganography tends to hide the very existence of the message or any communication form, whereas cryptography aims is to conceal the content of the secret message. Hiding the occurrence of communication can be done by embedding a secret message into an innocent cover medium, such as an image, which no one else than the sender and the recipient can suspect.

Steganalysis is the art and science of detecting a secret communication. Hiding a message will most likely leave detectable traces in the cover medium. The information hiding process changes the statistical properties of the cover, which is a steganalyst attempts to detect. The process of attempting to detect statistical traces is called statistical steganalysis.

Nowadays the rapid growth of the Internet has lead to an extensive transfer/exchange of files such as images, video and data files. Exchanging, download and/or uploading files over the Internet have become familiar to a large percentage of the population because it is so easy and fast to do. There is a conversion from analogue to digital, such as digital images which are easily made and manipulated. This is where steganography takes place. Digital files can contain hidden messages and as it was stated above there is no control of how many or what kind of files are distributed through the Internet.

In 2001, US officials stated that they have suspicions that terrorists communicate using steganography in the internet.

"Hidden in the X-rated pictures on several pornographic Web sites and the posted comments on sports chat rooms may lie the encrypted blueprints of the next terrorist attack against the United States or its allies. It sounds farfetched, but U.S. officials and experts say it's the latest method of communication being used by Osama bin Laden and his associates to outfox law enforcement. Bin Laden, indicted in the bombing in 1998 of two U.S. embassies in East Africa, and others are
hiding maps and photographs of terrorist targets and posting instructions for terrorist activities on sports chat rooms, pornographic bulletin boards and other Web sites, U.S. and foreign officials say.”[18]

A lot of articles, like the above, were published at that time and brought the world’s attention on the use of steganography. Government officials, especially in the US, have shown a large interest on steganography and steganalysis research over the last years.

Using steganography one can easily make, for example, an image with a hidden message and send it over to the recipient without anyone knowing if it has a message embedded or not. On the other hand there is no clear evidence of widespread use of steganography but since it is simple to make a stego-file, it is assumed that anyone can really secretly communicate through digital files. In addition to this, there are over 300 stego-systems on the Internet so one can assume that people are using them for communicating through the Internet.

Throughout the last years a large number of stego-systems were developed, which are easy to obtain and use within the internet. Most of the stego-systems use as the cover medium images, and that will be the focus on this project. The message hiding process can be done to any type of images, such as BMP, GIF and JPEG images. The steganographic techniques and the attacks on them in this project will be based on JPEG images. JPEG is currently the most common format for images that are transmitted over email or generally used in the Internet.

Most of the embedding techniques that were developed in the last years have been successfully attacked. There are a lot of statistical properties that secure stego-systems should preserve and each time a more secure embedding algorithm is developed, steganalysts find a new statistic that they focus their attack on. A general methodology [9] used by steganalysts is discussed in the report and in addition to this a specific attack [8] based on this methodology is implemented. On the other hand, a new steganographic approach, (YASS) [25] was proposed in the 9th International Workshop on Information Hiding (11/06/2007), which is resistant to any up to date attacks.
1.2 Objectives of the project

- Overview of current steganographic and steganalysis techniques.
- Implement a specific steganalysis technique using Matlab, which was an attack on Outguess 0.2 stego-system. [8]
- Analyse the chosen technique experimentally.
- Evaluate the technique and compare with results on the literature.
- Provide a brief methodology on how to implement the blind feature-based steganalysis technique. [10]
- Analyse the new steganographic technique YASS [25] which appeared in the mid of June towards the end of the project.
- Propose ideas and directions on what can be done to attack YASS in the future.

1.3 Outline of the report

After the introduction on steganography, steganalysis and the aims of the dissertation, the next chapter provides details on the history of steganography, describes in more depth these two terms and discusses about their different types and scenarios. Chapter 3 gives a brief introduction to existing stego-systems, embedding techniques and the attacks that have been developed for them. A general methodology [4] for attacking stego-systems that work on JPEG images is discussed on chapter 4. In addition to this, the attack on Outguess [3] is introduced which is based on that methodology. The attack implementation results are evaluated and compared with the ones on the literature. In the last chapter, a brief methodology on feature-bases steganalysis technique [10] is given. YASS stego-system [25] is introduced and the results of the experiments done are given. Lastly, project evaluation, recommended future work and conclusions are described.
CHAPTER 2

2. THE ART OF STEGANOGRAPHY AND STEGANALYSIS

2.1 History of Steganography

Steganography is a Greek word which consists of στεγανός (steganos) and γραφή (graphi) and means covert writing. The first steganographic type of communication was first used from the ancient Greeks (430 BC). The Greek historian Herodotus describes the use of steganography in his book ‘The histories of Herodotus’ in two cases. First, Histiaeus the tyrant of Miletus in order to send a message shaved the head of his most trusted slave and tattooed the message on the slave’s head. When the hair grew back to normal, he sent the slave to the message destination. The other case of steganography reported by Herodotus was used by Demeratus who wrote a warning message by scratching it on a wooden writing tablet with the wax scrapped off, and then place a new wax on.

In recent history, more advance steganographic techniques were used. One of them was invisible ink, where messages were written with specific substances that would disappear after a while. The message would later be revealed by heating or other chemical reactions. Advances in photography produced more advanced techniques and were used for military purposes. A shrinking technique called microfilm was proposed by French photographer Dagron and was used during the Franco-Prussian War (1870-1871). Messages were shrunk to small dots and sent by pigeon carriers. This was the only way at the time for the provincial government in Tours to communicate with Paris, since pigeons could not carry paper dispatches. In Germany between World War I and World War II a microdot technique was used. The size of around one page of information that could contain a photograph as well was reduced to the size and shape of a typographical dot.

2.2 Steganographic System

A modern steganographic system (stego-system) [2] consists of a cover, the embedding algorithm, the secret message and a secret key. The cover can be an image, an audio file, a data
file or any digital file, in this project we will work on digital images, so from now on an image is going to always be the cover. The stego-system takes the cover image, the secret message and the key and produces a stego image using its embedding algorithm as shown in Figure 2.2.1. The stego image should have no visual differences from the cover and be indistinguishable from human eyes. The recipient of the stego image can easily do the inverse process using the same stego-system and key and retrieves the hidden message.

**Figure 2.2.1**

**Stego-System**

In a traditional steganographic set-up formulated as a prisoner’s problem by G. J. Simmons [26], Alice wants to send a message to Bob by hiding the information in a cover message. All communication between them goes through a warden. The goal of Alice and Bob is to find a method of communication (escape plan) that would allow them to send messages whilst the warden will be unaware of their existence. Different steganographic scenarios can be distinguished based on what information is available to the warden and what she can do to the messages:

**Pure steganography:** The warden might not be aware of the method that Alice and Bob are using and there is no key used in their method. This is called pure steganography [3] and it is the simplest form. However, Alice and Bob should not rely on the fact that the warden isn’t aware of
the method used, since if the warden can detect the presence of communication, it will be somehow easy to extract the message because the message can be hidden in as many ways as the number of embedding algorithms which exist.

**Secret-key steganography:** The warden has full knowledge of the communication method but does not know the secret key shared by Alice and Bob. This scenario complies with Kerckhoff’s principle. Thought, some researchers argue that this principle might be too strong for steganography since a complete knowledge of the stego-system could include a complete knowledge of the cover mediums, which may be very difficult for the warden to obtain. [2]

**Passive warden:** If the warden can detect the presence or absence of a communication but not modify the messages, the communication scenario is called the passive warden scenario.

**Active warden:** If the warden is able to make changes to the messages or extract the message, we talk about an active warden scenario.

A stego-system is considered to be better and more efficient the more statistical properties of images it can conserve. It becomes useless if one can identify the presence of a hidden image from a stego image. As in the early days of cryptography, a lot of different steganographic systems have been developed where every new one tends to be more secure than the previous one by being invulnerable to any known steganalysis attacks. The steganalyst in the attacks described in this project will act as a passive warden who inspects images.

### 2.3 Steganalysis

Steganalysis is the art and science of detecting hidden messages from images made from stego-systems. Steganalysis is a fast growing science and relatively new. Most steganalysis publications are written in the last 10 years. The purpose of steganalysis is to distinguish if an image contains a secret message or not. Researches on this field tend to find statistical properties of images that the stego-system doesn’t conserve or find methods that one can find out if the image was altered at all or not. Thus, steganalysis is considered successful if it can guess whether an image contains a hidden message or not with a probability higher than random guessing.
Steganalysis also attempts to find more information of the image and hidden message such as the type of embedding algorithm, the length of the message, the content of the message or the secret key used. A steganalysis attack can find any of the above and one can lead to another. For example, there are a large number of attacks implemented that approximate the message length and thus see if there is a message hidden.

In practice, a steganalyst is frequently interested in more than whether or not a secret message is present. The ultimate goal is to extract and decipher the secret message. However, in the absence of the knowledge of the stego technique and the stego and cipher keys altogether, this task may be extremely time consuming or completely infeasible. Therefore, any additional information, such as the message length or its approximate placement in image features, could prove very valuable to the analyst. From the research done on steganalysis, the basis of the attacks is to just be able to distinguish between a cover and a stego image. As, it was mentioned earlier, finding a stego image may lead to even extracting the message. Even if the exact stego system that was used to make the stego isn’t known, there are basically a few known steganographic embedding techniques that could have been used. A dictionary attack or other more efficient cryptanalysis attacks could be done in each system in order to find the secret message. Though, Neils Provos [23, 24] in 2001 used his steganalytic software StegDetect [21] in order to test a large sample of images that were downloaded using a web crawler from Usenet and eBay. He used a distributed dictionary attack [23, 24] on suspected stego images, which were a very small percentages of the images tested, and wasn’t able to find any secret messages.

A steganalyst apart from acting as a passive warden and maybe be able to extract and read the message, can also act as an active one and disable or destroy the message. As opposed to cryptanalysis where the cryptanalyst attempts to decrypt encrypted messages, the steganalyst attempts to detect the existence of the message.

Many steganalysis researchers such as Neil F. Johnson and S. Jajodia [16, 17] attempt to categorise steganalysis attacks to recover modify or remove the message, based on information available. With cryptography, comparison is made between any possible parts of the plaintext and parts of the ciphertext. In steganography, comparisons may be made between the cover, the stego image, and possible parts of the message. The message in a stego object may or may not be encrypted. If it is encrypted and the message is extracted, the cryptanalysis techniques may be applied.
In order to define attack techniques categories used for steganalysis, we need to consider the corresponding techniques in cryptanalysis. Attacks categories for cryptanalysis are *ciphertext-only, known plaintext, chosen plaintext*, and *chosen ciphertext*. [16, 17] In ciphertext-only attacks, the cryptanalyst only has the ciphertext to be decrypted. On the other hand, the cryptanalyst may have the encrypted message and portion of the decrypted message which may be used for a known plaintext attack. In chosen plaintext attack, the cryptanalyst has some ciphertext which corresponds to some selected plaintext. It is the most favourable case for the cryptanalyst since the latter can find the key or a pattern on how to decipher the rest of the plaintext. In the case where the encryption algorithm and ciphertext are available, the cryptanalyst can encrypt plaintext looking for exact matches in the ciphertext. This case is called chosen ciphertext attack and can be used to find the secret key used. The challenge with cryptanalysis and the difference from steganalysis challenge is not in detecting that something has been encrypted, but actually decrypt the message. Somewhat parallel attacks to extract the message are available to the steganalyst, but in the field of steganalysis because of the fact that we have more than just the ciphertext and plaintext. The difference in the challenge of steganalysis makes this categorisation harder to be done. These attacks can be categorized into 5 forms and it is assumed that the steganalyst always has the stego medium at least:

**Chosen Stego Attack**: In this scenario, the steganalyst is aware of the steganographic algorithm used to make the stego medium and attempts to make stego mediums from cover mediums in order to match the intercepted stego medium. This description is based on the chosen ciphertext attack but in the case of steganography it is more complicated to do. In theory, attempting to make new stego mediums to match the intercepted one sounds right, but in practice it is very hard to accomplish that since not only the cover medium is unknown but also the message embedded as well. So, accomplishing a match of the intercepted with the chosen stego is very hard to do in practice. A more realistic description of this attack could be that the steganalyst, since the steganographic algorithm and the stego medium are known, can make new stego images in order to derive a methodology to use for the specific algorithm and thus attack the intercepted medium. The above description is realistic since it is what some steganalysts do in the “real world” and not just in theory.

