The Unseen Challenge Data Sets

Anderson Rocha
Siome Goldenstein

Institute of Computing
University of Campinas (Unicamp)
CEP 13084–851, Campinas, SP – Brazil
{anderson.rocha, siome}@ic.unicamp.br

Walter Scheirer
Terrance Boult

Vision and Security Technology Lab
University of Colorado
CO 80933-7150, Colorado Springs, CO, US
{wjs3,tboult}@vast.uccs.edu

Abstract

Nowadays, it is paramount to study and develop robust algorithms to detect the very existence of hidden messages in digital images. In this paper, we provide two data sets for the Unseen Challenge of the ***First IEEE Workitorial on Vision of the Unseen (WVU)****. Example usage of the data sets is demonstrated with the stegdetect analysis tool, with surprising results reported. Our objective is to challenge researchers to assess their Digital Image Steganalysis state-of-the-art algorithms.*

1. Introduction

Steganography is the art of secret communication. Its purpose is to hide the presence of communication, as opposed to cryptography, which aims to make communication unintelligible to those who do not possess the correct keys [1]. Steganography has a lot of applications such as feature location (identification of subcomponents within a data set), captioning, time-stamping, and tamper-proofing (demonstration that original contents have not been altered). However, there are indications that Steganography has been used to spread child pornography via the Internet [2, 3].

Thus, it is paramount to study and develop robust algorithms to detect the very existence of hidden messages [4]. In this venue, *Digital Steganalysis* comprises the techniques that have been devised to distinguish between *non-stego* or *cover objects*, and *stego objects*. *Non-stego* objects are those that do not contain a hidden message. *Stego-objects* are those that contain a hidden message. Further, the ultimate goal of Steganalysis is the recovery of the hidden message. This hidden message is often encrypted, leaving the recovery of the "plaintext message", after hidden message recovery, as a problem of cryptanalysis. The *vision* aspect of steganlaysis is the understanding of how digital images are manipulated to carry *unseen* data.

In this paper, we provide two data sets for the Unseen Challenge of the *First IEEE Workitorial on Vision of the Unseen* (WVU). Our objective is to challenge researchers to assess their Digital Image Steganalysis state-of-the-art algorithms. We hope that with these data sets it will be easier to assess existent and new detection algorithms under similar validation conditions in the near future.

We provide two data sets: one loss-less- and one lossy-compressed. The reason is that currently, key image Steganography software either uses DCT embedding, or bit twiddling/insertion (c.f., Section 4). We have used four different message sizes for four different JPEG algorithms and three different PNG algorithms. In the following sections, we provide more details about the data set setup.

The remainder of this paper is organized as follows. Section 2 presents some Steganography historical remarks. Section 3 defines some important Information Hiding concepts. Section 4 outlines the commonest Steganography approaches available in the literature. Section 5 brings a detailed description of the two data sets we have created. Section 6 shows the Steganography tools we have used to create the data sets. Section 7 shows how these data sets may be used for experimentation, with the stegdetect analysis tool as an example. Finally, Section 8 presents some conclusions and remarks.

2. Historical remarks

Throughout history, people always have aspired to add more privacy and security for their communications [5, 6]. One of the first documents describing Steganography comes from the *Histories* of Herodotus. In one such anecdote found in this work, a man named Harpagus killed a hare, hid a message in its belly, and sent it along with a messenger posing as a hunter. [5]. In another instance, Histaieus, in order to convince his allies that it was time to begin a revolt

978-1-4244-2340-8/08/$25.00 ©2008 IEEE

against Medes and the Persians, shaved the head of his most trusted slave, tattooed the message on his head and waited until his hair grew back before sending him out as a covert messenger.

Flashing forward through the centuries to our own time, we find that after the September 11^{th} attacks, some researchers have raised attention to the possibility that the Al Qaeda terror network had used Steganography techniques to coordinate the World Trade Center attacks. Almost six years later, no proof of this claim has surfaced [8, 9, 10, 4]. However, Steganography continues to remain in today's security news [4].

