STRATEGIES FOR STEGANALYSIS OF BITMAP GRAPHICS FILES

THESIS

Christopher J. Fogle
Captain, USAF

AFIT/GCS/ENG/99M-05

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not necessarily reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/99M-05

STRATEGIES FOR STEGANALYSIS OF BITMAP GRAPHICS FILES

THESIS

Presented to the Faculty of the Graduate School of Engineering
Of the Air Force Institute of Technology
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Christopher J. Fogle, B.S.
Captain, USAF

March 1999

AFIT/GCS/ENG/99M-05

STRATEGIES FOR STEGANALYSIS

OF BITMAP GRAPHICS FILES

THESIS

Christopher J. Fogle, B.S.
Captain, USAF

Approved:

Henry B. Potoczny — Feb. 17, 1999.
Dr. Henry B. Potoczny, Ph.D. — Date
Chairman

Gregg H. Gunsch — 12 FEB 99
Gregg H. Gunsch, Lt Colonel, USAF — Date

John S. Crown — 12 Feb 99
John S. Crown, Major, USAF — Date

Acknowledgments

I would like to express my sincere appreciation to my research advisor, Dr. Henry Potoczny, for giving me incredible freedom to explore the subject of steganalysis and perform research that I felt was important. His wit and encouragement made a seemingly impossible task well worth the effort. I thank my committee members, Lieutenant Colonel Gregg Gunsch and Major John Crown, for their interest and support of this initial foray into the unique field of information hiding. Also, to the many government agencies and fellow academic researchers who shared their insights with me to help me get started, I thank you.

I must also thank my fellow classmates who endured several, seemingly endless presentations on steganography and steganalysis. Their kind words of encouragement made the long research process well worth the journey.

Finally, and most importantly, I would like to express my most heartfelt appreciation to my wife and best friend, Debbie, and my kids, Stephen and Dayna. Their love and understanding was the greatest support and comfort to me during the long hours spent away from home. Without them in my corner, my research most likely would not have been possible, and almost certainly not fun. We came here as a family, and with God's help, we succeeded as a family!

Christopher J. Fogle

Table of Contents

Page

List of Figures

Page

List of Tables

Page

Abstract

Steganography is the art and science of communicating through covert channels. The goal of steganography is to hide the fact that a message is even being transmitted. In the context of today's digital world, this ancient practice is enjoying resurgence due to the plethora of hiding places made possible by modern information media. Of particular concern is the use of graphics image files to conceal both legitimate and criminal communications.

Steganalysis of graphics files includes methods to detect the presence of embedded information [1] and subvert the information channel of a particular container file by distorting or overwriting the embedded information. While manual investigation of a file sometimes yields indicators that embedded information is present, the process of examining thousands of files is painstaking. This thesis explores steganalysis strategies that could reduce the search space an investigator must confront. Emphasis is placed on those methods that are easily incorporated in an automated detection tool. Such methods could be employed in a variety of missions, including information protection, intelligence, and law enforcement.

When the properties of the original cover file are known *a priori*, detection is trivial. This thesis develops two strategies that provide reliable detection of embedded information when original file characteristics are unknown – a problem known as blind

---

[1] The term *information* is used throughout this document with the understanding that *data* is actually being embedded, and that it becomes information once it is detected, assembled, and interpreted.

steganalysis. The first technique applies a common digital image processing point-noise detection filter to the steganalysis problem. Pixels in the image that are significantly different from those around it generate a larger response to the filter than those pixels that are similar. The percentage of "different" pixels is used to select images that possibly contain embedded information. The second strategy examines the byte-frequency distribution of a file. Files that have byte-value distributions which fall outside the typical population or sample distributions are selected as suspect images.

The results indicate that when the strategies are used in combination, they provide a high probability of detection in 75% of the search space. The point-noise threshold test achieves 98% success rates in those cases where the embedding process results in a visibly distorted image. The byte-frequency analysis test achieves a 100% success rate against a popular steganography tool that produces no visible distortion.

Strategies for Steganalysis of Bitmap Graphics Files

I Introduction

1.1 Steganography Defined

Steganography is an ancient art that has been reborn in the digital world. The word *steganography* comes from the ancient Greek words *stegos,* which means 'covered or hidden from view,' and *graphein* –'to write [9]. A more modern definition has been submitted by steganography researcher Markus Kuhn as the "art and science of communicating in a way which hides the existence of the communication [1]." Its goal is to "hide messages inside other 'harmless' messages in a way that does not allow any 'enemy' to even detect that there is a second secret message present" – in other words, the information is hidden in plain sight. While some might immediately liken steganography to cryptography, the two are fundamentally different. Hidden information is not necessarily secure, and secure information is not necessarily hidden. The distinction between the two is made clear in the following discussion.

In the science of cryptography, information is secured by transforming the original data into encrypted data via an enciphering scheme. As shown in Figure 1, encryption produces output, or ciphertext, that is basically unintelligible.

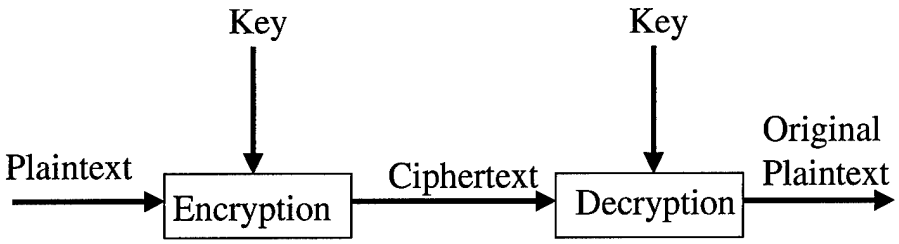


Figure 1, Classical Cryptography

In contrast, steganography leaves the original data unchanged and hides it. Through the use of an embedding technique, the original information is hidden within an innocuous cover media. To the casual observer, the cover appears normal. The original data is eventually recovered by applying the reverse of the original embedding technique, as shown in Figure 2.

The presumption in cryptography is that an adversary knows communication is occurring and is able to intercept it. An adversary is often aware that the information is encrypted and knows the algorithm used to encrypt it. The basis for security, therefore, is the time and level of resources needed to break the encryption key.