**Stego Only Attack**: In a stego-only attack the steganalyst does not have any other information available apart from the stego medium investigated. It is similar to the ciphertext only attack and
it is the hardest scenario for the cryptanalyst. Realistically, the only way a steganalyst would be able to attack it is by trying every possible known attacks on current steganographic algorithms.

**Known Cover Attack:** In a known cover attack apart from the stego medium, the original cover medium is also available. In this scenario, the steganalyst can find differences in the two mediums and hence attempt to find what kind of steganographic algorithm was used. This attack is similar to known plaintext attack.

**Known Message Attack:** A known message attack can be used when the hidden message is revealed. The steganalyst by knowing the hidden message can attempt to analyse the stego image for future attacks. Even by knowing the message, this may be very difficult and may even be considered equivalent to the stego-only attack.

The above categorization of steganalytic attacks is not often used since the main goal of steganalysis is to find the presence or the absence of a hidden message. Most of the existing steganalysis attacks were implemented by knowing the algorithm used, as Kerckhoffs’ principle implies, in order to derive a methodology by making stego images with known cover and thus compare their statistics. As it was mentioned before, the challenge of steganalysis is to first find the presence of secret communication. A less theoretical and more practical categorisation of attacks that is mainly used is the following. In the case of a known algorithm, an attack that works for that specific algorithm is called **targeted steganalysis** attack. In addition to this, there are steganalysis attacks that can apply on all or a selected set of steganographic algorithms and are called **blind** or **semi-blind** attacks.

A **targeted steganalysis** technique works on a specific type of stego-system and sometimes limited on image format. By studying and analysing the embedding algorithm, one can find image statistics that change after embedding. The results from most targeted steganalysis techniques are very accurate but on the other hand, the techniques are inflexible since most of the time there is no path to extend them to other embedding algorithms. Also, when a targeted steganalysis is successful, thus having a higher probability than random guessing, it helps the steganographic techniques to expand and become more secure.

A **blind steganalysis** technique is designed to work on all types of embedding techniques and image formats. In a few words, a blind steganalysis algorithm ‘learns’ the difference in the
statistical properties of pure and stego images and distinguish between them. The ‘learning’ process is done by training the machine on a large image database. Blind techniques are usually less accurate than targeted ones, but a lot more expandable.

**Semi-blind** steganalysis works on a specific range of different stego-systems. The range of the stego-systems can depend on the domain they embed on, i.e. spatial or transform.
CHAPTER 3

3. STEGANOGRAPHY AND STEGANALYSIS METHODS

3.1 Digital Image Formats

GIF (Graphics Interchange Format)

Graphics Interchange Format (GIF) was introduced as an image format by CompuServe in 1987. It uses a palette of up to 256 colours from the 24-bit RGB colour space, and stores both the palette and the pixel matrix. The palette is a table that associates each palette selection number with a specific RGB value. Each entry of the pixel matrix is the index of a colour in the palette. The restriction on the number of colours in the palette is what makes GIF format efficient only for low colour depth images. It is mainly used for simple images such as drawings, graphics or small logos. More information on GIF can be found on the GIF specification [30].

BMP (Bitmap)

Bitmap image format was a standard with Windows 3.0 or later. BMP images map the colour values of the image pixels to two dimensional arrays. Depending on the colour depth, images are generally stored as 1, 2, 8, 16 or 24-bit. Greyscale images have 256 colours and are stored as 8-bit.

JPEG (Joint Photographic Experts Group)

JPEG stands for Joint Photographic Experts Group, which was the committee that created the standard, was issued as a standard image compression method in 1992, it was approved as such in 1994 and it is the most commonly used standard for lossy compression until today. It is very efficient with photographic images and can make an excellent quality image despite the fact it is a lossy compression technique. By lossy compression it is meant that some visual quality is lost in the compression process. On the other hand, JPEG standard includes a lossless compression method as well, but we will focus on the lossy compression in this project. In order to understand JPEG, let us describe the steps of compression.
The first step is to convert the image from RGB colour space to YCbCr, where Y stands for the luminance and Cb, Cr stand for the chrominance of the image. Then, the chrominance information will get downsampled because of the fact that the human vision is less sensible on chrominance, although the downsampling step is optional. The luminance values in each block are subtracted (shifted) by 128. Thus, for an 8-bit image each pixel will be now in the range of $[-128,127]$. Dividing the image into 8x8 blocks is the next step. In the case of an image that does not divide evenly into 8x8 blocks, pixels will be added for the compression needs and will be discarded at the end. An 8x8 block of an image at this step, is shown in Figure 3.1.1.

**Figure 3.1.1**

**Shifted 8x8 block**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-72</td>
<td>-78</td>
<td>-65</td>
<td>-26</td>
<td>5</td>
<td>6</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>-74</td>
<td>-79</td>
<td>-57</td>
<td>-16</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>-77</td>
<td>-77</td>
<td>-44</td>
<td>-2</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>-78</td>
<td>-72</td>
<td>-32</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>-79</td>
<td>-65</td>
<td>-22</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>15</td>
<td>22</td>
</tr>
<tr>
<td>-78</td>
<td>-53</td>
<td>-13</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>31</td>
</tr>
<tr>
<td>-71</td>
<td>-37</td>
<td>-2</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>21</td>
<td>33</td>
</tr>
<tr>
<td>-63</td>
<td>-25</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>19</td>
<td>32</td>
</tr>
</tbody>
</table>

Next step is to convert each 8x8 block to a frequency-domain representation. This domain is also called the transform domain. The conversion is done using a normalised two-dimensional forward Discrete Cosine Transform (DCT) and the DCT equation is the following.

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

Where: $C(u), C(v) = \frac{1}{\sqrt{2}}$ for $u, v = 0$ ;

$C(u), C(v) = 1$ otherwise.

The results from this transform are floating-point numbers as shown in Figure 3.1.2. After the DCT transform, the block elements are changed to integers with quantization. The block elements
are divided by an integer based on an 8x8 quantization matrix which all the blocks of the specific image use, and are rounded to the nearest integer. The quantized block can be seen in Figure 3.1.3.

**Figure 3.1.2**

**DCT 8x8 block**

<table>
<thead>
<tr>
<th></th>
<th>-123.5000</th>
<th>-240.7616</th>
<th>-93.1346</th>
<th>-38.9828</th>
<th>24.2500</th>
<th>11.0152</th>
<th>-0.1179</th>
<th>0.4723</th>
</tr>
</thead>
<tbody>
<tr>
<td>-71.6520</td>
<td>-20.7505</td>
<td>-7.0538</td>
<td>56.8199</td>
<td>27.6432</td>
<td>0.2226</td>
<td>0.0328</td>
<td>0.1871</td>
<td></td>
</tr>
<tr>
<td>3.5206</td>
<td>10.8420</td>
<td>17.2353</td>
<td>12.2581</td>
<td>-10.0766</td>
<td>-14.6249</td>
<td>0.3536</td>
<td>0.1465</td>
<td></td>
</tr>
<tr>
<td>-7.3769</td>
<td>-8.3640</td>
<td>6.1741</td>
<td>0.5235</td>
<td>-1.3241</td>
<td>-0.0249</td>
<td>-0.4715</td>
<td>-0.1215</td>
<td></td>
</tr>
<tr>
<td>-0.2500</td>
<td>5.6918</td>
<td>-0.2300</td>
<td>0.1642</td>
<td>0.5000</td>
<td>0.1204</td>
<td>0.0957</td>
<td>-0.6612</td>
<td></td>
</tr>
<tr>
<td>-0.6334</td>
<td>0.7593</td>
<td>-0.2576</td>
<td>0.6284</td>
<td>0.2149</td>
<td>0.5204</td>
<td>0.0918</td>
<td>-0.8284</td>
<td></td>
</tr>
<tr>
<td>-0.6465</td>
<td>-0.0120</td>
<td>0.3536</td>
<td>0.5681</td>
<td>0.1557</td>
<td>0.2111</td>
<td>0.2647</td>
<td>-0.4056</td>
<td></td>
</tr>
<tr>
<td>0.2036</td>
<td>0.0835</td>
<td>-0.5705</td>
<td>0.2418</td>
<td>-0.4222</td>
<td>0.1821</td>
<td>0.0282</td>
<td>0.2474</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.1.3**

**Quantized and rounded 8x8 block**

<table>
<thead>
<tr>
<th></th>
<th>-15</th>
<th>-40</th>
<th>-19</th>
<th>-5</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-12</td>
<td>-12</td>
<td>-3</td>
<td>-1</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

We can see in Figure 3.1.3 that high frequency coefficients have turned to zero. Because of the fact that human vision can not see the difference in high frequency brightness the image will still have the same visible appearance. The (1,1) element in the table is called the DC coefficient and it is treated separately from the remaining 63 coefficients called the AC coefficients. Finally, all the coefficients, keeping the DC in the same spot are ordered into the ‘zig-zag’ sequence as shown in Figure 3.1.4. The ‘zig-zag’ ordering helps for the final step which is entropy coding. This is done with two entropy coding methods: Huffman coding and arithmetic coding. Entropy
coding will not be discussed further in this paper, since for the JPEG images that we analyse we only use the JPEG coefficients. More information on JPEG can be found on [27]. The decompression procedure is the exact reverse of the compression one.

Figure 3.1.4

Zig-Zag sequence

Jpeg is currently one of most common image formats used in the Internet. JPEG works well with photographic images due to the fact that colours are ‘true’ in contrast to GIF which is better for animations, drawings and computer graphics. We will focus more on this type of images, since a large number of the most recent and more efficient stego-system work on the transform domain. Many of the embedding methods and steganalysis techniques applied on them, are the same for all formats.

3.2 Embedding Methods

One of the major factors that steganography on images drew so much attention is the simplicity of the general embedding process, which is embedding on the least significant bit (LSB). This type of embedding is called sequential embedding. For example for a greyscale 8-bit bmp image, we can flip the LSBs of its pixel values with message bits. This is so easy and fast to implement and run, because it can be done in 1-2 lines of code. The same type of embedding can be done on all types of images. For JPEGs, we can simply flip the LSBs of the JPEG coefficients and for GIF do the same on the palette indexes for example.
Most of the early stego-systems algorithms were based on *sequential embedding*. Sequential embedding is very easy for someone to do or for someone to extract a message from a known stego image embedded sequentially. Since there is no key used for this embedding process, in the case that someone knows that a stego image was made with this technique he/she can simply read the message by just extracting the LSB plane. This type of embedding can be successfully attacked by both a visual attack and a statistical test.

Sequential embedding was improved by using a *pseudo-random* number generator (PRNG) to select the pixel values or coefficients at random. This pseudo-random type of embedding depends on the key provided by the user. Most stego-systems use pseudo-random embedding algorithms with small variations.

*Statistics-Aware embedding* includes any steganographic techniques that take into consideration statistics used in the past to attack any previous stego-systems. In most cases LSB flipping will still be the main part of the embedding algorithm but a few changes occur in the conditions of how the flipping is done. The LSBs values, for example, can either increment or decrement depending on the conditions of the specific algorithm. In addition to this, there are some stego-systems that make corrections on the image values in order to preserve certain statistics. In the following section we will view a few popular stego-systems and attacks on them in order to show how steganography and steganalysis advance as time passes.

