An Introduction to Steganography

Robert Keller, rkeller@ucsc.edu

***Abstract*—This paper provides an introduction to steganography, the practice of hiding a message inside media without evidence that the message exists, and steganalysis, the study of how to discover and read said messages. The paper begins from the very basics with the definitions, and provides background about different forms of the art. It then moves into basic general implementation details and gives a very detailed explanation of how to implement LSB steganography. Finally, steganalysis techniques and methods for fighting those techniques are discussed.**

***Index Terms*—Steganography, steganalysis, LSB steganography, implementation**

I. Introduction

IT is a common practice in the world of computer security to send messages that are intended to be secret. The most obvious way to do so is to apply some sort of code to the message, rendering it unreadable. However, this method still provides one type of information to intruders that may be trying to intercept communication: it shows them that the user is sending a secret message.

This is where steganograpy comes in. Steganography is the art of sending a message without allowing any party other than those sending and receiving the message that it exists. Many methods have been invented to accomplish this task, the most common being embedding the message in a digital image. In this paper, we will cover the basics of both embedding such a message in a way that is very difficult to detect and the basics of searching for a message that others have hidden.

One note to remember is that cryptography and steganography are not mutually exclusive. Cryptography adds security to all messages, and this includes those hidden by steganography. Encrypting a message before hiding it makes the process much more secure. This paper assumes a knowledge of the basics of cryptography, but will only reference them sparingly.

II. Different forms of Steganography

Steganography has existed for quite some time. In ancient times, steganography was performed by tattooing a message on a servant's head and then letting his or her hair grow over it. The servant could then deliver the message in a completely unsuspicious manner without anybody knowing that the message even existed.

Today, methods have become rather more advanced[1]. One of the most popular ways to send a message is by secretly embedding it in an image and then sending the image instead. Many algorithms have been invented to do so, but most focus on the basic technique of changing the least significant bits of each color value in the image. Because the human eye can't tell the difference, most people won't suspect that a message is being passed and in some cases may even harbor the image for months without knowing it contains a message for somebody else.

Messages are also steganographically embedded into audio. Audio files offer several advantages and disadvantages. On the plus side, changes to the least significant bits of an audio file are normally extremely hard for humans to detect, much harder than when the same technique is applied to images. This is fairly easy to see; though a human can look at both a modified and unmodified image at the same time, it is much harder for he or she to listen to multiple pieces of audio at once and be able to tell the difference. As a disadvantage, most steganographic files must be compressed losslessly, which means that audio files will be far larger than image files. A final reason to use images instead of audio files is the relative ease with which images are passed around the Internet. A person sending music back and forth raises more suspicion than a person sending images.

Finally, video files have also sometimes been used to send steganographic images. Video files are much, much larger than either audio or image files, but because of their massive size, small messages are several orders of magnitude harder to detect. A two-page text message hidden inside a 5-GB DVD would be extremely hard to discover, especially if more advanced hiding techniques were used.

This paper will focus entirely on the image format, as it is the most widely used and researched.

III. Choosing an Image

The very first thing that should be done when sending a steganographic message is choosing the image. Having a good image is vitally important to making sure that the message will be undetectable. If the image has certain properties it will be very easy to tell that it has been modified.

A. Logos and Solid Colors

Firstly, images with large areas of solid colors should be avoided. This can be seen by looking at the most extreme case, hiding a message in an image that's 100% white. Changing the values of even the least significant bits (LSBs) of these will produce a small amount of contrast, which is very easily detectable by the human eye.

Images that are logos are normally created using vector graphics, and as such are very simple and contain large ares of solid colors. Changing the LSBs introduces noise to an image, but in images where no noise was initially present this is very easily detectable.

Instead of choosing an image with solid colors, images that have large areas of texture or that are initially grainy or bad quality are much better. The preponderance of noise in these images makes any introduced noise nearly unnoticeable to the human eye.