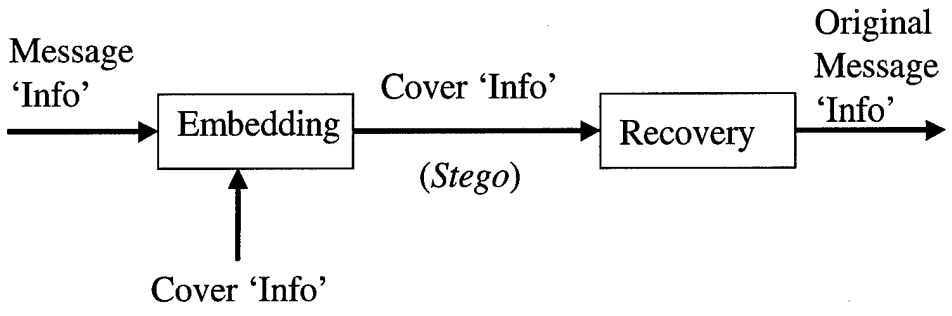


Figure 2, Steganography

In contrast, steganography presumes the enemy is able to intercept a cover, but cannot ascertain any other information other than the cover message. The information is merely hidden; no security to the cover or embedded message is implied. A certain degree of security can be imparted by using keys to both encrypt the data before embedding it and as a seed for the hiding algorithm. The combination of these two techniques, shown in Figure 3, has become commonplace in steganographic systems.

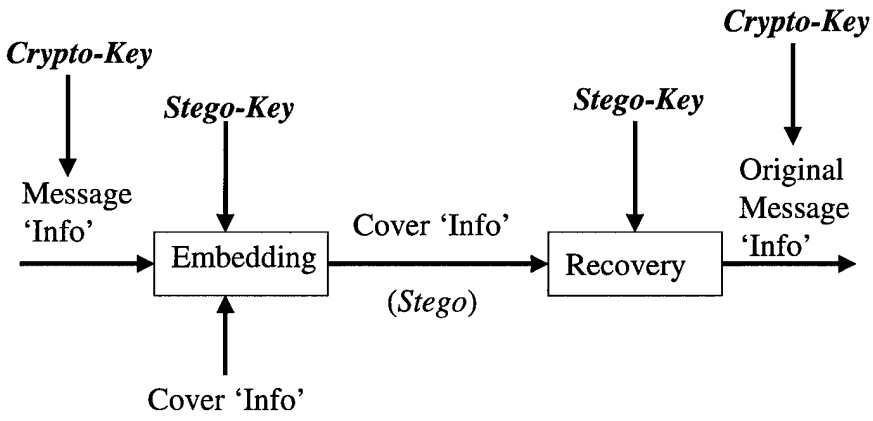


Figure 3, Combined Protocol

1.2 Historical Perspective

The ancient *art* of steganography did not limit the meaning of *cover* strictly to forms of communication. In fact, history is replete with examples where secret messages were concealed in the bodies of dead animals or tattooed on a servant's shaved head [9]. These techniques, although somewhat effective for small amounts of information, are certainly unproductive in today's information-hungry, digital world. A broad definition such as this indicates an important historical aspect of successful steganographic systems: they utilize cover media that does not call undue attention to itself. In doing so, the basic premise of steganography – concealing the presence of communications – is preserved.

Using this premise as a backdrop, it is not surprising that steganography has enjoyed a rebirth in the computerized world of today. As computers continue to pervade the daily routines of millions of people, their use as instruments of steganography makes perfect sense. Steganography exploits covers that are commonplace and mundane – a niche that computers fill in today's society.

Steganography's rise in popularity can also be attributed, in part, to the United States government's prohibition on the exportation of cryptographic material. The current basis for export control – the *Wassenaar Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies* – was adopted on 13 July 1996, and includes cryptography products on its export control list. Although this stance has been softened somewhat by a recent executive order that allows vendors to ship encryption products using 56-bit key-lengths worldwide, it has prompted some to use steganography as a means to conceal information from casual interception.

Another reason for the rise in popularity of steganography is the large size of the cover space provided by digital media, particularly in the various computer file formats. The available hiding space is more or less directly proportional to the size of the cover file. For example, a file that embeds a bit of information in every byte of cover could store a file that is one-eighth its size. This figure doubles if two bits are embedded per byte, and so on. The plethora of hiding places available to the modern steganographer is truly astounding.

The benefits of using steganography to conceal malicious logic (i.e. computer viruses and the like) are another reason for the increased attention in information hiding. Current computer attack methods include protocols for slipping Trojan software past virus detection mechanisms. The use of steganographic methods to conceal the presence of the malicious code could allow it to remain undetected until the attack is launched. The malicious code could be used to decode certain instructions, also possibly hidden via steganography in other files, and execute the attack. A similar protocol could be developed where the Trojanized code could be used to decode hidden messages that reside inside routine communication channels. In this way, the *message* and the *reader* would remain undetectable.

Finally, the foremost reason for the resurgence of steganography is the advent of digital watermarking. Based on the same principles as digital steganography, this promising technology is touted by industry as an anti-fraud and forgery godsend. Industries, particularly music and movies, have spent millions on techniques to hide company logos and other proprietary markings in digital images, videos, and music recordings. They have funded a majority of the academic research in steganography,

with an interest in creating robust, tamperproof digital *fingerprints*. Consequently, this anti-piracy technology has created collateral interest in basic steganographic techniques. Even though relatively few companies have begun marketing commercial steganography products, impressive non-commercial products have been released on the Internet.

As stated above, the majority of the research in both academia and industry has dealt with digital watermarking. While some research concentrates on making steganography more secure, pure steganography seems to have taken a back seat to the more profitable watermarking lobby. Within the Department of Defense (DoD), no significant unclassified research effort in pure steganography exists. If any other efforts do exist, they seem to have joined the ranks of other classified projects.

1.3 The Problem

Finding a role for steganography in the DoD is a simple exercise. Intelligence agencies will certainly benefit from hiding information from casual interception or observation. However, as with any offensive strategy, an equally important defensive strategy must exist due to the assumption that adversaries will eventually use the same methods against you. Cryptography has its antithesis in cryptanalysis, and so the term coined for attacks on steganographic systems is similar – steganalysis.