### 3.3 Stego-Systems

**EzStego**

EzStego was developed by Romana Machado and embeds messages in GIF images. This system embedding is simple; it sorts the colour palette of the image and embeds sequentially in the LSBs of the pixels indexes. The palette is sorted in such a way that the neighbour colours are hardly to distinguish.
S-Tool

S-tool was developed by Andy Brown and it is a sequential embedding type of stego-system. It works on both BMP and GIF images and can also encrypt the hidden message with various encryption algorithms such as DES and IDEA. [15, 16]

JSteg

JSteg was developed by Derek Upham and it is a transform domain stego-system that sequentially embeds the message bits in the LSB of the JPEG coefficients excluding 0 and 1. [24] There is no key required, so anyone that knows that a stego is made using this system can extract the message.

Outguess

Outguess 0.1 was developed by Niels Provos [22] and improves the embedding algorithm of JSteg by using a PRNG in order to get the coefficients randomly. The LSB of the selected non-zero non-one JPEG coefficient is embedded with the message bit. Outguess was improved with a second version [20, 22] (Outguess 0.2) which preserves the first-order statistics of the image by making appropriate corrections after embedding. (Chapter 4)

F3, F4 and F5

F3 embedding process is different than any previous stego-systems in the way JPEG coefficients LSB flip. The absolute value of the coefficient will always decrease or will not change. Meaning that, for example, if the message bit to be embedded is 1 in a coefficient equal to 2 where its LSB is 0, 2 will not become 3 as usual but decrease to 1. In addition to this, only zeros are excluded in this algorithm, but if any new zero is created the last message bit embedded will be re-embedded.

F4 differs from F3 in the way the negative JPEG coefficients LSB change. The process is swapped meaning that if we want to embed a 0 in a coefficient equal to -5 the new value would not be -4 but will remain -5 and in the case of embedding a 1 in -5, -5 will change to -4.
F5 embedding algorithm [7, 29] is the same as the one F4 uses. In addition to this, F5 improves embedding efficiency by using matrix encoding. During the embedding, first the message length and the number of non-zero AC JPEG coefficients are calculated and then Hamming coding is used to embed $p$ bits into $2^p - 1$ pixels making 1 change maximum in the coefficients, where $p$ is the message block size. Thus matrix embedding achieves to minimize the number modifications of the cover image.

### 3.4 Steganalysis Attacks

#### Visual Attacks

Steganalysis by visual attack was used early in steganalysis research. The idea of visual attacks is to remove any parts of the image that cover the message in order for the human eye to distinguish where there is any hidden message or still image content. An example for sequential embedding can be to extract the LSB plane of the image and check for any possible suspicious structure in the image. The LSB plane of a natural greyscale bmp image can be seen in Figure 3.3.1, where it is clear that there are not any suspicious structures, while viewing the LSB plane of a stego made with sequential embedding we can see some sort of structure on the left-most part which can lead to further investigation in the image.

![Visual Attack](image)

**Figure 3.3.1**

**Visual Attack**

<table>
<thead>
<tr>
<th>Natural Image</th>
<th>Stego Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Natural Image" /></td>
<td><img src="image" alt="Stego Image" /></td>
</tr>
</tbody>
</table>
Another more technical way to make a visual attack is to apply specific filters on the image and compare it with a known natural image filtered with same filter. An example on this can be found on [28] by Andreas Westfeld and Andreas Pfitzmann, where the palette of a stego GIF image made with EzStego is replaced by a black and white palette such that the colours with an even index become black and the odd ones become white. This is done in order to prevent the LSB change to always be indistinguishable, because of the fact that neighbour colours are not similar anymore. We can see in Figure 3.3.2, that by using this filter an effective visual attack can be achieved.

**Figure 3.3.2**

**Filtered Visual Attack (Pictures taken from [28])**

Visual attacks in general work only for spatial domain embedding and are efficient against the first few stego-systems developed. They won’t work against any recent stego-system. In addition to this, visual attacks are unreliable in the sense that they are only applicable to certain type of images. Finally and most importantly they are not efficient because of the fact that they rely on human interaction and cannot be automated.
Statistical Attacks

Chi-Square Attack

Andreas Pfitzmann and Andreas Westfeld [28] introduced a method based on statistical analysis of Pair of Values (PoVs) that are exchanged during sequential embedding. This attack works on any sequential embedding type of stego-system such as EzStego and Jsteg. Sequential embedding makes PoVs in the values we embed in. For example, embedding in the spatial domain makes PoVs \( (2i, 2i+1) \) such that 0 ↔ 1, 2 ↔ 3, 4 ↔ 5, …, 252 ↔ 253, 254 ↔ 255. This will affect the histogram \( Y \) of the image’s pixel value \( k \), while the sum of \( Y_{2i} + Y_{2i+1} \) will remain unchanged. Thus the expected distribution of the sum of adjacent values is (2) and the \( \chi^2 \) value for the difference between distributions with \( v-1 \) degrees of freedom is (3). From (2) and (3) we get the \( \chi^2 \) statistic for our PoVs as in (4).

\[
E(Y_{2i}) = \frac{1}{2}(Y_{2i} + Y_{2i+1}) \quad (2)
\]

\[
\chi^2 = \sum_{i=1}^{v-1} \frac{(F - E(F))^2}{E(F)} \quad (3)
\]

\[
\chi^2_{PoV} = \sum_{i=1}^{127} \left( \frac{Y_{2i} - \frac{1}{2}(Y_{2i} + Y_{2i+1})}{\frac{1}{2}(Y_{2i} + Y_{2i+1})} \right)^2 \quad (4)
\]

The probability of embedding is determined by calculating the \( p \)-value (5) for a sample from the values examined, which starts at the beginning of the image and gets increased for each measurement. This attack though, does not work for pseudo-random type of embedding.
\[ p = \Pr(\chi^2_{k-1} \geq \chi^2_{PoV}) = 1 - \frac{1}{2^{k-1} \Gamma \left( \frac{k-1}{2} \right)} \int_0^{\chi^2_{PoV}} e^{-\frac{x}{2}} x^{k-1} \, dx, \tag{5} \]

where \( \Gamma \) is the Euler Gamma function.

**Generalised Chi-Square Attack**

The generalised \( \chi^2 \) attack [24] was implemented in order to detect data that are randomly distributed in an image. The difference from the \( \chi^2 \) attack is that instead of increasing the sample size for each measurement and starting the test at a constant position, the sample size is kept constant but the position changes by 1% for each measurement. The sum of the \( p \)-values for all measurements is calculated and compared to a detection threshold.

**Pairs Analysis**

The generalised \( \chi^2 \) attack does not calculate an estimation of the message length and can be sometimes wrong if the message has a significant difference in the number of zeros compared to ones. Pairs analysis was designed specifically for GIF images but works well for greyscale BMP images as well and can estimate the length of the message embedded. The method is based on colour cuts \( Z(c, c') \) where \( c \) and \( c' \) are two colour values. Scanning the image looking for those two colours we assign 0 to \( c \) and 1 to \( c' \). From all colour cuts we get two vectors \( Z \) and \( Z' \) where for greyscale images \( Z = Z(0,1) \& Z(2,3) \& ... \& Z(254,255) \) and \( Z' = Z(1,2) \& Z(3,4) \& ... \& Z(255,0) \). Next step we calculate the homogeneous pairs 00 and 11 in \( Z \). We define \( R(q) \) as the expected relative number of homogeneous pairs in \( Z \) after flipping \( q \% \) pixel values. \( R(q) \) is a parabola with its vertex at 0.5 and \( R(0.5) = 0.5 \). We can calculate \( R(q) \) by dividing the number of homogenous pairs in \( Z \) by the total number of pixels. In order to find \( q \) we need to approximate \( R'(q) \) by firstly calculating \( R'(0.5) \) (6). Since \( R(q) \) is a parabola with vertex at 0.5 we have that \( R(q) = R(1 - q) \). By accepting that \( R(0) \equiv R'(0) \), we can compute \( q \) from (7).
\[
R'(0.5) = \sum_{i=1}^{n-1} 2^{-i} h_k ,
\]

where \( h_k \) is the number of homogeneous pairs in the sequence \( a_ia_{i+1}, a_2a_{2+i}, \ldots, a_{n-k}a_n \) and \( Z' = a_ia_2\ldots a_n \).

\[
4D(0.5)q - 4D(0.5)q^2 = D(q) , \text{ where } D(x) = R(x) - R'(x)
\]

**Histogram Analysis Attack**

Histograms analysis attack works on JPEG sequential and pseudo-random embedding type stego-systems, such as JSteg and Outguess 0.1. It can effectively estimate the length of the message embedded and it is based on the loss of histogram symmetry after embedding. Since even positive and odd negative coefficients tend to decrease while odd positive and even negative tend to increase, a formula can be derived. We will denote the histogram values of the coefficients as \( H_i \). After embedding \( q\% \) length message, we get (8) and (9) for even and odd histogram values respectively.

\[
J_{2i} = \left( 1 - \frac{q}{2} \right) H_{2i} + \frac{q}{2} H_{2i+1}
\]

\[
J_{2i+1} = \frac{q}{2} H_{2i} + \left( 1 - \frac{q}{2} \right) H_{2i+1}
\]

Writing \( a = \frac{1 - q}{2} \), \( b = \frac{2}{1 - q} \) and solving for \( H_{2i} \) and \( H_{2i+1} \) we simplify (8) and (9). From histogram symmetry we know (10).

\[
\sum_{i>0} H_{2i} + \sum_{i<0} H_{2i+1} = \sum_{i<0} H_{2i} + \sum_{i>0} H_{2i+1}
\]

Substituting in (10) the simplified equations of (8) and (9) we can estimate the relative message length by using (11).
\[ q = 1 - \frac{\sum \Delta J_i}{J_1}, \]

where \( \Delta J_i = J_{2i} - J_{2i+1} \) for \( i > 0 \) and \( \Delta J_i = J_{2i+1} - J_{2i} \) for \( i < 0 \).
CHAPTER 4

4. ATTACKING THE OUTGUESS

4.1 Outguess

Outguess was proposed by Neils Provos [22] in order to counter the chi-square steganalysis attack which is effective for sequential message embedding. Outguess randomized the embedding process, while still skipping JPEG coefficients equal to 0 and 1. The message bits are embedded along a random walk that is determined by a pseudo-random number generator (PRNG) derived from a private key provided by the user. As most steganographic algorithms operating in the transform domain, Outguess flips the LSBs of the JPEG coefficients with the bits of the message and this process can be seen in Figure 4.1.1.

![Figure 4.1.1](image)

**Outguess Embedding Method**

<table>
<thead>
<tr>
<th>Original Jpeg Coefficients</th>
<th>-5</th>
<th>-4</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Bits</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Modified Jpeg Coefficients</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Though, this kind of embedding whether randomized or not still makes pairs of values. A generalised chi-square attack (Chapter 3.4) was suggested that could attack this type of embedding. Moreover, the symmetry of the histogram is destroyed and a histogram analysis technique (Chapter 3.4) can attack it efficiently as well. An example on how the symmetry is destroyed can be seen in figure 4.3.4.

To counter this, a second version of Outguess [20] was proposed by Neils Provos. The algorithm this time has two phases: one for the embedding process which is the same as the first version of
Outguess and one to make corrections in the histogram. So, Outguess0.2 is a statistics aware cryptographic system.

A priori estimate for the amount of data that can be hidden is being calculated at the first step, while it can still be able to preserve frequency count based statistics. Thus, not all of the redundant bits are used when embedding the hidden message. In fact, the selection process allows no more than half of the redundant bits to be used for data. If we know what kind of statistical tests are being used to examine an image for modification, we can use the remaining redundant bits to correct any statistical deviation that the embedding process created. [20] This is what Neils Provos based the second version of Outguess on.