3. Terminology

According to the general model of *Information Hiding*, a *clean object* is the one that does not contain any hidden content. Its counterpart is named *stego object*. The *embedded data* is the message that we want to send secretly. Often, we hide the embedded data in an innocuous medium referred to as *cover message*. There are many kinds of cover messages such as *cover text*, when we use a text to hide a message, or *cover image*, when we use an image to hide a message. The embedding process produces a *stego object* (e.g., stego image) which contains the hidden message. We can use a *stego key* to control the embedding process so as to restrict detection and/or recovery of the embedded data to parties who have the correct keys.

Figure 1 shows the typical process of hiding a message in an image. First, we choose the message we want to hide. Further, we use a selected key to hide the message in a previously selected cover image which produces the stego image. Additionally, the selected key can be used as the seed of a pseudo-random number generator that selects the image features to be altered in the embedding process (e.g., the LSBs to be used).

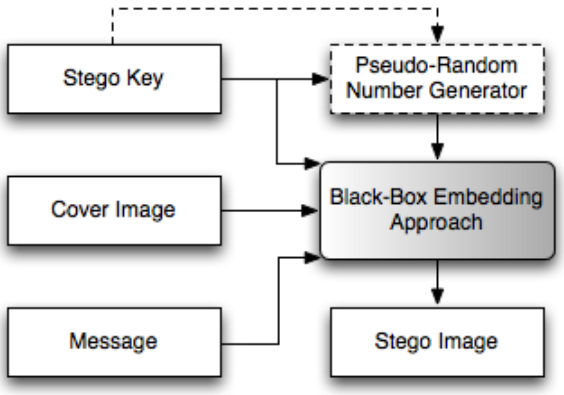


Figure 1. Typical scenario for data hiding.

When designing information hiding techniques, three competing aspects are considered: capacity, security, and robustness [11]. *Capacity* refers to the amount of information we can embed in a cover object. *Security* relates to an eavesdropper's inability to detect the hidden information. *Robustness* refers to the amount of modification the stego object can withstand before an adversary can destroy the information [11]. Steganography strives for high security and capacity. Hence, a successful *attack* to the Steganography consists of the detection of the hidden content. Our objective, with the two data sets we present in this paper, is first and foremost to detect whether or not an image contains a hidden message. Afterwards, if possible, it would be interesting to point out the estimated message size.

4. Steganography main techniques

In general, steganographic algorithms are based on replacing a noise component of a digital object with a pseudo-random secret message [12]. In the following, we review some of the main approaches to hide messages in digital images. We expect researchers to present reliable detectors for the very existence of hidden messages encoded by such approaches. It is interesting also to validate blind detectors able to point out stego images regardless of the embedding tool.

4.1. LSB insertion/modification

Among all message embedding techniques, *least significant bits* (LSB) insertion/modification is a difficult one to detect [12, 7, 13], and it is imperceptible to humans [7]. However, it is easy to destroy [13].

A typical color image has three channels: red, green and blue (R,G,B); each one offers one possible bit per pixel to the hiding process. In Figure 2, we show an example of how we can possibly hide information in the LSB fields. Suppose that we want to embed the bits **1110** in the selected area. In this example, without loss of generality, we have chosen a gray-scale image. In this case, we have one bit available in each image pixel for the hiding process. We want to hide four bits, hence we need to select four pixels. To perform the embedding, we tweak the selected LSBs according to the bits we want to hide. If the bit to be hidden is 1 (0), we tweak the selected pixel's LSB to 1 (0).

4.2. FFTs and DCTs

A very effective way of hiding data in digital images is to use a Discrete Cosine Transform (DCT) or a Fast Fourier Transform (FFT) and hide the information in the transformed space. The DCT algorithm is one of the main components of the JPEG compression technique [14]. In general, DCT and FFT work as follows:

1. Split up the image into 8×8 blocks.

Figure 2. The LSB embedding process. Image from the oil painting *Cape Cod Evening* by Edward Hopper.

2. Transform each block via a DCT/FFT. This outputs a multi-dimensional array of 64 coefficients.

3. Use a quantizer to round each of these coefficients. This is essentially the compression stage and it is where data is lost.

4. At this stage, you should have an array of streamlined coefficients, which are further compressed via a Huffman encoding scheme or similar.