The intelligence community is not the only area that might benefit from steganalysis. Just as computers have permeated family and business settings, their use by criminal elements has been equally widespread. It was only a matter of time before criminal elements began using steganographic techniques to hide their information. While hiding information in unused hard disk partitions and partially used file segments is one popular form of steganography, the use of graphics image files to hide information

is only beginning to increase. Consequently, law enforcement agencies are becoming increasingly interested in the benefits of steganalysis as a computer forensics tool.

By definition, steganalysis encompasses discovering and rendering useless covert messages that are embedded in cover information [8]. It aims to achieve one of three goals:

- disruption of covert communications,
- detection of hidden channels, or
- recovery of an embedded message.

These three goals represent a logical progression. In the first – disruption of the communications – an adversary could choose to attack all forms of communications. In this case, it is not necessary to know for sure whether or not the covert channel exists, as disruptive countermeasures would be applied across the board.

The second goal – detection – is employed when economy of resources or prosecution of offenders is important. For instance, the results of detection efforts could benefit law enforcement personnel in gathering probable cause for further actions. Reducing the search space would permit law enforcement to concentrate its efforts against the targets of greatest opportunity and probability of success.

Finally, there is recovery of the embedded message. In order to make recovery efforts efficacious, the vast number of *possible* hiding places must be reduced to a much smaller subset of *probable* hiding places. Of the three goals, detection seems to be the key to successful and efficient prosecution of information operations designed to thwart an adversary's use of covert channels.

In terms of difficulty, the first goal of steganalysis – disrupting the channel – is relatively trivial to accomplish in the case of graphics image files. Since effective steganographic techniques do not radically change the appearance of the cover image, the same techniques can be employed repeatedly with no more adverse visible affects than the original embedding process. Disruptive attacks on suspected covert channels in graphics files are easily implemented with simple filters, lossy compression techniques, and *bit-substitution* methods.

Assuming a reliable method of detecting steganography existed, achieving the third goal of steganalysis – recovering the embedded information – could prove a monumental task. Since steganography is not inherently secure, adversaries combine cryptography and steganography to improve the security of the covert channel.

Encryption improves steganography and increases the complexity of recovering the covert message in two ways. First, it secures the information in the event that the covert channel is compromised. Second, the stochastic property of encrypted information produces a signature that is less likely to be detected through the use of frequency analysis. Encryption notwithstanding, the algorithms for selecting hiding places within graphics images can be computationally secure. Brute force methods would require extensive computer resources to overcome the time complexities involved.

1.4 Scope

There are virtually no limits to the number of possible methods available for hiding information in digital media. However, the use of graphics image files appears to be more common on the Internet than the use of audio or text files. The goal of this research is to explore avenues for future steganalysis research with the purpose of

detecting the presence of embedded information in a graphics file with an automated process. The focus of this research is the blind steganalysis problem where the original characteristics of the cover media and the underlying embedded information are unknown. This research concentrates on steganalytic methods that might be readily extended to an automated tool or process.

The graphics file format chosen for this research is the Windows© bitmap format (BMP). It is one of the most common raster data formats for graphics files and can be stored in an uncompressed state. Other bitmap formats exist, but the majority of them are compressed for storage. Since manipulating the image data requires it to be logically uncompressed, less preprocessing is needed. The BMP format accommodates various pixel depths – the number of bits representing each pixel. The pixel depth chosen for this research is eight bits due to its popularity and reduced storage requirements. This pixel depth translates to 256 ($= 2^8$) different colors. Finally, BMP is an extremely popular format with almost universal acceptance. Nearly every image processing application reads, writes, and converts BMP. [11]

As a side note, it is important to realize that the widespread use of images with a pixel depth greater than eight bits could possibly alert a forensics investigator to the presence of criminal behavior. Also, the use of a less-popular format might also flag files as suspect. Therefore, while the use of eight-bit images affords the least protection, their use is less likely to garner suspicion.

As mentioned previously, the focus of this research is on the blind steganalysis problem. Consequently, no assumptions are made about the content of the underlying embedded information or the cover file. Even though the original files are referenced to

determine the specific properties that change in the embedding process, the files are not selected on the basis of their content. However, this research is limited to using color images as cover files. The use of grayscale images as cover yields visibly better output in the eight-bit formats. However, grayscale images are not all that common on the Internet. Exclusive use of grayscale images may draw undue attention to an adversary's communications. This would violate the premise of steganography to keep the existence of the communications secret.

There are almost as many steganography tools as there are embedding techniques, and covering them all would be a monumental task. The three tools that were chosen for this research utilize the most common and effective technique – bit substitution. They were chosen for the quality of their output, their availability as shareware, and their ability to handle eight-bit graphics formats. Despite many similarities, the tools incorporate distinctive differences in the way they manipulate the cover file during the embedding process.

In summary, although this research is a broad examination of the blind steganalysis problem, it is conducted within reasonable limits.

- It addresses detection of embedded information and does not confront the problem of recovery.
- It uses the Windows© BMP format.
- It only operates on the output of three popular steganography tools.

Expanding the scope is discussed in Section 6.4, Recommendations for Future Work.

1.5 Approach

This thesis presents a logical sequence of events upon which future research in steganalysis can be modeled. The opening sections address background that is crucial to understanding the steganalysis problem mentioned earlier. Chapter 2 introduces graphics file formats and the principles associated with manipulating digital images. Intricacies of the BMP format are presented in the context of the Windows© BMP format specification. Chapter 3 covers basic definitions associated with image steganography and introduces various methods used to embed information in digital media. It also highlights several factors that influence the quality of the output from steganography. Finally, it presents background information on steganalysis.

The background information presented in the first two chapters provides hints to possible detection strategies. Three primary strategies evolve. The first is to explore signatures, or anomalies, induced by the steganography tool. The second is to examine the characteristics of the image data. The third strategy is to examine the characteristics of the entire file independent of the components that comprise a BMP file. Of these three strategies, the latter two are examined in depth. The first strategy, examining tool anomalies, represents a very limited application that would require reinvestigation each time a new tool is released.

Two techniques emerge as candidates for further investigation. The first technique examines the image data of a cover file after filtering it with a standard point-noise detection filter. A point-noise filter is a common filter from image processing and is typically used in edge detection algorithms. The filter technique is extended here to provide an indication of the presence of embedded information.