B. Image Size

A very important part of embedding a steganographic message is finding an image that is much larger than the message. If the message takes up every pixel in the image, the entire image will be noisy and will look very degraded when compared with the original. However, if the message is a much smaller percentage of the pixels, it will be much harder to notice any changes.

It should also be noted that images that are too large can produce suspicion when they stand alone. If a collection of images on a suspect computer are all normally compressed, but one or two images are far larger than the rest, those images will immediately be suspect. For this reason, images should be chosen so that they are slightly on the larger end of the normal range of images that are in a collection.

C. Multiple Images

Something that the budding steganographer should always do is test how much the insertion of a message changes an image. There are several calculations that can be done on both the original and stego images, such as calculating mean square error or Watson's metric [2], that should be tested on multiple pairs of images to make sure that the message is as hard to detect as possible. In general, when applying such calculations, the larger the difference between the results of the original and stego images, the less likely an image should be to choose.

IV. Implementing LSB Steganography

This section contains a step-by-step implementation of LSB steganography for images. The implementation described here (mostly) mirrors that used in the Java program included with this file, Stego.jar. The source code for the jar can be found inside the jar itself.

A. Goal of the Project

The final product should be a program that can both store and remove messages from images. These messages should be encrypted for extra security, and should be stored in the image in such a way that both their detection and removal should be extremely difficult. The program included with this file also performs limited steganalysis; this will be covered later on in the paper.

B. Writing the Message

The very first thing that should be done upon input of the message is encryption. In Stego.jar, encryption is performed using a cipher called DES. Being cryptography, DES is beyond the scope of this paper. Because the message will always be encrypted during writing and reading, we will hereafter refer to the encrypted version of the message as "the message" for ease of reading.

The immediate and most intuitive way to write the message is to break it into bits, and then write one bit to each pixel of the image in order. For example, if our message was 0100101, we would change the least significant bit of each of the first 7 pixels from the top left of the image to match the ones in our message. Though this technically works, it is fraught with weaknesses.

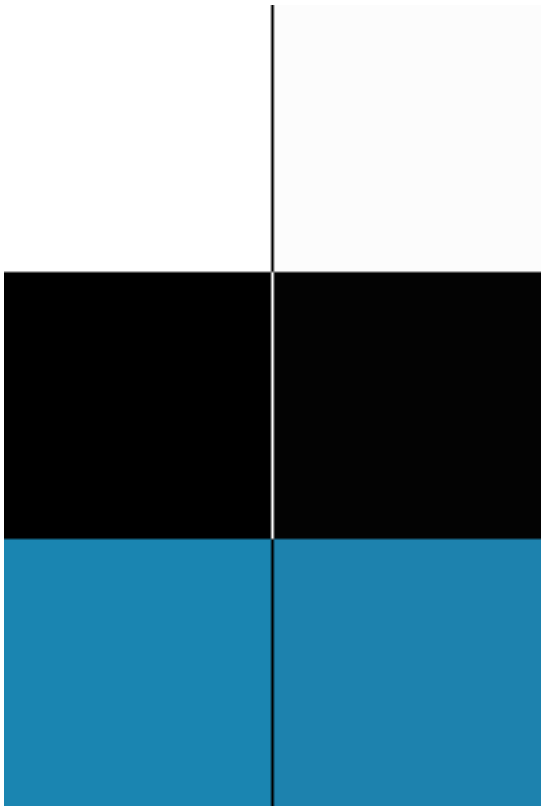

Fig. 1. Maximum color change by modifying the 2 LSBs.

One major difference between this implementation of steganography and others that can be found on the Internet is that this implementation writes to the *two* least significant bits of each color value. The main reason for doing this is size. As we saw in our discussion of embedding stego messages in video files, a message that changes a small amount of the pixels in its cover is far less likely to be discovered. The tradeoff in this is that the pixels that are modified will be slightly easier for the human eye to recognize.

When we change the two least significant bits of each color value, we can calculate the largest distance that a pixel can change from its original value. Each color value can change by a maximum of 3 values, meaning that the maximum change would be 9 values, which is still nearly indistinguishable by the human eye. Figure 1 contains a comparison of colors changed by the maximum possible value. As we can see, there is nearly no difference.