5. To decompress, apply the inverse DCT/FFT.

The hiding process using a DCT/FFT is rather difficult to detect because if someone analyzes the pixel values of the image directly, he/she would be unaware that anything is different [7]. There are many possibilities of hiding data in images via DCTs/FFTs as we show next.

4.2.1 Least significant coefficients

We can use least-significant bits of the quantized DCT/FFT coefficients as redundant bits in which to embed the hidden message. The modification of a single DCT/FFT coefficient affects all 64 image pixels in the block [11].

JSteg (c.f., Section 6.2.3) is a simple algorithm that sequentially replaces the least significant bit of DCT or FFT coefficients with the message's data. On the other hand, Outguess (c.f., Section 6.2.4) is an improvement over JSteg in the sense that it chooses the best coefficients to perform the hiding in order to minimize the amount of changes in the cover image.

4.2.2 Block tweaking

We can hide the data in the quantizer stage [7]. If we want to encode the bit value 0 in a specific 8×8 block of pixels, we can do this by making sure that all the coefficients are even in such a block, for example by tweaking them. In a similar approach, bit value 1 can be stored by tweaking the coefficients so that they are odd.

With the block tweaking technique, a large image can store some data that is quite difficult to destroy when compared to the LSB method. Although this is a very simple method and works well in keeping down distortions, it is vulnerable to noise [7, 12].

4.2.3 Coefficient selection

This technique consists of the selection of the k largest DCT or FFT coefficients $\{\gamma_1 \dots \gamma_k\}$ and the modification of them according to a function f that also takes into account a measure α of the required strength of the embedding process. Larger values of α are more resistant to error, but they also introduce more distortions.

The selection of the coefficients can be based on visual significance (e.g., given by zigzag ordering [7]). The factors α and k are user-dependent. The function $f(\cdot)$ can be such as

$$f(\gamma_i') = \gamma_i + \alpha b_i \tag{1}$$

where b_i is a bit we want to embed in the coefficient γ_i.

4.2.4 Wavelets

DCT/FFT transformations are not so effective at higher-compression levels. In such scenarios, we can use wavelet transformations instead of DCT/FFTs to improve robustness and reliability [7].

Wavelet-based techniques work by taking many wavelets to encode a whole image. They allow images to be compressed by storing the high frequency details in the image separately from the low frequency ones. Hence, we can use the low frequencies to compress the data. Furthermore, we can use a quantization step to compress even more. Information hiding techniques using wavelets are similar to the DCT/FFT ones. Some of them, also use the quantization process to embed the message. Other ones, use coefficient selection [7].

5. Data sets

As we have mentioned earlier, in this paper we provide two data sets: one stored in the PNG loss-less-compressed image format and the other one stored in the lossy-compressed JPEG image format.

We have chosen these image formats based on their prevalence on the Internet. Thus, we have downloaded the

images from common Internet web-crawlers such as Google Images[1], Yahoo Images[2], and Flickr[3].

For the PNG data set, we have used the Steganography tools: *Camaleão*, *SecureEngine*, and *Stash-It*. For the JPEG data set, we have used the Steganography tools: *F5*, *JEmbed*, *JPHide*, and *Outguess*. For each one of these data sets, we create an embedding scenario comprising four different content embedding sizes.

5.1. Message sizes

Each stego category uses a different embedding tool. For each stego tool, we provide 4 different embedding sizes: *tiny, small, medium*, and *large* messages:

1. **Tiny messages**. They are those which use less than 5% of the channel capacity.
2. **Small messages**. They are those which use more than 5% and less than 15% of the channel capacity.
3. **Medium messages**. They are those which use more than 15% and less than 40% of the channel capacity.
4. **Large messages**. They are those which use more than 40% of the available channel capacity.

For the PNG data set, this division is always explicit. However, for the JPEG data set, this division is not always reported.

The hidden messages content embodies single random bit sequences, snippets of MP3 songs, plain-text sequences, and other images. No encryption prior to the encryption some of the tools perform as part of their embedding process was performed.

5.2. Categories creation

Each data set contains one set of *clean images* and one set of *stego images* created using the tools described in Section 6.