The second technique explored in this thesis examines the distribution of bytes in the cover file as a whole. An elementary byte frequency analysis is used to distinguish an original cover file from a manipulated copy. This part of the research provides a foundation upon which future file *signatures* can be identified. These signatures are key to detecting files that contain embedded information.

These two strategies are tested on a library of images that contain embedded information. The images were created using three steganography tools obtained as shareware from the Internet and commercial vendors. Image processing and data analysis software was used to examine the results of the tests. The results of these tests are compared against the results of the same tests applied to the original image files. Two metrics are developed which indicate possible embedded information for the given class of files. First, the amount of point noise in an image for a given sensitivity threshold was used in the first test. For the second test, the frequency of particular byte values between 0 and 255 were examined. Chapter 4, *Methods*, describes the two testing techniques and the specific steps in the data collection phase of this research.

II Graphics File Formats

2.1 Introduction

Research in steganalysis of graphics files requires a background in two key areas – graphics file formats, and steganographic techniques. This chapter introduces the first area – digital image representations and graphics file formats, specifically bitmap formats on which this research effort is based. The various methods of storing and representing graphics image data are explained, and a summary of the popular Microsoft® Windows bitmap format is presented.

2.2 Image Data Representation

Representing analog data in a digital environment is a classic problem in communications technologies, and has led to the development of various methods to translate between analog and digital. The ultimate goal of this process is to maintain as much of the original image information as possible. While the specifics of analog to digital translation are beyond the scope of this research, the basic principle involves sampling the continuous analog signal at a given frequency and quantizing the sampled signal to a discrete scale. The disparity between the analog signal and the digital signal decreases as the sampling frequency and the number of discrete quantization levels increases. A significant tradeoff is in the increased space required to store the digital

information. Today, the principal storage schemes for digital images are based upon two principle methods of representing image data – vector and bitmap formats. [6][10][11]

2.2.1 Vector Data

Vector data considers the image as a collection of objects such as polygons, lines, and curves. The image is translated into values that specify key points of the objects in the image. These key points typically include coordinates of intersections, box corners, or circle radii. Connecting these key points with others and placing the objects within the image according to some predefined grid renders the image.

Vector data requires attribute information such as line thickness, color, etc., as well as rules or conventions that aid a program in reproducing the desired image. It is popular with many CAD/CAM applications because it is easy to scale [11]. Vector data does not lend itself to the more common steganographic techniques and, as such, is not considered in this research.

2.2.2 Bitmap Data

Bitmap data is the most common data representation today [10]. It is a set of numerical values that specifies the color of individual picture elements in the image. A picture element, or pixel, is the smallest addressable element of an image and is obtained by first dividing the target image using a rectangular grid like that in Figure 4. The intensity (or color) value of the image at a particular grid coordinate is the associated pixel value. Reproducing pixels according to their spatial coordinates renders the image.

Bitmap data patterns its format on raster devices that use a series of rows, or scan lines, to display images. Unlike vector data, bitmaps are not easy to scale and sometimes require more memory or disk space to store and process. However, because each pixel

can be manipulated independently of the others, bitmap data is an excellent steganographic instrument. [11]

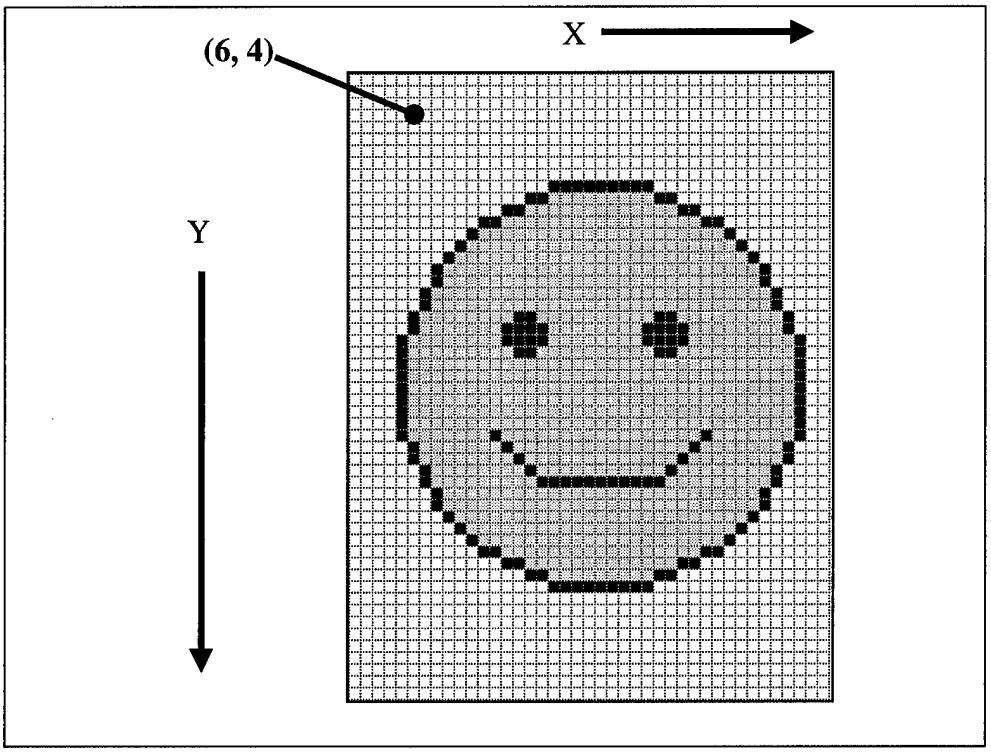


Figure 4, Image with Grid (Pixel) Overlay

2.3 Basics of Computer Bitmap Graphics

This section highlights several terms and concepts involved with bitmap graphics that are important to researchers working with steganography.

2.3.1 Color

Sir Isaac Newton was the first to discover that sunlight was not simply white, but rather a continuous spectrum of colors ranging from red to violet. He divided the color spectrum into six regions – violet, blue, green, yellow, orange, and red – and noticed that the transition between the regions was smoothly blended [6]. Later scientists would

eventually discover that different colors equate to different wavelengths of light, and that the color we see is actually the wavelength of light reflected off an object. An object that appears white is actually balanced in all visible wavelengths.