However hard to detect the message would be by the human eye, however, a computer performing statistical analysis would still be able to detect a message's presence by noting that the randomness, or variance, of the LSBs in the image change sharply just after the first 7 pixels, which is where our theoretical message ends. (Randomness is much higher in the message data because it has been encrypted, and encrypted messages have extremely

high entropy,) Many sources have thought to fix this by making an offset part of they key needed to extract the message. The message would then be inserted in a place other than the beginning of the image, and could only be found and extracted if the offset was known. However, this approach is still weak to statistical attacks. A computer would notice that the randomness of the LSBs was much higher in a single area of the message.

In this implementation, we suggest a solution to this problem: instead of placing all message bits into the image in order, we first compute a hash value of the provided key and then use that value to seed a pseudorandom number generator (PRNG). The generator is then cycled to produce a sequence of numbers that represent single pixels in the image. When message bits are written, they are written to those pixels, which are not necessarily in order or adjacent. This provides an added layer of security: even if analysis determines that there is a hidden message in the image, no decryption process can begin without first getting the message bits out of the image in the correct order, which for long messages is computationally infeasible.

It should be noted that both scrambled and unscrambled writing methods are present in Stego.jar, though scrambled writing is used by default. The unscrambled writing methods can be called from StegImage.java.

A key problem in steganography is both storing and determining the length of a message. This implementation takes the simple approach and simply stores an unencrypted 3-byte integer at the beginning of the data stream. In an unscrambled writing process, doing something like this would be very dangerous. Any user would be able to notice where the levels of randomness were highest and then simply look at the beginning of the area to find an integer. Scrambling the pixel order, however, protects against this, because there is no way to tell which bits represent part of the message length without knowing the key.

C. Reading the Message

If the message is written using the technique described above, it will only be able to be removed from the image by a person that has the key. The key is entered as the seed of a PRNG and then used to generate the locations of the pixels in the message. After the first 4 pixels are read, the first 3 bytes containing the message length are known, and the program knows how many more bytes to read.

The reading process is far simpler than the writing one, because it only requires the key and a PRNG. The same key is used to generate the pixel positions and to decrypt the message; this is secure because no algorithm yet exists for finding the seed of a PRNG given the pattern that it produces. This means that even if a third party discovers the message and somehow puts it back into the correct order (thus allowing them to know the sequence of random numbers that were generated), he or she would not know the seed, which itself is only a hash value of the actual key, and hash values are one-way functions.

V. Steganalysis

As the name implies, steganalysis is the opposite of steganography. It is the process of determining whether or not a given image contains a hidden message. Steganalysis relies heavily on statistics to do its job; there is really no perfect way of discovering if a particular image contains a message. There are a few methods that are in widespread use, but there are far too many in general to be collected into a single paper.

A. Comparison

The easiest and most reliable method for detecting a stego message is comparison. Comparison is only valid when the person that hid the message did so in an image that is available to the public. This allows both the normal image and the stego image to be accessed, and a message can be detected by simply comparing the pixel values.

This method (quite obviously) can be fought by only using original images as cover images. In the case that no original images are available, however, a widely available image can simply be resized and saved with a different compression level in order to create a new image. Images such as these are arguably more or less secure, because though they provide less suspicious targets for analysis than original images, they also have the chance of being widely distributed if messages are being sent by posting on a public forum.

B. Image Quality Metrics

A very intuitive method for discovering messages hidden in images is through the use of image quality metrics, which are mathematical operations that can be performed on pixels in images to determine a level of quality [3]. Because adding a stego message to an image distorts the colors, the resulting image contains artifacts and has a lower level of quality than the original. Image quality metrics can be used to detect this to a much more accurate level than the human eye.

One relatively simple IQM is the bit transition metric [4]. It is based off of the observation that an image can be broken down into several binary images, which are simply grids of 1's and 0's, by selecting the nth bit of color in a pixel. When this is done, analysis reports that 1's and 0's are not randomly distributed: instead, they are concentrated in physical areas on the image. We define a *bit transition* as a place on the border between physical areas of 0's and 1's.