The stego images are divided into four categories, one for each embedding tool used. In both data sets, we also divide the *clean images* into *modified* and *non-modified* images. The *modified* images are those which have undergone some image processing manipulation such as cropping, overlay, and object-appending. The objective was to create a more realistic scenario with possible manipulations that would make hidden message detection more difficult. The *non-modified* clean images are those without any kind of image processing manipulation.

The JPEG data set also contains additional subcategories. Each one of its four stego categories is divided into: *Animals, Business, Maps, Natural, Tourist*, and *Vacation* sub-categories. The challengers can use this information to further provide any insight about the detection on specific image categories or simply merge the subcategories. Finally, the JPEG *clean category* also contains a *Misc* sub-category which embodies part of the images of PNG clean data set including the *modified* and *non-modified* sub-categories.

[1] http://images.google.com
[2] http://images.yahoo.com
[3] http://www.flickr.com

5.3. Training and testing data sets

We have divided the two data sets into training and testing, as we show in the following sub-sections.

5.3.1 PNG data set

Table 1 presents the number of images for each one of training PNG stego categories.

	Tiny	Small	Medium	Large
Camaleão	400	400	400	400
SecureEngine	380	387	385	380
Stash-It	399	400	400	400
Total	1,179	1,187	1,185	1,180

Table 1. Number of images within each training PNG stego category. 4,731 images in total.

Table 2 presents the number of images for the training PNG clean category.

Non-modified	2,000
Append-modified	666
Crop-modified	667
Overlay-modified	667
Total	4,000

Table 2. Number of images within the training PNG clean category. 4,000 images in total.

Table 3 presents the number of images for each one of testing PNG stego categories.

	Tiny	Small	Medium	Large
Camaleão	250	250	250	250
SecureEngine	250	250	250	243
Stash-It	250	250	250	250
Total	750	750	750	743

Table 3. Number of images within each testing PNG stego category. 2,993 images in total.

5.3.2 JPEG data set

Table 4 presents the number of images for each one of training JPEG stego categories. For this category, we do not

provide any prior information about the message sizes. The division is by image category. It is possible to have more than two equal images within a category with different hidden content.

	F5	JPHide	JSteg	Outguess
Animals	1,732	2,127	244	436
Business	3,779	–	124	11
Maps	3,361	–	112	68
Natural	5,211	1,113	232	70
Tourist	4,968	1,721	268	160
Vacation	2,960	353	100	35
Total	22,011	5,314	1,080	780

Table 4. Number of images within each training JPEG stego category. 29,185 images in total.

Table 5 presents the number of images for the training JPEG clean category.

Animals-Non-modified	61
Business-Non-modified	31
Maps-Non-modified	28
Natural-Non-modified	58
Tourist-Non-modified	67
Vacation-Non-modified	25
Misc-Non-modified	1,996
Misc-Append-modified	665
Misc-Crop-modified	666
Misc-Overlay-modified	662
Total	4,259

Table 5. Number of images within the training JPEG clean category. 4,259 images in total.

Table 6 presents the number of images for each one of testing JPEG stego categories.

	Tiny	Small	Medium	Large
F5	250	250	250	250
JPHide	240	322	318	101
JSteg	198	202	199	198
Outguess	481	421	–	–
Outguess 0.13	491	425	–	–
Total	1,660	1,620	767	549

Table 6. Number of images within each testing JPEG stego category. 4,596 images in total.

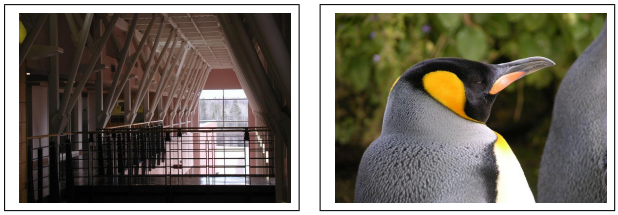
Table 7 presents the number of images for the testing JPEG and PNG clean category.

5.4. Data set examples

Figures 3 and 4 depict some examples from the PNG and JPEG data sets, respectively. The depicted images do not contain any kind of image manipulation or hidden content.

	Clean
PNG set	504
JPEG set	998

Table 7. Number of images within the testing PNG clean and testing JPEG clean categories.