Central to the science of color is the characterization of light. Light is either chromatic or achromatic. Chromatic light is described using three attributes: radiance, brightness, and luminance. The first two are generally regarded as the color information of light, and the latter is the *amount* of light. Achromatic light is light that is described only by its amount, also known as intensity. *Gray level* is a term that is used to describe achromatic light. It is a scalar measure that ranges from black to white. [6]

2.3.2 Pixel Depth

Pixel depth refers to the number of bits used to represent a single pixel value and is sometimes called bit depth. The pixel depth determines the number of discrete levels (or colors) a pixel value can represent. In terms of color images, n-bits can represent 2^n colors. The most common pixel depths are 1, 2, 4, 8, and 24 bits. The pixel depth of a particular image is important in steganography because it influences the format of the graphics file and affects the choice of steganographic techniques. [11]

2.3.3 Color Model

Most output devices render specific intensity or color levels as a combination of several components. These components are usually some set of fundamental colors known as a color model. For instance, on color monitors, the pixel's color is a combination of red, green, and blue light sources. On color printers, the colors cyan, magenta, and yellow are used. Each of these color models defines color space within

which a device operates. Image data is usually specific to a particular color model and must be translated before it can be displayed on disparate color model devices. [11]

The purpose of a color model is to standardize the specification of color. It is a subspace within a three-dimensional coordinate system where every color represents a single point. One of the most common color models is the red-green-blue model or RGB. Each pixel value is comprised of a triplet that corresponds to a particular quantity of each of the three colors. It is convenient to think of these quantities as percentages of each color; for instance, (0%, 0%, 0%) might be black, and (100%, 100%, 100%) would be white.

In the Windows© bitmap format, eight bits are allocated for each color, so the values of each color range from 0 to 255. Zero represents the complete absence of that color, and 255 represents total saturation. As such, a 24-bit pixel value permits over 16 million (2^{24}) possible colors. [6][11]

2.3.4 Palettes

An image palette contains those colors or intensity levels that are used in an image. Palettes, also known as color maps or color tables, are primarily used to save space in the file [10][11]. Each pixel corresponds to a triplet that holds its color information. In order to represent the widest possible range of colors, a palette is used. It defines a set of colors in the image and is often represented as an array of three-byte tuples. The image data, then, is not the three-byte pixel value, but rather an index into the palette, as shown in Figure 5. Its pixel depth limits the number of colors used in the image. An 8-bit image such as the Windows© bitmap format has a maximum of 256 colors or palette entries. [11]

Palettes are not always optimal features to have because the benefit of using a palette quickly breaks down above eight-bit pixel depths. A palette with a pixel depth of 24-bits would have approximately 16.7 million entries – one for every possible color. Such a file would require over 50 megabytes of space for the palette alone. Consequently, palettes are generally only used for eight-bit pixel depths or less.

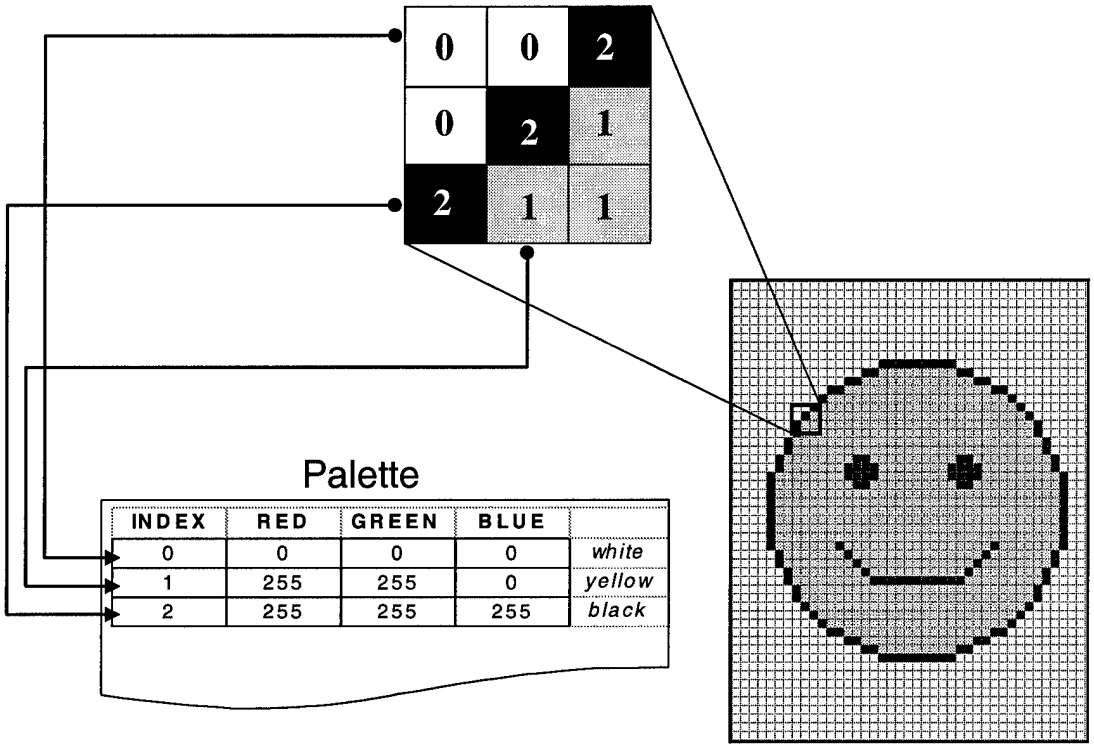


Figure 5, Image Data as Palette Index

2.3.5 Grayscale

The term grayscale refers to an image that contains only shades of gray. This is not to be confused with binary images that are composed of only black and white [10]. The appearance of a color pixel is a combination of two components: chrominance and

luminance. Chrominance combines the color information of radiance and brightness, while luminance is the brightness of the light [6][11]. A grayscale image, therefore, is a map of the intensity of each pixel relative to other pixels in the image. It is an image that has been rendered without its chrominance information [6]. The intensity of each pixel is translated to a particular grayscale value in the image's palette. A grayscale image's palette runs the gamut of color values, except the values for each of the three elements in a tuple is the same. For instance, the triple (127, 127, 127) is a grayscale value. However, the tuple (127, 127, 128) is not. Although the difference between the two triples is imperceptible to the human eye, the second triple is a color value and not grayscale.