4.2 Background

The method of the attack on Outguess that is described in this Chapter was pioneered and published by Jessica Fridrich et al [8].

A general methodology that is used for various steganalysis attacks is to find a macroscopic quantity $S(q)$ that predictably changes with respect to the relative length of the secret message embedded, where $q$ is the length of the message and $q \in [0,1]$, 0 for no message embedded and 1 for maximum length message. A definition of macroscopic quantity can be that it describes a quantity that is used for objects which can be measured and observed by the human eye. The term is mainly used in Physics and for our case, we can define it as a quantity that can be measured and can change with respect to another quantity. Such a macroscopic quantity can be, for example, the number of modifications in the DCT coefficients histogram of the image under investigation. Histogram coefficients change was used for an attack [5, 7, 9] on F5 and it was based on the fact that the number of zeros increases while the number for other coefficients tends to decrease due to the way F5 embeds (Chapter 3.3).

In order to determine how $S$ behaves, such as if it linear monotonically increases/decreases or exponentially increases/decreases, experiments must be done by changing the value of $q$. When the form of $S$ is known, some extreme values of our macroscopic quantity must be calculated from the stego image in order to find under which parameters it depends on. The most common extreme values are the value of $S$ for the cover image and the value of $S$ for the stego with
maximal length message embedded, $S(0)$ and $S(q_{\text{max}})$ respectively. By determining the parameters under which $S$ changes a formula to find $q$ can be derived. Once the parameters are known, we can estimate the message length $q$ by solving the equation $S(q) = S(\text{stego})$, where $S(\text{stego})$ is the value of $S$ for the stego we investigate. $S(q)$ will be called the distinguishing statistics.

In order to find the extreme values $S(0)$ and $S(q_{\text{max}})$ from the image investigated, in most cases it is needed to make two new images, one that will be used to calculate $S(0)$ which can be an estimation of the cover image and one to calculate $S(q_{\text{max}})$ which can be a stego image with maximal message embedded. Making a stego with maximal message embedded is a trivial task, since using the stego image we can investigate and embed on the latter a maximal length message, it overrides any previous embedding. In this targeted steganalysis technique based on Outguess 0.2, it is known that Outguess is overwriting the LSBs and as a result a second embedding will override any previous embedding. So, we can create a stego image with maximum embedding, which is similar to a stego that could be created from the original cover image with maximal embedded message. Thus, in order to find $S(q_{\text{max}})$ it is needed to find the maximum capacity of the image with respect to the same steganographic system and embed a message of that size in the image. Now we have a stego image with $q_{\text{max}}$ where $q_i = 1$ and we can use it to calculate our statistics.

On the other hand, estimating the cover image, in order to find the extreme value $S(0)$, requires a different methodology. Calculating $S(0)$ can be difficult since there is no access to the cover image used. A way to do this is to attempt to estimate the cover image. The estimated cover image should have many macroscopic properties close to the original cover in order to be useful. The estimation of the cover image is done by calibrating the stego image.

### 4.3 Calibration

In order to estimate the cover image we take into account how JPEG works. Based on the fact that JPEG images have a block structure of 8x8 blocks and are formed by quantized DCT coefficients, which tend to be robust to small distortions such as compression and embedding, we
can estimate the cover image. Thus, by decompressing and recompressing an image with different block structure we can estimate the cover image. This is done by using the following calibration technique on the stego image under investigation and can be seen in Figure 4.3.1:

1) Decompress the stego image using its quantization table.
2) Crop the decompressed stego image by 4 pixels, either column-wise or row-wise or at the edges.
3) Compress the cropped image using the same quantization table.

**Figure 4.3.1**

Calibration Technique

![Calibration Technique Diagram](image)

The new calculated DCT coefficients will not exhibit clusters due to quantization and because of the cropping by 4 pixels the standard JPEG 8x8 grid will not take into account any previous compression. Thus, the new DCT coefficients will not be influenced at all by any previous embedding and quantization. The calibrated stego image is perceptually similar to the original cover and thus the statistical properties of DCT coefficients are similar to those of the original image. We can see that this holds, by comparing the histogram of a cover image with the histogram of a calibrated stego image made using the original cover as in Figure 4.3.4.

To test that the calibrated image preserves the statistics of the cover, a stego was made using Jsteg. For this example the stego image has a message equal to 80% of the images maximum embedding capacity. We will view three histograms, one for the cover image, one for the stego image and one for the calibrated image. The two actual images used are shown in Figure 4.3.2.
As it was mentioned in Chapter 3, Jsteg embeds randomly in the LSB of the quantized JPEG coefficients, excluding 0 and 1. We can see how this affects the histogram by viewing the histograms of the AC JPEG coefficients of the cover (Figure 4.3.3) and the stego image (Figure 4.3.4).

Figure 4.3.3 gives a very nice and clear example of a JPEG image histogram, and we can clearly see the fact that the number of coefficients tend to symmetrically decrease from 0 on both negative and positive sides. Thus, the histogram is symmetric at 0 and any transform domain embedding on the image without histogram corrections will break this symmetry. Based on this, the histogram analysis attack (Chapter 3) was implemented for Jsteg that can estimate the length of the message.
The loss of symmetry can be clearly seen in Figure 4.3.4. In addition to this, because of the fact that Jsteg shifts the LSBs of the quantized DCT coefficients, excluding 0 and 1, there are a lot of cases where in a pair of value (2i, 2i+1) the even coefficient is larger than the odd one which is very rarely to see in a pure cover image.

Next step is to calibrate the stego image, by decompressing it, crop by 4 columns on both sides, left and right, and recompress it. The new image is perceptually similar to the original cover as it can be seen in Figure 4.3.4.

### Figure 4.3.4
**Original Cover and Estimated Cover Images**

<table>
<thead>
<tr>
<th>Baboon Cover Image</th>
<th>Calibrated Baboon Estimated Cover Image</th>
</tr>
</thead>
</table>

Furthermore, we plot the histogram of the AC coefficients for the calibrated images, which can be seen in Figure 4.3.5. It is clear that the symmetry of the histogram is preserved by calibrating the image, which can prove that by calibration one can preserve statistic properties of the cover image.
Figure 4.3.5
Histogram of the Calibrated Image

Figure 4.3.4
Comparison of the Original Cover and the Calibrated Image

| Cover Image Histogram | Calibrated Stego Image Histogram |
By comparing the two histograms, we can see that the histogram of the calibrated stego is very close to the cover’s histogram. Though, they have a few differences such as in the case of the number of -1 and 1 coefficients which is larger than the actual ones. This is not that bad since what we were looking for to preserve was the histogram symmetry. The basic point of this though, is not to see just that the histograms are similar to our eye, but to see how the calibrated image preserves the statistics of the cover which in this example was the histogram symmetry. It is not simple by just viewing the histograms to make sure the above statement holds. Thus, histogram analysis attack technique (Chapter 3.4) was used in order to attack the two images. The attack is based on how the coefficients behave after embedding, where the even positive and the odd negative ones tend to decrease while the odd positive and even negative tend to increase. By analysing the original symmetry and the behaviour of the coefficients based on the length of the message, a formula was made to estimate the relative message length. The output of the attack is the relative message length which is the percentage of embedding with respect to the maximum capacity of the image and is in the range of [0, 1]. The results of applying histogram analysis attack on the cover, stego and calibrated images are shown in Table 4.3.1.

<table>
<thead>
<tr>
<th></th>
<th>Actual Embedding Percentage</th>
<th>Histogram Analysis Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cover</td>
<td>0%</td>
<td>0.0109</td>
</tr>
<tr>
<td>Stego</td>
<td>80%</td>
<td>0.7958</td>
</tr>
<tr>
<td>Calibrated Stego</td>
<td>?</td>
<td>0.0049</td>
</tr>
</tbody>
</table>

The calibrated stego preserves statistics with respect to the histogram and the result of the histogram analysis technique shows that both the cover and the calibrated stego are non-stego images. This steganalysis technique is very accurate and it shows how a calibrated image can be used as an estimation of the cover image.

4.4 Attack Outguess

Outguess 0.1 embeds the message bits randomly in the LSBs of the quantized coefficients disregarding coefficients equal to 0 or 1. This first version of Outguess was successfully attacked by chi-square and histogram analysis attacks (Chapter 3.4). Outguess 0.2, on the other hand, in
addition to the embedding of the message bits the same way as the previous version it preserves the histogram statistics by making corrections to the coefficients. Thus, using any histogram statistics as the distinguishing statistic to attack it will not work with the second version of this steganographic tool. In order to attack it, it is needed to find something else that predictably changes with respect to the embedding percentage.

Based on the general methodology that was explained on the previous section, Outguess 0.2 can be attacked. The distinguishing statistic S that is used in this method is estimated with Blockiness.

**Definition 4.1**

*Blockiness B defines the sum of spatial discontinuities along the boundary of all 8x8 JPEG blocks and is calculated using the following formula:*

\[
B = \sum_{i=1}^{\frac{M-1}{8}} \sum_{j=1}^{N} |g_{8i,j} - g_{8i+1,j}| + \sum_{j=1}^{\frac{N-1}{8}} \sum_{i=1}^{M} |g_{i,8j} - g_{i,8j+1}|
\]

*where \( g_{i,j} \) is a pixel value in an \( M \times N \) grayscale image and \( \lfloor x \rfloor \) is the integer part of \( x \).*

In other words, Blockiness calculates the difference between the pixel values at the boundaries of each JPEG block. The differences of the pixel values are calculated for both column and row boundaries and the sum of those gives our Blockiness value. Since the blocks at the edges of an image do not have boundaries on all sides, the formula calculates the differences at the boundaries of blocks excluding the ones on the edges.

As we can see on Blockiness formula, rather than calculating the discontinuities along the boundaries of each 8x8 block at a time, it calculates the sum of the differences of each 8th row with its neighbouring one and the same for each 8th column row. Figures 4.4.1-2, highlight each 8th row as green with its neighbouring row as red and each 8th column as blue with its neighbouring column as yellow. Thus, Blockiness for an image is calculated by the sum of the absolute value of every blue column minus the neighbouring yellow column and adding up the sum of the absolute value of every green row minus the neighbouring red row.
Figure 4.4.1
Blockiness Illustration