(a) (b)

Figure 3. PNG data set examples. *Credits* to Flickr contributors: (a) Angeline Q.; and (b) Synwell Liberation Front.

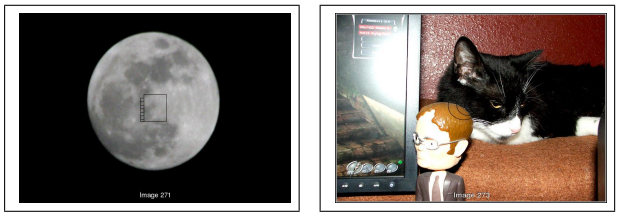
(a) Animals. (b) Maps. (c) Business. (d) Tourist. (e) Nature. (f) Vacation.

Figure 4. JPEG data set examples.

5.5. Data set manipulated examples

Figures 5–7 depict some images containing possible image manipulations. Such manipulations can be prior to or after the message embeddings and can appear both in the training and testing PNG and JPEG data sets. In general,

such manipulations are simple and were performed using Image Magick[4], a software suite to create, edit, and compose images.



(a) (b)

Figure 5. Object appending manipulation examples. *Credits* on the original images to Flickr contributors: (a) Robert Keereweer; and (b)TReflex

(a) (b)

Figure 6. Image cropping manipulation examples. Such examples were cropped from larger images. *Credits* on the original images to Flickr contributors: (a) Michael Mahler; and (b) Gerad Coles.

(a) (b)

Figure 7. Overlay manipulation examples. *Credits* on the original images to Flickr contributors: (a) Alan Brid; and (b) Mike Lee

6. Steganography tools

In this section, we present the Steganography tools we have used in the data set creation. All tool URLs were last accessed on 4/27/08. Additionally, some tools are available from our website[5]. Tools are presented in alphabetic order.

6.1. Bit twiddling/insertion

In this section, we present the Steganography tools we have used in the creation of the PNG data set. These techniques rely on bit twiddling/insertion, i.e., usually they alter some pixels directly (e.g., LSB insertion).

6.1.1 Camaleão

Created by Rocha et al. [15], Camaleão[6] is a simple LSB insertion/modification Steganography software. The tool uses cyclic permutations and block cyphering to hide messages in the least significant bits of loss-less compressed images. Camaleão is available both in C++ and Java implementations.

6.1.2 SecureEngine

Created by Adrien Pinet, SecureEngine[7] is a Steganography tool designed for JPEG, PNG and BMP images. It works with five cryptographic algorithms: Blowfish, Gost, Vernam, Cast256, and Mars.

6.1.3 Stash-It

Created by Chris Losinger, Stash-It[8] is a simple Windows based stego program for data hiding inside a perfectly normal BMP, GIF, TIFF, PNG or PCX file. Stash-It software hides data in the least significant bits of images. It does not appear to have any additional encryption features.

6.2. DCT coefficient twiddling/insertion

In this section, we present the Steganography tools we have used in the creation of the JPEG data set. These techniques rely on bit DCT twiddling/insertion, i.e., usually they work on the frequency domain (e.g., DCT coefficients insertion/modification).

6.2.1 F5

Created by Pfitzmann and Westfeld [16], the F5[9] motivation is that it is resilient to χ^2 statistical attacks [17]. Instead of replacing the LSBs of quantized DCT coefficients with the message bits, F5 decreases the absolute value of the coefficients. The F5 algorithm embeds message bits into randomly-chosen DCT coefficients and employs matrix embedding that minimizes the required number of changes to embed a message of certain length.

6.2.2 JPHide

Created by Allan Latham, JPHide[10] uses the cryptographic Blowfish algorithm to provide a stream of pseudo random

[4] http://www.imagemagick.org
[5] http://www.liv.ic.unicamp.br/wvu/
[6] http://www.ic.unicamp.br/ rocha/sci/stego
[7] http://www.sharewareplaza.com/SecurEngine-download_4268.html
[8] (http://www.smalleranimals.com/stash.htm)
[9] http://www.inf.tu-dresden.de/ aw4
[10] http://linux01.gwdg.de/ alatham/stego.html

control bits which determine where to store the bits of the hidden file. The program allows embeddings lower than 35KB only. As stated by the authors, larger embeddings can cause visual and/or statistical artifacts.