2.4 Bitmap Graphics File Formats

Encyclopedia of Graphic File Formats [11] is an excellent source for information on graphics files standards since it contains over 85 different formats. Such a broad range poses a considerable challenge to any comprehensive work on digital images. For the purposes of this research, the Microsoft® Windows bitmap (BMP) format was chosen as the target format. This section presents background information on the BMP format that may aid in understanding the image processing techniques used in this research. Although there are two distinct versions of the BMP format, we will limit the discussion to the more recent and popular version, version 3.x (Version 3 bitmaps).

The format is designed for Intel-based machines, so the least significant byte appears in the lowest address in a word. The BMP format varies slightly according to the pixel depth of the image. It contains three mandatory elements – the bitmap header,

information header, and bitmap data – and an optional palette for that is not included with 24-bit pixel depth images.

2.4.1 Bitmap Header

The bitmap header is 14 bytes long and is the first element in a BMP file. The most noteworthy feature of the bitmap header is the first field. This two-byte value designates the file as being a BMP file. Other formats have similar identifiers, although adherence to the standard codes is purely voluntary. Table 1 shows the structure of the BMP header [11].

Table 1, Structure of Bitmap Header

ImageFileType	Always 4D42hex, or "BM" in ASCII.
FileSize	Size of the entire file in bytes.
Reserved1	Not used; always set to 0
Reserved2	Not used; always set to 0
ImageDataOffset	Varies according to palette size, if any. For a 256-color image, the offset is 1079.

2.4.2 Information Header

The next element in a BMP file is the information header. It is 40 bytes long and contains information about the image data that is used by applications to properly manipulate and display the image. The most applicable fields in the information header are the width and height fields, and the number of bits per pixel. Steganography tools use these fields to determine which bits to manipulate in order to minimize visual distortion. Table 2 shows the structure of the information header; Murray [11] should be consulted for further details concerning the information in contained in this header.

Table 2, Structure of Information Header

HeaderSize	Size of the information header; should equal 40 for v3.x.
ImageWidth	Width of the image in pixels.
ImageHeight	Height of the image in pixels.
NumberOfImagePlanes	Always 1.
BitsPerPixel	Pixel depth. Valid values are 1, 4, 8, or 24.
CompressionMethod	Bitmaps may be uncompressed or compressed using four or eight-bit run-length encoding.
SizeOfBitmap	Size of the (compressed) image data in bytes.
HorzResolution	Horizontal resolution – used in determining proper printing or displaying parameters.
VertResolution	Vertical resolution – used in determining proper printing or displaying parameters.
NumColorsUsed	Number of entries in the palette. If 0, then the palette is completely filled.
NumSignificantColors	Used to determine the most often used colors in the image in case the display cannot handle all colors.

2.4.3 Palette

The palette appears next in a BMP file that contains eight-bit pixel depth data or less. Those images that have a pixel depth of 24 bits do not contain a palette. The palette is a 256-element array of four-byte color values. The color model used in BMP is the RGB model described earlier, although the data is ordered as blue, green, red (BGR).

Each color component is one byte; an unused reserved fourth byte completes the four-byte tuple. Table 3 shows the structure of a BMP palette tuple [11].

Table 3, Structure of Palette Tuple

Blue	Eight-bit blue component.
Green	Eight-bit green component.
Red	Eight-bit red component.
Reserved	Not used – always 0.

It is important to note that the BMP standard does not specify an ordering scheme for palette color values, although it does provide recommendation [11]. For 16-color images, it recommends ordering the palette according to the frequency of occurrence for each color value; that is, those occurring most frequently appear first in the palette. In practice, a 256-color image might have a palette ordered by frequency of occurrence, intensity level, or even randomly. This subtle point is an important factor in creating visibly undetectable steganographic images. Ordering the palette by intensity level prior to embedding the information tends to produce the highest quality output.

2.4.4 Image Data

The final component in a BMP file is the image data. The current implementation of BMP allows only a single image in each file; it does not support animation. The image data in a BMP file is always aligned along byte boundaries. Each row, or scan line, is a multiple of four bytes in length and is padded when necessary.

When the image has an eight-bit pixel depth or less, the data in the image is an index into the palette, and it is ordered sequentially according to scan lines in the image.

The image is stored with the bottom most scan line first; that is, the lower left pixel in an image is the first value in the data. Consequently, most steganographic tools embed data into a BMP image beginning at the bottom of the image and proceeding upwards.

When the image has 24-bit pixel depth, no palette is used and each pixel requires three contiguous bytes. The data remains byte-aligned according to scan lines and stored in BGR order. [11]

III Graphics Image Steganography

3.1 Introduction

Once the intricacies of graphics file formats and image representation are understood, the next step is to formulate steganalysis strategies. Understanding the various methods used to embed information in graphics image files aids in this effort. This section focuses on steganography and highlights several unique methods of embedding information in both image and non-image files. It also presents several factors which affect steganography and discusses steganalysis and possible detection strategies.

3.2 Definitions

Over the last several years, an informal dictionary has begun to take shape in steganography. Its use is reinforced by the efforts of several leading researchers and the establishment of Information Hiding Workshops sponsored by Cambridge University in 1996 and Intel Corporation in 1998. Additionally, its widespread adoption is evident in the numerous papers and journal articles published in the past three to five years. This section presents important terms used in image steganography and throughout the remainder of this research.

3.2.1 Files

The process of embedding information in a graphic or text file usually involves two classes of files – cover and message files.

3.2.1.1 Message File

The message file is the information that is hidden or embedded during the steganographic process. It represents a broad range of information sources – voice, graphics, or text – depending on the type of information a user wishes kept secret. In reality, if the information is captured in a computer file, it can be embedded. The only technical restriction on the message file is that it be small enough to fit within another file. The hiding capacity of a steganographic technique is a function of the method used to hide the message and the size of the container file.