Figure 4.4.2
Blockiness Illustration on a 16x16 Block of Greyscale Values

<table>
<thead>
<tr>
<th></th>
<th>84</th>
<th>99</th>
<th>116</th>
<th>125</th>
<th>132</th>
<th>137</th>
<th>135</th>
<th>128</th>
<th>129</th>
<th>130</th>
<th>127</th>
<th>123</th>
<th>124</th>
<th>128</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>86</td>
<td>106</td>
<td>118</td>
<td>123</td>
<td>127</td>
<td>127</td>
<td>124</td>
<td>126</td>
<td>129</td>
<td>128</td>
<td>125</td>
<td>126</td>
<td>129</td>
<td>128</td>
<td>124</td>
</tr>
<tr>
<td>38</td>
<td>59</td>
<td>86</td>
<td>104</td>
<td>115</td>
<td>124</td>
<td>130</td>
<td>133</td>
<td>126</td>
<td>130</td>
<td>131</td>
<td>129</td>
<td>128</td>
<td>130</td>
<td>129</td>
<td>124</td>
</tr>
<tr>
<td>26</td>
<td>42</td>
<td>65</td>
<td>87</td>
<td>104</td>
<td>120</td>
<td>131</td>
<td>138</td>
<td>132</td>
<td>135</td>
<td>136</td>
<td>132</td>
<td>129</td>
<td>130</td>
<td>129</td>
<td>126</td>
</tr>
<tr>
<td>30</td>
<td>36</td>
<td>48</td>
<td>66</td>
<td>86</td>
<td>104</td>
<td>120</td>
<td>129</td>
<td>136</td>
<td>140</td>
<td>139</td>
<td>133</td>
<td>129</td>
<td>130</td>
<td>130</td>
<td>129</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>30</td>
<td>45</td>
<td>64</td>
<td>84</td>
<td>106</td>
<td>122</td>
<td>133</td>
<td>138</td>
<td>140</td>
<td>134</td>
<td>130</td>
<td>130</td>
<td>131</td>
<td>130</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>26</td>
<td>34</td>
<td>43</td>
<td>58</td>
<td>85</td>
<td>109</td>
<td>121</td>
<td>130</td>
<td>137</td>
<td>136</td>
<td>132</td>
<td>132</td>
<td>131</td>
<td>129</td>
</tr>
<tr>
<td>29</td>
<td>33</td>
<td>37</td>
<td>55</td>
<td>28</td>
<td>33</td>
<td>59</td>
<td>87</td>
<td>110</td>
<td>123</td>
<td>135</td>
<td>137</td>
<td>135</td>
<td>133</td>
<td>131</td>
<td>127</td>
</tr>
<tr>
<td>24</td>
<td>19</td>
<td>21</td>
<td>43</td>
<td>47</td>
<td>48</td>
<td>50</td>
<td>80</td>
<td>103</td>
<td>121</td>
<td>125</td>
<td>127</td>
<td>134</td>
<td>136</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>19</td>
<td>23</td>
<td>34</td>
<td>43</td>
<td>44</td>
<td>42</td>
<td>41</td>
<td>57</td>
<td>81</td>
<td>106</td>
<td>118</td>
<td>128</td>
<td>137</td>
<td>138</td>
<td>132</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>20</td>
<td>30</td>
<td>36</td>
<td>32</td>
<td>26</td>
<td>24</td>
<td>31</td>
<td>55</td>
<td>84</td>
<td>105</td>
<td>122</td>
<td>135</td>
<td>137</td>
<td>131</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>19</td>
<td>27</td>
<td>29</td>
<td>24</td>
<td>18</td>
<td>15</td>
<td>19</td>
<td>37</td>
<td>60</td>
<td>82</td>
<td>104</td>
<td>122</td>
<td>130</td>
<td>129</td>
</tr>
<tr>
<td>27</td>
<td>24</td>
<td>24</td>
<td>29</td>
<td>29</td>
<td>24</td>
<td>21</td>
<td>21</td>
<td>16</td>
<td>25</td>
<td>39</td>
<td>56</td>
<td>78</td>
<td>102</td>
<td>120</td>
<td>129</td>
</tr>
<tr>
<td>28</td>
<td>25</td>
<td>24</td>
<td>26</td>
<td>25</td>
<td>22</td>
<td>21</td>
<td>21</td>
<td>16</td>
<td>18</td>
<td>23</td>
<td>33</td>
<td>52</td>
<td>79</td>
<td>106</td>
<td>123</td>
</tr>
<tr>
<td>22</td>
<td>20</td>
<td>21</td>
<td>23</td>
<td>22</td>
<td>19</td>
<td>20</td>
<td>24</td>
<td>19</td>
<td>18</td>
<td>18</td>
<td>21</td>
<td>33</td>
<td>55</td>
<td>83</td>
<td>104</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>22</td>
<td>26</td>
<td>26</td>
<td>22</td>
<td>23</td>
<td>26</td>
<td>22</td>
<td>21</td>
<td>19</td>
<td>18</td>
<td>23</td>
<td>38</td>
<td>64</td>
<td>85</td>
</tr>
</tbody>
</table>
Since we are only calculating the Blockiness for a 16x16 block in figure 4.4.2, it is only possible to calculate the difference between 8 and 9 columns and the difference of 8 and 9 rows. In a 16x16 image, all 8x8 blocks are at the edges of the image and thus we cannot calculate any discontinuities at the boundaries of the edges. The sum of both the differences, which is the Blockiness of this block, is 358.

In order to implement the Blockiness formula we calculate the sum of the pixel values differences for column and row boundaries one by one and add them at the end. The implementation to calculate each sum is straight-forward and we can see the Matlab code on figure 4.4.3.

**Figure 4.4.3**

**Blockiness Matlab Implementation**

```matlab
function [ B ] = Blockiness( image )

image=double(image);

M=size(image,1); %height of image
N=size(image,2); %width of image

B1=0; %initialize first sum
B2=0; %initialize second sum

for i=1:(M-1)/8 %for all possible 8x8 blocks row-wise
    temp=image(8*i,1:N); %put all elements of 8*i\(^{th}\) row in a vector
    temp2=image(8*i+1,1:N); %put all elements of 8*(i+1)\(^{th}\) row in a vector
    temp3=sum(abs(temp-temp2)); %sum of the absolute value difference %of the two vectors
    B1=B1+temp3; %left-hand side of the formula
End

for j=1:(N-1)/8 %for all possible 8x8 blocks row-wise
    temp=image(1:M,8*j); %put all elements of 8i\(^{th}\) column in a %vector
    temp2=image(1:M,8*j+1); %put all elements of 8*(i+1)\(^{th}\) column in a %vector
    temp3=sum(abs(temp-temp2)); %sum of the absolute value difference %of the two vectors
    B2=B2+temp3; %right-hand side of the formula
End

B=B1+B2; %add the two sums
```
4.5 Blockiness Analysis

In this section, we will discuss and analyse how Blockiness can be used to calculate the length of a message embedded in an image using Outguess 0.2 and finally show how the formula to calculate the message length $q$ is derived. For each statistic that is used to derive the formula we will provide some data and plot them in order to provide a proof.

First of all we need to find the form of our distinguishing statistic $S$, which in our case is found by Blockiness. As it was mentioned in Section 4.1 we need to calculate the Blockiness for two extreme values, in order to get $S$. The two extreme values are the Blockiness of the stego image and the Blockiness of the stego with a second embedding of maximal length message and are denoted as $B_{\text{stego}}(0)$ and $B_{\text{stego}}(q_{\text{max}})$ respectively. Then the distinguishing statistic $S_{\text{stego}}$ can be calculated from $B_{\text{stego}}(0)$ and $B_{\text{stego}}(q_{\text{max}})$: $S_{\text{stego}} = B_{\text{stego}}(q_{\text{max}}) - B_{\text{stego}}(0)$.

Blockiness $B(q)$ linearly increases with respect to $q$ and can be proved in the following plots (Figure 4.5.1-3) and so the form of $S$ is linearly increasing. In order to calculate $q$ we need to use and calculate our distinguishing statistic $S$ for three different images, which are made based on the image under investigation. We will treat each $S$ as a slope. The three slopes of $S$ we use are the following: slope $S_0$ for the calibrated cover image, slope $S_1$ for an image with maximal embedded message and slope $S_q$ for the stego image, where $S_q \in [S_1, S_0]$, which means that the slope for the stego should be in between slopes $S_1$ and $S_0$. This will be shown graphically later on this section. $S_0$ is what is expected for the original cover image, $S_1$ is what we get for a stego image with maximal message embedded and $S$ is for the stego under investigation with an unknown message length $q$.

In order to show experimentally how Blockiness behaves with respect to message length, and thus see that it increases linearly with the number of flipped coefficient LSBs, we will make three graphs, one for each slope. In addition to this, we will see how the three slopes look like. We will start with a cover image and embed on it, using Outguess 0.2, 10 times by increasing each time the message length. If our above statement holds the slope $S_0$ should be linear and increasing
with respect to message length. The data for Blockiness calculation for each image is shown in Table 4.5.1 and the plot for $S_0$ is shown in Figure 4.5.1.

### Table 4.5.1

Blockiness Data for Cover Image with Increasing Message Length Embedded

<table>
<thead>
<tr>
<th>Message Length $q$</th>
<th>Blockiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>230151</td>
</tr>
<tr>
<td>0.1</td>
<td>231966</td>
</tr>
<tr>
<td>0.21</td>
<td>233499</td>
</tr>
<tr>
<td>0.31</td>
<td>235399</td>
</tr>
<tr>
<td>0.4</td>
<td>236559</td>
</tr>
<tr>
<td>0.5</td>
<td>238819</td>
</tr>
<tr>
<td>0.61</td>
<td>239831</td>
</tr>
<tr>
<td>0.7</td>
<td>241546</td>
</tr>
<tr>
<td>0.81</td>
<td>243214</td>
</tr>
<tr>
<td>0.9</td>
<td>243967</td>
</tr>
<tr>
<td>1</td>
<td>246138</td>
</tr>
</tbody>
</table>

### Figure 4.5.1

Plot of Blockiness vs. Message Length from Table 4.5.1 Data
As it was expected, Blockiness is linearly increasing with respect to message length. Having more points for the plot would make the line even more linear and straight, but it was not needed in order to see if our assumption holds. Next step is to see how Blockiness behaves for $S_{stego}$. For this example, a stego image that was made to use for the above calculations will be used and follow with it the same procedure. We will make 10 new images by embedding on top of the stego using Outguess 2.0 again. The stego that is used is the one with 0.5 message length. We are expecting again that it will be linearly increasing with respect to $q$ and that $B_{stego}(q) > B_{cov}(q)$, $\forall q$. $B_{cov}(q)$ is the Blockiness of the cover image with a message of length $q$ embedded. The data for the Blockiness calculation for each new image made is shown in Table 4.5.2 and the plot for $S_{stego}$ is shown in Figure 4.5.2.

### Table 4.5.2

**Blockiness Data for Stego Image with Increasing Message Length Embedded**

<table>
<thead>
<tr>
<th>Message Length $q$</th>
<th>Blockiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>238819</td>
</tr>
<tr>
<td>0.1</td>
<td>239820</td>
</tr>
<tr>
<td>0.2</td>
<td>240440</td>
</tr>
<tr>
<td>0.3</td>
<td>241023</td>
</tr>
<tr>
<td>0.4</td>
<td>241548</td>
</tr>
<tr>
<td>0.5</td>
<td>242016</td>
</tr>
<tr>
<td>0.6</td>
<td>242953</td>
</tr>
<tr>
<td>0.7</td>
<td>244211</td>
</tr>
<tr>
<td>0.8</td>
<td>245344</td>
</tr>
<tr>
<td>0.9</td>
<td>246530</td>
</tr>
<tr>
<td>1</td>
<td>247388</td>
</tr>
</tbody>
</table>
$S$ is also linear and increases with respect to message length, while $B_{\text{stego}}(q) > B_{\text{cov}}(q)$ holds $\forall q$. Last we need to see how Blockiness behaves for $S_1$. $S_1$ is the slope for an image with a maximal length message embedded. Thus, our starting image this time will be the stego with maximal length embedded, that was made from the original cover in our first plot. We will embed, on top of that, new messages and will again make 10 new stego images by increasing the message length each time. Again we expect that the curve will still be linearly increasing and that $B_{\text{max}}(q) > B_{\text{stego}}(q) > B_{\text{cov}}(q), \forall q$. The data for the Blockiness calculation for each new image made is shown in Table 4.5.3 and the plot for $S_1$ is shown in Figure 4.5.3.
Table 4.5.3
Blockiness Data for Maximal Length Message Embedded Stego Image with Increasing Message Length Embedded

<table>
<thead>
<tr>
<th>Message Length q</th>
<th>Blockiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>246138</td>
</tr>
<tr>
<td>0.1</td>
<td>246234</td>
</tr>
<tr>
<td>0.2</td>
<td>246740</td>
</tr>
<tr>
<td>0.3</td>
<td>246497</td>
</tr>
<tr>
<td>0.4</td>
<td>247503</td>
</tr>
<tr>
<td>0.5</td>
<td>246850</td>
</tr>
<tr>
<td>0.61</td>
<td>247585</td>
</tr>
<tr>
<td>0.7</td>
<td>248151</td>
</tr>
<tr>
<td>0.8</td>
<td>247834</td>
</tr>
<tr>
<td>0.9</td>
<td>247695</td>
</tr>
<tr>
<td>1</td>
<td>248267</td>
</tr>
</tbody>
</table>

Figure 4.5.3
Plot of Blockiness vs. Message Length from Table 4.5.3 Data
The slope is almost linear and increasing with respect to message length. Since we embed on an image which already contains a maximal length message the slope is not as linear as we expected. We can assume that this happens because only a part of the message is overwritten each time and because of the randomized embedding process of Outguess there can be variations on the exact positions that new message was embedded and thus small variations on the Blockiness calculation. The difference in Blockiness though between the two extreme values is smaller than the previous two slopes. This is realistic since, as it was mentioned above, the image contained already a maximal length message.