6.2.3 JSteg

Created by Derek Upham, JSteg[11] is apparently the first tool to do Steganography in JPEG images. Version 1.0 includes 40 bit RC4 encrytion, determination of the amount of data a JPEG can hide beforehand, and user-selectable JPEG options (e.g., degree of compression). Specifically, JSteg hides the data inside images stored in the JFIF format of the JPEG standard. The embedding occurs after the DCT coefficients quantization (c.f., Section 4.2). After this step, the LSB of all non-zero frequency coefficients are replaced with successive bits from the message to be hidden, and these modified coefficients are sent to the Huffmann encoder.

6.2.4 Outguess

Created by Niels Provos, Outguess[12] relies on data specific handlers that extract redundant bits and write them back after modification. For JPEG images, Outguess preserves statistics based on frequency counts. As a result, statistical tests based on simple frequency counts are unable to detect the presence of steganographic content [18]. Outguess uses a generic iterator object to select which bits in the data should be modified. A seed can be used to modify the behavior of the iterator. It is embedded in the data along with the rest of the message. By altering the seed, Outguess tries to find a sequence of bits that minimizes the number of changes that have to be made in the data.

7. Example Usage

To demonstrate the usage of the data set, we have run the popular stegdetect[13] analysis tool against the training and testing JPEG sets. Stegdetect is capable of detecting all algorithms in the JPEG set, with the exception of Outguess 0.2. First, stegdetect is run with default parameters against the training set, which is designed to provide a sizable quantity of data for positive identification purposes, as well as examples of clean images with modifications. The results for stegdetect run against the entire training set are presented in Table 8. The overall false detect rate for the clean image set is 8.6%. Good detection and algorithm recognition is achieved for every algorithm except Outguess 0.2, which is to be expected. At this point, an analysis algorithm can be tuned and re-run against the training set to evaluate performance variation.

	Detected, C	Detected, I	No Steg
Clean	-	360	3809
F5	22011	0	0
JPHide	4506	604	204
JSteg	638	421	21
Outguess 0.13	220	10	295
Outguess 0.2	13	5	237

Table 8. Results for stegdetect run against the entire training set. "Detected, C" indicates the correct steganography algorithm was identified, "Detected, I" indicates steganography was detected, but an incorrect algorithm was identifed, and "No Steg" indicates no steganography was detected.

Second, stegdetect is run with the default parameters against the test set, which is designed to provide four controlled levels of embedded content per algorithm. The results for stegdetect run against the entire testing set are presented in Table 9. Similar to the results achieved for the training set, the overall false detect rate for the clean image set is 8.0%. Twenty of the clean images failed to be processed by stegdetect. There are significant differences between the results presented in Tables 8 and 9, with weaker performance noted overall. This is not surprising, based on the designed difficulty of the testing set. We provide detailed testing results for JPHide in Table 10. Looking at this table, we see that stegdetect actually performs poorly with the large embeddings (non-intuitive), as well as for the small and tiny embeddings (expected). Thus, we can learn a great deal about the conditions at which various analysis algorithms fail with the testing set.

	Detected, C	Detected, I	Negative
Clean	-	899	79
F5	0	216	784
JPHide	333	153	495
JSteg	444	353	0
Outguess 0.13	206	31	679
Outguess 0.2	32	37	833

Table 9. Results for stegdetect run against the entire testing set.

JPHide	Large	Medium	Small	Tiny
Detected, C	51	249	25	8
Detected, I	47	61	35	10
Negative	3	8	262	222

Table 10. Detailed results for stegdetect run against all four directories of the JPHide testing set.

[11] http://zooid.org/ paul/crypto/jsteg/

[12] http://www.outguess.org/

[13] http://www.outguess.org/detection.php

8. Conclusions and remarks

The data sets we are providing consist of both lossless (PNG) and lossy (JPEG) imagery, across four different embedding tools per image type. We have attempted to create variation in the set by introducing four distinct sizes of embedded content in the PNG set, and a variety of embedding sizes in the JPEG set, along with a series of modified images that contain no embedding. The training set is large enough to provide the researcher with a wealth of data to construct and/or evaluate various algorithms and detection techniques. From the results reported for stegdetect, we were able to draw some surprising and interesting conclusions about its operational performance for varying levels of embedding content.