3.2.1.2 Cover File

The cover file is the medium that contains the message file after the steganographic process is applied. Once again, the intent of steganography is to maintain the initial visible quality of the cover file after the message file is embedded. Consequently, the file should not draw undue attention to itself; it should comprise features and characteristics generally found in other files of its particular class. The inherent properties of a file will dictate optimal steganographic techniques for use in given classes of cover files. Even though steganographic techniques exist for various types of cover files, this research uses only bitmap graphics files. It should be noted that cover files are also known as container files or stego-files. The latter term usually only applies to the cover file after the message file has been embedded.

3.2.2 Bytes and Bits

Bytes and bits that comprise the files mentioned above are also named according to their roles in the steganographic process. The terminology used to describe the bytes and bits is less standardized, but it is presented here as a basis for future reference. Any discussion of bits and bytes assumes bit substitution as the underlying steganographic technique.

3.2.2.1 Message Bits

Message bits comprise the message file and are embedded in cover files. The notion of message *bytes* instead of message *bits* does not make sense in the context of bit substitution, although the number of bytes in a message file is used to determine a cover file's ability to embed an entire message file. Message bits are sometimes referred to as *secret bits*.

3.2.2.2 Cover Bytes and Bits

Cover bytes and bits comprise the cover file and represent the set of possible bytes and bits that can be used for hiding information. As such, this definition does not apply to those bytes and bits which make up the headers and palettes of graphics image files, since only image data is used to embed message bits. Also, this definition does not designate the bytes and bits as having been selecting for hiding information.

3.2.2.3 Hiding Bytes and Bits

Also known as a container file, the cover file should appear as an innocent-looking file. This file may be a graphic image or text, and its bits are known as *cover bits*. The next type of file contains the information that needs to be hidden and is known

as the "*message*" file. The bits that comprise this file are known as *secret bits.* Not every cover bit is used to hide a secret bit, and the number of secret bits that can be successfully hidden is limited by several factors discussed later. Those cover bits that are substituted or otherwise selected to hide secret bits are called *hiding bits.* Generally, these bits are among the least significant bits in each word or byte – that is, they are least likely to cause any detectable distortion.

The hiding bit and hiding byte selection process is an area ripe with opportunities for optimizing and improving steganographic techniques. Several tools use a random pattern based upon a secret key supplied by the user, while others opt for an approach that alters bits at random before embedding the message file. All of these techniques attempt to disrupt easily discovered patterns and impart a degree of *security via uncertainty.* In the sections that follow, other techniques of bit and byte selection are presented which demonstrate the extent of techniques available.

3.3 Examples of Text and Image Data Steganography

3.3.1 Substitution Method

The substitution method hides information by replacing selected cover bits with message bits [2]. The technique is easy to implement. In order to set a hiding bit to '1', a bit mask of 0's is created with a '1' in the desired bit position. Then the mask is applied to the selected cover bit, or byte, using the logical OR operator. Alternatively, if a '0' is desired in the cover bit position, the mask used is the 1's-compliment of a mask of 0's and it is applied to the cover bit using a logical AND. The illustration in Figure 6 demonstrates this technique.

Original:		0001 1110	Original:		0001 1111
Mask:	'OR'	0000 0001	Mask:	'AND'	1111 1110
Resulting byte:		0001 1111	Resulting byte:		0001 1110

Note: The last bit is the desired cover bit (hiding bit).

Figure 6, Example Substitution Method

Replacing the bits is not difficult; this operation can be achieved with minimal code. What makes this method effective is the algorithm used to select the cover bit for substitution. One method is to substitute a bit in every cover byte with a bit from the message file. The algorithm used to select the hiding bit might call for every byte to be affected sequentially from the beginning of the container file, or the cover byte can be selected at random. Most implementations of the substitution method construct a function from some user-defined key or code. This key is then required for recovering the message from the cover file.

Least-significant bit substitution is the most popular steganographic technique employed with graphics image files. Despite its popularity, though, the substitution method is most likely to violate the *natural* order of bits in the cover file.

3.3.2 Selection Method

One way to preserve the original order of hiding bits in a cover file is by not embedding information in the first place. This technique, known as selection, is really more of a semaphore technique than steganography. It starts with repeated scans of an image and the manipulation of the image between scans via rotation or image enhancement techniques. Each unique image is stored and reduced to a number using a message digest algorithm such as a cyclic redundancy code or hash code. The resulting

numbers will be different for each image file due to the subtleties introduced between scans.

Next, some secret information that requires transmission such as a large number, or some other bit sequence similar to a private key is presented. An adversary could select a file from the set of those scanned that reduces to the secret number that requires transmission. The selected file is then transmitted in the clear. At the receiving end, the file is again reduced using the same message digest algorithm, and the secret number is recovered.

It is easy to see the benefits of the selection method since the secret information does not induce any statistical anomalies in bit or byte patterns of the cover file. In fact, Aura [2] describes this method as similar to a one-time pad in encryption. However, it is also easy to see that it involves a great deal of work to obtain a reasonably sized library of possible cover files. The information-carrying capacity – or the amount of information that can be transmitted in the cover file – is also quite small compared to other steganographic techniques.

3.3.3 Constructive Method

The idea of maintaining statistically normal properties of the cover file is not limited to the selection method. A largely theoretical process under investigation is known as the constructive method. It relies on a sophisticated model of the cover file or class of files. A message file is transformed to appear like a statistically normal file in the target class. At the heart of such a model is the concept of a mimic function [2][13]. In the context of steganography, a mimic function has the effect of changing a message file so it assumes the statistical *signature* of a cover file. In order for this method to be

successful, a method of obtaining a reliable signature of a class of files must be developed – a task as daunting as any in the area of steganalysis.

The constructive method works as follows. First, we must assume an accurate and reliable signature of a particular file class exists – in this case, a word processing document format such as Microsoft® Word. If an adversary has a digital image message file to transmit, he can transform the image file using a mimic function. The image file assumes the statistical characteristics of a Word document and likely escapes detection by an automated process. Of course, the file would draw suspicion as soon as an attempt is made to read the file using Word. Since enough work remains to be completed in determining file signatures, it seems unlikely a solution will result from anything short of an inordinate amount of work.

3.3.4 Parity Method

Parity embedding methods, like substitution methods, involve altering selected cover bytes. In this case, however, the parity of the selected byte is the steganographic element versus the value of its least-significant bit. The hiding byte is altered so that its parity matches the value of an associated message bit. Although the techniques are similar, the parity method has the added benefit that any bit within a hiding byte can be selected as a hiding bit. The determination as to which bit to manipulate is not fixed. This extra choice allows optimization of embedding algorithms in order to decrease distortion of the statistical properties of the cover file.