In order to see, how a stego image made with Outguess 0.2 can be attacked using Blockiness as the distinguishing statistic and the other reason we plotted these three lines, we will start by plotting them into one graph which is shown in Figure 4.5.4.

**Figure 4.5.4**

*Plot of the Three Previous Blockiness Data vs. Message Length*

![Plot of the Three Previous Blockiness Data vs. Message Length](image)

We can clearly see from the above figure and by following the general formula $S = B(q_{\text{max}}) - B(0)$ that: $S_0 > S_{\text{stego}} > S_1$. Thus, $S$ is largest for the cover image, smallest for the maximal length embedded stego image and in between for the stego under investigation. A
more appropriate and accurate relationship for the three $S$ is $S_0 \geq S_{\text{stego}} \geq S_1$, if we take into account the fact that a cover or a maximal length message embedded stego image can be the image under investigation.

Lets denote the blockiness for the cover image $B_0(p)$, the blockiness of the stego image with message $q$ as $B_q(p)$, and the blockiness of the stego image with maximal message embedded as $B_1(p)$, where $p$ is in the range of our two extreme values zero and the value of the maximum possible message length that can be embedded in the image, thus we have $p \in [0, q_{\text{max}}]$, and $S_q = B_q(q_{\text{max}}) - B_q(0)$, $S_0 = B_0(q_{\text{max}}) - B_0(0)$ and $S_1 = B_1(q_{\text{max}}) - B_1(0)$. All this data can be calculated as it was mentioned above and from these we can find a formula to calculate $q$.

Now we need to find a formula in order to calculate the message length $q$ based on the three slopes. A formula for $q$ can be found using interpolation to our set of points. “Interpolation is any procedure for fitting a function to a set of points in such a manner that the function intercepts each of the points”. Our set of points $(q, S)$ are $(0, S_0)$, $(q, S_q)$ and $(1, S_1)$. In our case since the slopes are linear we use linear interpolation, which uses a linear polynomial as the interpolation function and its formula is (1) given the points $(x_0, y_0)$ and $(x_1, y_1)$. With our set of points we can see how the plot looks like in Figure 4.5.5. Looking at the graph and using (1) we can obtain (2) and by solving (2) for $q$ we get (3). This is our formula (3) to calculate $q$.

\[ y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} \]  

(1)
Figure 4.5.5

Linear Interpolation to the Known Points

\[ S_q = S_0 - q(S_0 - S_1) \]  \hspace{1cm} (2)

\[ q = \frac{S_0 - S_q}{S_0 - S_1} \]  \hspace{1cm} (3)
4.6 Detection Process

Summing up the above, the detection process for a stego image made using Outguess 0.2 consists of the following steps:

**Detection Algorithm**

1) Decompress the stego image and calculate its Blockiness \( B_q(0) \).
2) Embed the maximal length message in the stego using Outguess, decompress and calculate its Blockiness \( B_q(1) \).
3) Estimate the cover image, by calibrating the stego, and calculate its Blockiness \( B_0(0) \).
4) Embed the maximal length message in the estimated cover image using Outguess, and calculate its Blockiness \( B_0(1) \).
5) Using the stego image from the above step (Step 4) embed the maximal length message again using Outguess and calculate its Blockiness \( B_1(1) \).
6) Calculate the three slopes: \( S_q = B_q(1) - B_q(0) \), \( S_0 = B_0(1) - B_0(0) \), \( S_1 = B_1(1) - B_0(1) \).
7) Calculate the message length \( q \) using (3).

The detection algorithm contains randomization since we are making new images using Outguess. Outguess, as it was mentioned before, processes the image twice, once to embed the message bits along a random walk and once to make corrections to the coefficients to preserve the histogram. The key used determines how the embedding is done and thus a change on key can have better or worst results for our detection process. In order to be more accurate, the detection process was repeated 10 times for each image and the \( q \) values were averaged.

4.7 Results and Examples

We can see how the algorithm works to detect an image, and how \( q \) varies in the following example. The image tested for this example is a stego image made using Outguess 0.2 with 0.3546 message embedded. Tables 4.7.1a-b shows the five blockiness values calculated throughout the detection process, the calculated values of \( S_0 \), \( S_q \) and \( S_1 \) and finally the \( q \). The process was done 10 times.
Table 4.7.1a
1-5 Detection Algorithm Runs Results

<table>
<thead>
<tr>
<th>Variable</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_q(0)$</td>
<td>236455</td>
<td>236455</td>
<td>236455</td>
<td>236455</td>
<td>236455</td>
</tr>
<tr>
<td>$B_q(1)$</td>
<td>246704</td>
<td>247252</td>
<td>246801</td>
<td>246162</td>
<td>246594</td>
</tr>
<tr>
<td>$B_o(0)$</td>
<td>224038</td>
<td>224038</td>
<td>224038</td>
<td>224038</td>
<td>224038</td>
</tr>
<tr>
<td>$B_o(1)$</td>
<td>238797</td>
<td>239703</td>
<td>239231</td>
<td>239786</td>
<td>239874</td>
</tr>
<tr>
<td>$B_i(1)$</td>
<td>240548</td>
<td>241608</td>
<td>241306</td>
<td>241868</td>
<td>241637</td>
</tr>
<tr>
<td>$S_o$</td>
<td>14759</td>
<td>15665</td>
<td>15193</td>
<td>15748</td>
<td>14836</td>
</tr>
<tr>
<td>$S_i$</td>
<td>1751</td>
<td>1905</td>
<td>2075</td>
<td>2082</td>
<td>1763</td>
</tr>
<tr>
<td>$S_q$</td>
<td>10249</td>
<td>10797</td>
<td>10346</td>
<td>10507</td>
<td>10139</td>
</tr>
<tr>
<td>$q$</td>
<td>0.3467</td>
<td>0.3538</td>
<td>0.3694</td>
<td>0.3835</td>
<td>0.3529</td>
</tr>
</tbody>
</table>

Table 4.7.1b
6-10 Detection Algorithm Runs Results

<table>
<thead>
<tr>
<th>Variable</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_q(0)$</td>
<td>236455</td>
<td>236455</td>
<td>236455</td>
<td>236455</td>
<td>236455</td>
</tr>
<tr>
<td>$B_q(1)$</td>
<td>246796</td>
<td>247755</td>
<td>247123</td>
<td>246811</td>
<td>246745</td>
</tr>
<tr>
<td>$B_o(0)$</td>
<td>224038</td>
<td>224038</td>
<td>224038</td>
<td>224038</td>
<td>224038</td>
</tr>
<tr>
<td>$B_o(1)$</td>
<td>239326</td>
<td>239407</td>
<td>238834</td>
<td>238695</td>
<td>239034</td>
</tr>
<tr>
<td>$B_i(1)$</td>
<td>242127</td>
<td>240742</td>
<td>240458</td>
<td>240745</td>
<td>241032</td>
</tr>
<tr>
<td>$S_o$</td>
<td>15288</td>
<td>15369</td>
<td>14796</td>
<td>14657</td>
<td>14996</td>
</tr>
<tr>
<td>$S_i$</td>
<td>2801</td>
<td>1335</td>
<td>1624</td>
<td>2050</td>
<td>1998</td>
</tr>
<tr>
<td>$S_q$</td>
<td>10341</td>
<td>11050</td>
<td>10668</td>
<td>10356</td>
<td>10290</td>
</tr>
<tr>
<td>$q$</td>
<td>0.3961</td>
<td>0.3076</td>
<td>0.3133</td>
<td>0.3411</td>
<td>0.3620</td>
</tr>
</tbody>
</table>

Adding up all the estimated relative message lengths $q$ of the above tables and taking the average value we get as a result that $q=0.3526$. This is very close to the actual message length which was
0.3546. The error is in the range of 0.01 which is very small. On the other hand, we can clearly see a lot of information from this example. The value of \( q \) every time we repeat the process is different, as it was expected. In our above data \( q \) values are in the range of 0.30-0.39. Even if the result wasn’t that close to the actual, the range we get is accurate enough to say that the method is very efficient.

On the other hand, a large sample of images must be tested in order to be sure that the implementation is correct and efficient. Testing the algorithm on more images, including extreme cases such as stego images with maximal length message embedded and pure cover images will make a more clear evaluation of it.

A sample of 50 greyscale images has been tested, consisting of 25 pure cover images and 25 stego images with various message lengths made with Outguess 0.2. As in a few steganalysis attacks that estimate the message length, distinguishing between a pure cover image and an image with a small message embedded is not always trivial. In many cases the value of \( q \) is very close to zero, but in other cases the answer is in the range of 0.01-0.05. The possibility of getting a false positive for a cover image is relatively high. Almost one third of the cover images tested was found as stego with a small message embedded, while the possibility of a false negative was zero. In two cover images the length that was found was 0.1 which is way off the range of the other false positives we got. This happened only for two images so it is rare to get such a result. The results are pretty accurate and this proves that Blockiness is a strong statistic.

The actual relative message length that is used on the calculations is computed from the output of Outguess for every image. We calculate it by dividing the message size with the correctable message size which is what Outguess calculates as the maximum capacity for an image. The result of the testing of 50 images is shown in the Figure 4.6.1.
In order to test if this technique can work on other transform domain stego-systems, it was tested on Jsteg as well. This was done to see if the technique was only suited for Outguess 0.2 because of the fact that the latter makes corrections on the histogram values. As it was expected, the method can be applied on simpler pseudo-random embedding algorithms as well. The results of 50 images made with Jsteg were similar to the ones of Outguess. The difference this time was that the maximum message length that could be embedded on an image should be calculated as well, in contrast to Outguess where it could be calculated by the output of the system. The maximum message length for Jsteg was the number of JPEG coefficients not equal to 0 or 1. Again the results for cover images are not always correct for the same reasons. Depending on the image tested we can have a difference between actual and estimated message length of around ±0.02 to 0.13 on extreme cases. In order to test it on Jsteg, an algorithm that does the embedding process of Jsteg was used, in order to automate the procedure. The sample this time was 10 images and for each one 4 stego images were made with random relative message length. The results are shown on Figure 4.6.2.
Figure 4.6.2

Results of Testing 50 images on JSteg

Plot of Relative Message Length vs. the Number of Image Tested

4.8 Evaluation.

From Figure 4.6.1 it can be clearly seen how accurate the method is. For stego images with messages embedded of a relative length larger than 0.1 the method is very accurate. As it was mentioned above, the only inaccurate thing about the implementation of the attack is that it has a relatively high chance to get false positives. An explanation of this could probably be that in some images the Blockiness of the cover image varies by a small but considerable margin from the Blockiness of the calibrated image. Since we approximate the cover image by calibration, \( B_q(0) \) for a cover image can be larger than \( B_0(0) \) for the calibrated image and the same for \( B_q(1) \) and \( B_0(1) \). The problem though is that if the difference of \( B_q(1) \) and \( B_0(0) \) is larger than the difference of \( B_q(1) \) and \( B_q(0) \), \( S_0 - S_q \) will be larger than 0 and depending on how much larger \( S_0 \) is from \( S_q \), the result will be more inaccurate. Again, as it was mentioned above a problem with many steganalysis techniques is that there can be a trade-off, considering the threshold, of being able to more accurately detect natural images with the accuracy on detecting
stego images with small message embedded. More specifically in this technique, a threshold could be set in the algorithm where if the answer is in the range of 0.01-0.05 the image could be considered cover than stego. On the other hand though, in my opinion, a high goal for steganalysis techniques should be the accurate detection of stego images with small messages and the ability to distinguish between the later and a natural image. This is because of the fact that, it is not likely for someone who wants to communicate by embedding the message on an image to embed a very large message. The message many times can be small and that is what is more important.