Valid approaches may perform detection, detection and recovery (size or content), detection and destruction, or fusion. In closing, we hope this challenge set will endure as a common comparison set for researchers working in this space. To our knowledge, a standard reference set has yet to emerge in the steganalysis field, thus, the set introduced here is a first step in that direction.

Data sets are available at:
http://data.vast.uccs.edu/wvu/
http://www.liv.ic.unicamp.br/wvu/datasets.php

9. Acknowledgments

We would like to thank the financial support of Fapesp (Grant 05/58103-3) and AF STTR, Contract No. FA9550-05-C-0172. Furthermore, we also thank Ana Cristina de Araújo Oliveira for the WVU'08 insightful logo and the people that have contributed to the data set creation: Edith Leung, Devin McKinney, Justin Schiff, Andrew Kurfess and Miguel Lezcano Gonzalez.

References

[1] F. Petitcolas, R. Anderson, and M. Kuhn, "Information hiding - a survey," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1062–1078, July 1999.

[2] S. V. Hart, "Forensic examination of digital evidence: a guide for law enforcement," National Institute of Justice NIJ-US, Washington DC, USA, Tech. Rep. NCJ 199408, September 2004.

[3] S. Morris, "The future of netcrime now: Part 1 - threats and challenges," Home Office Crime and Policing Group, Washington DC, USA, Tech. Rep. 62/04, 2004.

[4] A. Rocha and S. Goldenstein, "Steganography and steganalysis in digital multimedia: Hype or hallelujah?" *Journal of Theoretical and Applied Computing (RITA)*, vol. 14, no. 2, 2007.

[5] B. Norman, *Secret warfare, the battle of Codes and Ciphers*, 1st ed. Acropolis Books, 1980.

[6] D. Kahn, "The history of steganography," in *Intl. Workshop in Information Hiding (IHW)*, 1996, pp. 1–5.

[7] P. Wayner, *Disappearing Cryptography - Information Hiding: Steganography & Watermarking*, 2nd ed. Morgan Kaufmann, 2002.

[8] P. Wallich, "Getting the message," *IEEE Spectrum*, vol. 40, no. 4, pp. 38–40, April 2003.

[9] S. Cass, "Listening in," *IEEE Spectrum*, vol. 40, no. 4, pp. 32–37, April 2003.

[10] J. Kumagai, "Mission impossible?" *IEEE Spectrum*, vol. 40, no. 4, pp. 26–31, April 2003.

[11] N. Provos and P. Honeyman, "Hide and seek: an introduction to steganography," *IEEE Security & Privacy Magazine*, vol. 1, no. 3, pp. 32–44, March 2003.

[12] R. Anderson and F. Petitcolas, "On the limits of steganography," *Journal of Selected Areas in Communications (JSAC)*, vol. 16, no. 4, pp. 474–481, May 1998.

[13] A. Rocha and S. Goldenstein, "Progressive randomization for steganalysis," in *Intl. Workshop on Multimedia and Signal Processing (MMSP)*, 2006, pp. 314–319.

[14] R. Gonzalez and R. Woods, *Digital Image Processing*, 3rd ed. Prentice-Hall, 2007.

[15] A. Rocha, S. Goldenstein, H. A. X. Costa, and L. M. Chaves, "Camaleão: um software de esteganografia para proteção e segurança digital," in *Simpósio de Segurança em Informática (SSI)*, 2004.

[16] A. Westfeld, "F5 – a steganographic algorithm: High capacity despite better steganalysis," in *Intl. Workshop in Information Hiding (IHW)*, 2001, pp. 289?–302.

[17] A. Westfeld and A. Pfitzmann, "Attacks on steganographic systems," in *Intl. Workshop in Information Hiding (IHW)*, 1999, pp. 61–76.

[18] N. Provos, "Defending against statistical steganalysis," in *Usenix Security Symposium*, vol. 10, 2001, pp. 24–36.