When a stego message has been embedded in an image, the areas of 0's and 1's tend to be broken up because the bits in a stego message represent encrypted data instead of image data. This produces a jump in the amount of bit transitions in the image. For messages that are a large portion of the image, this becomes very noticeable and therefore easily detectable.

C. LSB Randomness

As described above, encrypting the message to be hidden causes the entropy of the message to skyrocket. In effect, it becomes entirely random. Randomness is a rare thing in data; so much, in fact, that encrypted data can be distinguished from normal data by analyzing its entropy.

LSB Randomness takes advantage of this fact. To perform this type of steganalysis, all the LSBs are read from an image and put back together into a data stream. Next, various statistical calculations are made, such as the variance or standard deviation of the data. If one area of the data is far more random than the remainder, then that area is the encrypted message.

VI. Fighting Steganalysis

All three analysis techniques above can be efficiently fought by changing the method of writing the message. As described, comparison can be fought by creating new images for steganography. Here we describe how the other two can be fought:

A. Image Quality Metrics

IQMs in general detect noise in a data stream. Normally, changing the LSBs of an image would introduce noise; however, if a smoothing function is applied to *every* LSB in the image, the noise will disappear.

The most popular smoothing function is the Fourier Transform, which is an invertible function that decreases entropy. It is a function of the complex plane, so in some cases, programmers will choose to implement the Discrete Cosine Transform instead. Both transforms can be efficiently implemented to run in $O(nlogn)$ time, but the Fourier Transform has proven to be more popular in the general population.

Because both transforms are invertible, a final step to any steganography program should be to smooth the values of the LSBs to prevent detection. This will make detection using IQMs much harder.

B. LSB Randomness

The transformations described above will in fact *not* work to remove randomness from the data stream. Though they will decrease the entropy of the data, they can in no way completely remove it, and will in most cases only remove a small amount. Therefore a separate method is needed to deal with the ease of detection for areas of heavy entropy in the LSBs.

Fortunately, the implementation described above effectively combats this method of steganalysis. Using a key to seed a random number generator and then changing the values of generated pixels gets rid of any large areas of randomness in the image's LSBs as long as the message is relatively small compared to the image (this reinforces the importance of choosing a message that's relatively small). When smoothing transforms are applied to these LSBs, the result will have far less entropy than an image with a message in one place would have.

VII. Conclusion

We have outlined the basics of steganography, as well as a complete implementation of LSB steganography. We have also discussed several steganalysis techniques, how they are used, and how to fight them.

Steganography is a very powerful tool; it allows secret messages to be sent without the opponent knowing that they exist. In fact, entire filesystems based off of steganography have been proposed. There are many more techniques that are beyond the scope of this paper as well; LSB steganography is not the only way to hide messages inside files. We encourage the reader to investigate the references section of this paper in order to learn more about how steganography is implemented in the real world, and hope that the reader will be able to make his or her own advancements in the field.

References

[1] Natarajan Meghanathan and Lopamudra Nayak, *Steganalysis Algorithms for Detecting the Hidden Information in Image, Audio, and Video Cover Media* International Journal of Network Security & Its Application (IJNSA), Vol.2, No.1, January 2010

[2] Mehdi Kharrazi, Husrev T. Sencar, and Nasir Memon, *Cover Selection for Steganographic Embedding* Proceedings ICIP, Atlanta, GA, October 2006.

[3] Ismail Avcibas, Nasir Memon, and Bulent Sankur, *Steganalysis Using Image Quality Metrics*, IEEE Transactions on Image Processing, Vol. 12, No. 2,Feb 2003

[4] Ismail Avcibas, Mehdi Kharrazi, Nasir Memon, and Bulent Sankur, *Image Steganalysis with Binary Similarity Measures*, EURASIP Journal on Applied Signal Processing, Vol 2005, Jan 2005