On the other hand, comparing the results of the implementation with the results from the literature [6] which are shown on Figure 4.8.1 we can see that the accuracy of detecting stego images relative length is similar. Though, it seems that the results in the paper show that the percentage of inaccurate detection on natural images is smaller. Only a 10% of the natural images tested were detected as stego. An explanation on the difference of this project’s results and the ones from the paper can be based on the type of images used. This could include the size of the images the quality factor that they were compressed with or the image structure.

**Figure 4.8.1**

**Results of Testing 70 images (taken from [8])**

**Plot of Relative Message Length vs. the Number of Image Tested**
CHAPTER 5

YET ANOTHER STEGANOGRAPHIC SCHEME

5.1 Introduction

The main advantage of the general methodology described on Chapter 4 is that an estimation of the cover image can be achieved by calibrating the stego image. The statistics of the cover image are approximated and thus, by analysis and comparison of the stego and the calibrated image, the image under investigation can be successfully attacked. All stego-systems that operate on the transform domain follow the standard JPEG 8 x 8 block grid and with calibration, steganalysts use that for their own advantage and make a new image by reconstructing it with a different block structure.

Over the last years, a lot of stego systems have been proposed, where each one tries to preserve more statistics than the previous ones. However, steganalysis research seems to have the upper hand at the moment since all stego-systems have been successfully attacked. Stating this might be a little risky to, but from any research that i have done, we were not able to find a current stego-system that resists all attacks; some stego-systems are able to resist a couple of attacks but are never resistant to all. The ability to approximate the cover image by calibration gives steganalysts a lot of information to take advantage off.

A new steganographic scheme [25] has been proposed and developed by K. Solanki, A. Sarkar and B. S. Manjunath, called Yet Another Steganographic Scheme (YASS), which was presented on 11th of June 2007 at the 9th Information Hiding Conference in France. The biggest advantage of this system is that it is the first up to date stego-system that can resist any recent blind steganalysis techniques. It is fairly new so that someone can not be absolutely sure that it will not be successfully attacked soon. The results in [25] show that the stego-system cannot be detected by the most recent and accurate blind steganalysis attacks.

Throughout the last years a number of blind steganalysis algorithms have been proposed and implemented. The main feature of a blind steganalysis technique is to devise a blind detector to inspect images and distinguish between a natural image and a stego image. The blind detector
‘learns’ what a natural unmodified image looks like by calculating features, can detect any embedding method and can even find what embedding technique was used. Features can be either first-order statistics, such as statistics calculated from the histogram, or higher-order such as Blockiness for example. A classifier needs to be trained on cover and stego images features in order to distinguish between them. Blind steganalysis methods that have been proposed and developed so far use different type of classifiers. Specific classifiers can out-perform other classifiers with the drawback of being slower and harder to implement.

One of the first blind steganalysis techniques was pioneered by Nasir Memon who used a set of Image Quality Measures [19] in order to train the classifier. Hany Farid [5] suggested a technique based on features calculated in the wavelet decomposition of the stego image. He observed that there exist strong higher-order statistical regularities within the wavelet-like decomposition in natural images which alter by embedding a message. Using Fischer Linear Discriminant analysis (FLD) he managed to classify these alterations. Using the same features H. Farid and Siwei Lyu [6] changed the classification method and used Support Vector Machines classifier in order to make it more efficient and flexible. Lastly, J. Fridrich et al. [10] proposed a blind technique based on DCT-features and calibration. This technique will be described in the next section and a brief methodology on how to implement such a blind steganalysis technique will be explained. The choice of this specific method was made because of the fact that it is based on calibration, DCT-features and other higher-order statistics such as Blockiness, which were the basis of the methodology described and analysed in Chapter 4.

5.2 Feature-Based Steganalysis by J.Fridrich et al

The main idea of the blind feature-based steganalysis technique [10], proposed by Fridrich et al, is that the blind detector is constructed by using calibrated features computed directly in the DCT domain. Each feature is calculated for both the stego and the calibrated image, and the final feature is obtained by comparing these two. This comparison is done by calculating the $L_1$ norm of the difference of the two features. The $L_1$ norm is defined for a vector or matrix as the sum of absolute values of all vector or matrix elements. We will call all features used as vector functionals and denote a vector functional from the stego image as $F$ and the one from the calibrated image as $F'$. Thus, the final feature will be obtained by (1).
\[ f = \left\| F - F' \right\|_{L_1} \]  

(1)

**First Order Statistics**

The first order statistic in the DCT domain that is used in this technique is the JPEG coefficients histogram. The first vector functional that is calculated is the *global histogram*. The global histogram is denoted as \( h_r \) where \( r \) is in the range from the minimum to the maximum value of the coefficients in the image. Thus, \( h_r \) is the number of JPEG coefficients equal to \( r \) divided by the number of all coefficients. The global histogram of the calibrated image is denoted as \( h'_r \). The final feature is obtained by (2).

\[
f_1 = \sum_{r_{\text{min}}}^{r_{\text{max}}} \left\| h_r - h'_r \right\|
\]

(2)

In order to be more accurate and efficient for any stego-systems that preserve the global histogram, *individual histograms* for low frequency coefficients are added to the set of functionals. Individual histograms for coefficients in a fixed \((i, j)\) position on every 8 x 8 block will be denoted as \( h_{ij} \). The number of functionals for individual histograms used in this technique is 5; one for each of the following positions: (1,2), (1,3), (2,1), (2,2), (3,1).

Finally, *dual histograms* are added as well to the set of functionals. Dual histogram is the number of how many times a coefficient value \( d \) occurs in the position \((i, j)\) over all 8 x 8 blocks in the image. It is denoted as \( g_{ij}^d \) and is an 8 x 8 matrix derived from (3). 11 dual histograms are used, one for each of the following values: \( d = -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5 \).

\[
g_{ij}^d = \delta(d, d_k(i, j)), \quad (3)
\]

where \( \delta(u, v) = 1 \text{ if } u = v \text{ and } 0 \text{ otherwise.} \)
So, this specific blind technique uses 17 functionals obtained from first order statistics. The rest of the functionals are obtained by second order statistics such as Blockiness, Variation and Co-occurrence Matrices.

**Second Order Statistics**

As it was described in Chapter 4, embedding in an image is likely to increase the discontinuities along the block boundaries. The difference in the formula of Blockiness used in this technique is that we divide the Blockiness of the image with the number of all values used in the calculation. For example, for a 32 x 32 image the Blockiness calculates the difference of 192 values. To be more specific it calculates the difference of the values in the 8th and 9th, 16th and 17th, 24th and 25th row and column, where each row or column consists of 32 elements. The total number of values for a M x N image can be calculated by (4). Two Blockiness measures \( B_a \) are included in the functionals for \( a = 1,2 \). Thus the formula for \( B_a \) is (5).

\[
B_a = \frac{N}{8} \left( \frac{M-1}{8} + M \frac{N-1}{8} \right)
\]

\[
N \sum_{i=1}^{M-1} \sum_{j=1}^{N} |g_{8i,j} - g_{8i+1,j}|^a + \sum_{j=1}^{N-1} \sum_{i=1}^{M} |g_{i,8j} - g_{i,8j+1}|^a
\]

Another inter-block dependency that changes when a message is embedded in the image is variation \( V \). Variation calculates the differences of coefficients in the same position in consecutive blocks first row-wise and then column-wise. The vectors of block indices are denoted as \( I_r \) and \( I_c \), while scanning the image by rows and by columns respectively and the exact formula to calculate \( V \) can be found on [10].

The last three functionals are calculated by the co-occurrence matrix of neighboring JPEG coefficients. The co-occurrence matrix is defined as the probability distribution of neighboring
JPEG coefficients and denoted as $C_{sr}$. The three functionals obtained from the JPEG image $J$ and the calibrated one $J'$ are:

1) $C_{0,0}(J) - C_{0,0}(J')$
2) $C_{0,1}(J) - C_{0,1}(J') + C_{1,0}(J) - C_{1,0}(J') + C_{-1,0}(J) - C_{-1,0}(J') + C_{0,-1}(J) - C_{0,-1}(J')$
3) $C_{1,1}(J) - C_{1,1}(J') + C_{1,-1}(J) - C_{1,-1}(J') + C_{-1,1}(J) - C_{-1,1}(J') + C_{-1,-1}(J) - C_{-1,-1}(J')$

The exact equations for calculating $C_{sr}$ as well as $V$, as it was mentioned above, can be found on [23]. More details on these were avoided since this is an overview of the methodology and explaining in more depth all the functionals would need a different approach.

**Classifier**

In order to train the classifier to be able to distinguish between a natural and a stego image a specific classification technique should be chosen. A more straightforward approach is to use the **Fisher Linear Discriminant** (FLD) classifier and train it to a large sample (training set) of natural images and stego images made with various different transform domain stego-systems. A classifier considers a set of observations, which in our cases is the functionals, for each sample with known class (natural or stego image). In a graphical point of view, FLD finds projections to a line such that samples from different classes are well separated. In order to calculate the projections, the result from applying FLD to the training set will be an eigenvector.

Using the projection a **Receiver Operating Characteristic** (ROC) curve can be calculated. ROC curves are generally used in decision-making projects. In statistics and in our case they are used in order to find a threshold and calculate detection accuracy or reliability by plotting the fraction of true positives versus the fraction of false positives. Again more details in the concepts of ROC curves and FLD were avoided in this project. Finally to sum up the above the classification phase consists of the following steps:
1) Gather a large sample of natural images and preferably resize them to the same dimensions and compress them with the same quantization quality.
2) Use the sample on the stego-systems of your choice and make a second sample of stego images.
3) Calculate the functionals from the two set of samples.
4) Apply FLD on the two sets of data, natural and stego images and find the generalized eigenvector.
5) Calculate the projections of the functionals on the eigenvector.
6) Calculate the ROC from the projections.

5.3 Introduction to YASS

YASS is a new approach on stego-systems and it is the only one known that can resist blind steganalysis techniques such as the one discussed above. The main idea behind YASS is the fact that it desynchronizes the steganalyst by embedding in random locations. By random locations it is meant that not only it selects random coefficients to embed like previous stego-systems but 8x8 blocks out of the basic JPEG grid are randomly selected. The advantage of this is that it makes it difficult for someone to estimate the cover image and thus its features by calibration.

As it was discussed in [25] and also proposed by J. Fridrich et al in [10] future steganographic schemes should take into consideration the key ingredients that contribute to the high accuracy of blind steganalysis techniques. The calibration process gives to the steganalyst a big advantage, since it provides an accurate estimation of the cover image. Calibrated statistics should be added to the set of preserved statistics. In addition to this, powerful machine learning with a large sample of images allows the machine to ‘learn’ even the slightest statistical feature variation. So, every single statistic that is used to identify stego images should be conserved.

In the proposal of YASS [25], it is mentioned that steganography researchers instead of trying to preserve all the statistics they can embed data in such a way that it does not allow the steganalyst to estimate the cover image statistics. This can be achieved by randomizing the embedding method. Randomization can include the choice of transform domain, the embedding method or the transform coefficient to hide. As it was mentioned earlier the choice for YASS was to randomize the embedding locations within the image.
5.4 YASS Embedding Method

YASS embeds data in 8x8 blocks whose locations are chosen randomly from a larger size of blocks. For a grayscale M x N image the steps of the randomized block structure hiding method of YASS is the following:

1) The image is divided into blocks of size B x B and B is called big block size and is always greater than the size of a standard JPEG block. Thus, the image is divided into $M_B \times N_B$ big blocks, where $M_B = \left\lfloor \frac{M}{B} \right\rfloor$ and $N_B = \left\lfloor \frac{N}{B} \right\rfloor$.
2) For each big block an 8 x 8 sub-block is chosen by pseudo-randomly choosing a position in the big block which is used as the starting point of the sub-block. These sub-blocks are used to embed data in.
3) For every sub-block its JPEG coefficients are computed with a design quality factor $QF_h$. The number of low frequency AC coefficients to be used for embedding is predetermined.
4) The image is then compressed using a quality factor $QF_a \geq QF_h$.

The parameters that YASS takes as input are the two quality factors $QF_a$ and $QF_h$, the number of AC coefficients that are used for embedding in each sub-block and the size of the big block. The source that is available for YASS at the moment is Matlab code, uses a random message and a specific seed for the PRNG without the need for the user to provide. Since the code is available, the functions can be altered so that testing can be done with different keys and messages embedded.

5.5 YASS Testing and Evaluation

The fact that the embedding of YASS is a lot different than any other stego-systems makes all of the steganalysis attacks we have been studying incapable to detect it. In addition to this, in the proposal of YASS [25] the stego-system is evaluated against six blind steganalysis attacks such as Farid’s [6] wavelet decomposition attack, Fridrich’s [10] feature based attack and others. The
result of the evaluation process is that the detection was random, thus the stego-system can not be attacked with those techniques.

In order to look in more depth on how someone can attack YASS we will discuss the advantages of YASS with respect to the methodology we described on Chapter 4. As it mentioned before, because of the random location technique of YASS, calibrating a stego image made with this stego-system will not have the same results as other stego-systems such as Outguess. YASS divides the image in big blocks and selects 8x8 blocks from the latter. Then it embeds data in the JPEG coefficients of that spatial block. Thus, the cover image cannot be approximated by calibrating the stego image. This is the first important characteristic.

In addition to this, it will be difficult for someone to calculate the maximum capacity of an image for YASS. The maximum capacity is based on the inputs of the user and more specifically on the number of AC coefficients that can be used for embedding in every 8x8 block and on the big block size. Increasing the number of AC coefficients will increase the maximum capacity and increasing the big block size will decrease the capacity since the number of 8x8 blocks to be used for embedding will decrease. Furthermore, as it is stated on the proposal of YASS [25] the number of bits that can be embedded increases as $QF_a$ increases and $QF_a > QF_h$. For example, moving from $QF_a$ and $QF_h$ equal to 50 to $QF_a$ equal to 75 and keeping $QF_h$ the same, the maximum capacity can be 3 to 10 larger than with equal quality factors depending on the image. To sum up the maximum capacity can depend on the quality factors, the number of coefficients used and the size of the big block. All are unknown except from $QF_a$ which is known since it’s the quality factor that is used to compress the image at the final step of embedding.

In order to see what can be done to attack YASS we will first test Blockiness and see how it behaves with respect to the number of bits hidden. To do this, the size of the big block was decreased for each new test starting from 21 to 9 while keeping the number of coefficients used in each 8x8 block constant. The results are shown in Figure 5.5.1 and we can see that Blockiness increases with respect to the message length for YASS as well, but the curve is not as linear as it was for Outguess.
Although Blockiness tends to increase, we can see that there is a large increase from the Blockiness of the cover to the Blockiness of a stego with just 1000 bits embedded. Blockiness for the rest of the stego images does not increase by such a large margin, but it generally still tends to increase. Furthermore, YASS was tested in order to make sure that an estimation of the cover image from a stego is impossible. Again, the same set of stego images is used and the Blockiness with respect to Number of Hidden Bits of the stego image is plotted and shown in Figure 5.5.2. The actual Blockiness of the cover is shown as well in the graph, and we can see that it tends to increase for the calibrated stego images as well. On the other hand we can see that it is closer to the actual Blockiness of the cover than the one of the stego images. Though, making a comparison does not lead to any useful results.
The only useful thing to notice is that Blockiness actually increases with respect to message bits hidden. But how can someone use this information to attack YASS? Looking at the tables in [25] we can see that the detection accuracy of Fridrics' and Pennys' [10, 14] blind steganalysis techniques is higher than other techniques. An explanation of this can be that they are the only attacks that use features capturing inter-block dependencies such as Blockiness and Variation. On the other hand, both of these features are based on the 8 x 8 grid which is not the case in YASS, but as we seen in Figure 5.5.1 the fact that Blockiness still increases might be still useful.

The way we interpret these observations is that there might be a way to actually manage to calculate the size of the big block used to make the stego. For example, an idea is to calculate Blockiness for every possible 8 x 8 block in an image. In order to calculate Blockiness in given positions in the image we make 16x16 blocks by moving the position of the starting point of the block by 1. The pseudo-code for this procedure can be seen in Figure 5.5.3.
Using the above algorithm for a stego image and the cover image that is made from, we plot their three dimensional plots. What we are looking to find is any edges that only occur on the stego image and generally see if we can find any differences that are somehow structured in order to possibly approximate the size of the big block used in the stego or other information that could be useful. The plot of the cover image is shown in Figure 5.5.4 and the one of the stego image in Figure 5.5.5.
Figure 5.5.4
3-D Plot of the Blockiness vs. Upper Left Position (x, y) of the 16x16 Block
Cover Image

Figure 5.5.5
3-D Plot of the Blockiness vs. Upper Left Position (x, y) of the 16x16 Block
Stego Image
Looking at the two plots we can see that the stego has a few higher edges but there is no actual structure in order to find a pattern and estimate the big block size. However, it might be possible to find something more useful by following specific statistics on the two set of data. The time was limited on the analysis of YASS and thus I was not able to find more details. In addition to this, the same algorithm was tested on different images and different size of blocks the image is divided to. In figure 5.5.6 we can see the 3-D plots of the same two images with block sizes this time equal to 32. Again there is no difference that can be seen visually, though we can see that the stego images Blockiness edges are higher than the ones of the cover.

Figure 5.5.6
3-D Plot of the Blockiness vs. Upper Left Position \((x, y)\) of the 16x16 Block

Another thing that was attempted was to see how many Blockiness values are equal between the two data sets. From the two data sets of the 16 x 16 block division where the size of each data set was 58801, we found 9240 same values. The size of the images tested is 256 x 256 and thus the number of different blocks is \((256 - 15)^2 = 58801\). The number of equal values is about 15%. Based on that difference, a final test was made in order to see if we could find a structure that could be useful. We compared the two sets for equal values, and constructed a Boolean vector.
Then the vector was resized to $241 \times 241$, where $256-15=241$. The resulted matrix was viewed as a binary image, which can be seen in figure 5.5.7.

**Figure 5.5.7**

*Binary Representation of the Comparison of the two Blockiness Data sets*

Viewing the image does not really give us any further information. Again there could be a sort of structure hidden in the image that just we were not able to see it. An interesting and unexpected thing that we notice is that the differences in Blockiness values give a layout of the images structure. The original image was Lenna and we can see them and compare in Figure 5.5.8. It is probably irrelevant in the actual testing on YASS but it was an interesting observation nevertheless.
To sum up, we tried to analyse YASS with Blockiness, since we found out that it was increasing with respect to hidden message bits. We experimented by calculating the Blockiness for all possible blocks in the stego image and compared the results with the ones of the original cover. We did not find some structure or something useful but I do not exclude the possibility of finding methods that include Blockiness that could be used to attack YASS. The fact that YASS uses a complete different embedding process than any other known stego-system makes it hard to analyse and find a way to attack it.
CHAPTER 6

CONCLUSIONS

6.1 Evaluation and Problems Encountered

To begin with, in this study, an attempt to discuss several theoretical and practical forms of cryptography and steganalysis was made as an introduction to this field. The categorisation analysis was made in order to give our personal view on how steganalysis attacks categories, which are based on steganographic categorisation, can be interpreted and be closer to the practical form of steganalysis. An overview of classic stego-systems, embedding techniques and steganalysis methods was given as an introduction to the method that was implemented as part of the project.

The implementation on the attack on Outguess (Chapter 4) was successful and the results were similar to the ones on the literature [8]. The methodology of the attack was analysed experimentally by calculating data for each step of the theory behind the attack. Though, the implementation on Outguess could be more automatic, such as the one made on Jsteg. Outguess embedding process cannot be easily implemented since it has many factors that takes into consideration. In addition to this, UNIX and programming through it was something new to me and I found out quite late that I have to work in this environment to have something more complete. In addition to this, I managed to run Outguess late, as I first wanted to make sure the methodology works and test it on Jsteg. Despite all these, the implementation was accurate.

YASS stego-system, as it was mentioned before, was published on mid June and the time was limited to analyse it in more depth and possibly find better results. On the other, an attempt to use statistics that were already discussed in the report was made in order to find some structure on the embedding process of YASS. We attempted to introduce and express the ideas we had on YASS by providing appropriate graphs and explanations. This stego-system uses a new way of embedding procedure and can be very difficult for someone to manage and attack it or find useful information in a limited time.
6.2 Recommended Future Work

Further work can be done in the future in order the implementations and analysis, that was done in this report, more complete. The recommended future is listed as follows:

- The implementation of the attack on Outguess can be extended and become more automatic calling Outguess commands within the scripts.
- Testing of the method can be done on all transform domain stego-systems in order to be sure that this methodology can work on all current stego-systems that follow the JPEG grid, unlike YASS that uses a different embedding algorithm than usual stego-systems.
- More testing and maybe some changes in the final detection algorithm in order to be more accurate when testing stego images with small message length embedded and cover images.
- An implementation of the feature-based steganalysis technique that is discussed on Chapter 5 can be done by following the methodology described.
- Further analysis on YASS to try out more statistical properties in order to attack it or find more useful information on the structure of the stego images.

6.3 Conclusion

Steganography and steganalysis are fast growing sciences. A large amount of stego-systems and accurate steganalysis techniques, either target or blind, have been proposed and developed throughout the last 10 years. An interesting and appropriate question about steganography is if it actually is a large threat or it is rarely used by individuals. Government officials, especially in the US, have stated that steganography is a threat since there were suspicions that it is used by terrorists. On the other hand, even if steganography is widespread used the Internet is so large and chaotic that it makes it hard or even impossible to actually look at the right places. The attempt of N. Provos [23, 24] to use his steganalysis software in addition with a Web Crawler that can make a local copy of any JPEG images it encounters on a given Web site, resulted to detect 1% out of two million images downloaded. He used a distributed dictionary attack in order to find the secret messages but was not able to find a single message. Based on a study that found nearly 25% of all passwords vulnerable to dictionary attacks he concluded that steganography is not widespread used or they were looking at wrong places. In addition to this, an important thing to notice is that when someone wants to communicate through steganography, in our case, a short message will
most likely be embedded in the stego image. The ability of detecting a stego image with significant small message should be the most important thing.

The methods described are very accurate and with the use of blind steganalysis techniques many different stego-systems can be attacked with a single attack. As far as my research on steganography and steganalysis gone, it seems that steganalysis had the upper hand providing so many different approaches on detecting stego systems. On the other hand, stego-systems should try and be resistant to a wide range of attacks by preserving a lot of statistical properties such as YASS which is the most recent stego-system proposal.
REFERENCES