Figure 7 demonstrates how the parity method works. In this example, the parity of an associated byte is based upon the number of '1's in the hiding byte. A hiding byte with an even parity equals '0' and an odd parity equals '1.' The embedding process

simply matches a hiding byte's parity to the value of an associated message bit. Recovery, is a matter of retrieving the parity of the hiding bytes and reconstructing the message with the associated bit values.

Cover bytes:	10110011	10101100	11101000
Parity:	odd	even	odd
Secret Bits:	0	1	1
New cover bytes:	10110001	10101101	11101000
New parity:	even	odd	odd

Figure 7, Parity Method

This method has the advantage that in order to change the parity of the hiding word, any bit in the word can be flipped. The bit should still be one of the least significant bits to avoid disrupting the rendered image, but it does not have to be the *very least* significant bit.

It is easy to see from the example that the parity method is still likely to distort the image as in the substitution method. The strength of this method lies in the uncertainty over which bit was changed in the hiding byte. Taking this one step further, if the statistical characteristics of the container file are accurately calculated, the hiding bit can be selected to minimize deviation from these characteristics. Used in this manner, the parity method can be more secure than a straight-forward substitution. [1]

3.3.5 Word and Line Shift Encoding

Up to this point, the focus has been on methods of embedding information in digital images that can only be retrieved in electronic form. The embedded information

cannot be extracted once the file is rendered (e.g. printed or viewed on a terminal). This section presents a unique method of marking documents and recovering them after the document is rendered. Although this method is not currently useful for digital images, it highlights how truly creative the developers of steganography techniques can become.

3.3.5.1 Technique

Researchers at AT&T Bell Laboratories [3] [4] [5] developed a process known as shift encoding which uses spatial information to hide data in document images, such as Postscript documents. This process cannot be used on text word processors since the positioning of the document's tokens – characters, words, lines, etc. – is not precise. Graphic *images* of documents, on the other hand, have exact coordinates for each token. They are very similar to vector image formats.

The shift encoding method is simple and intuitive, yet it maintains its reliability even after rendering by some of the noisiest devices in operation – low resolution fax machines, plain paper copiers, and digital scanners. In a very simplified form, the method involves altering the position of characters, words, or lines in a document by a very minute amount – 1/150 of an inch. The alteration is essentially invisible to the untrained eye. However, when it is overlaid with the original image, the altered tokens stand out from the surrounding text.

The researchers have characterized the methods according to the token that is manipulated. *Word-shift encoding* shifts characters or words horizontally on a line of text, *line-shift encoding* involves moving entire lines of text vertically, and *feature encoding* alters the font of certain characters to make them different from other similar letters [3][4]. Of course, decoding the document to retrieve the embedded information

does not actually involve overlaying the original image. Instead, the researchers developed precise measurement techniques to accomplish the same feat.

Decoding the document image involves scanning it and measuring the minor alterations relative to their unmanipulated neighbors. Each token has a center of mass known as a centroid. By measuring the centroid's deflection left or right of the normal position, the altered token can be detected. An extension of this technique returns a different bit value depending on the direction of the deflection (e.g. left is '1', right is '0'). In order to recover the tokens reliably, only lines that extend the full width of the page are selected for shifting. Also, top and bottom lines, and words at either end of a line are exempt from shifting.

3.3.5.2 Robustness

Although these techniques are remarkable, they are far from foolproof. Feature encoding is the least robust method since noisy reproduction devices often blur characters and induce other abnormalities which might make an *unaltered* character appear *altered.* Word and line-shift encoded documents are sensitive to stretching which occurs when the document is photocopied [4]. The direction in which the paper goes through the copier determines the skew. Paper that proceeds lengthwise stretches the image vertically, while an edge-first copier stretches the text image horizontally. Stretching effects the precision of the centroid measurements. However, of the two shift encoding methods, line-shift encoding is more reliable since line spacing is assumed to be constant throughout a document. Because of this unique property, line-shift encoding is recoverable without use of the original document.

3.3.5.3 Application

The amount of information that is embedded using shift encoding is significantly less than other steganographic techniques, but its application with rendered documents makes it an ideal technique for encoding copyright or registry information. A typical scenario might begin when a document is requested from an organization that needs to protect its documents. The owner of the document verifies the identity of the requester and produces some unique code from the requester's personal information. This identifying information is then embedded in the document prior to its transfer to the requester. Should the authenticity or origin of the document ever be questioned, the embedded information is retrieved to provide verification – or evidence, depending on the situation.

3.4 Factors Affecting Steganography Using Bitmap Graphics Files

The four examples presented above highlight a few clever methods in steganography tools available today. The most popular method employed with graphics file formats utilizes bit substitution, of which least significant bit (LSB) substitution is the most common. The LSB substitution method is an easy one to employ and is quite effective if the proper preprocessing is done to the cover image. Such preprocessing includes manipulating the colors in the palette, compressing the hidden message prior to embedding, and inserting random noise throughout the image. The following sections present several factors that affect steganography. They include techniques that can be employed to improve the quality of the steganographic output and make it more difficult to detect the presence of embedded information. Each of these techniques assumes LSB substitution as the embedding technique.

3.4.1 Increase Pixel Depth

Perhaps the most effective method of increasing the visible quality of a steganographic image is to utilize 24-bit pixel depth. This is easily achieved by converting the image to a 24-bit image before embedding the message file. The most overwhelming reason why this method is so successful is due to the absence of the palette.

As mentioned earlier, 24-bit images do not contain a palette; rather, the pixel values are the RGB tuples. When the LSB of a pixel is changed, it changes the index into the palette by one place. Since the palette is not ordered in any specified manner, there is no guarantee that the new pixel value will be visually similar to the original color. The resulting steganographic image can appear somewhat grainy. However, when 24-bit images are used as cover, the altered LSB only increases or decreases the individual color component – either red, green, or blue – by a very small amount. The result is analogous to changing the pixel color from red to reddish. The human eye cannot readily detect such minute changes.

Another reason 24-bit images are superior to lesser pixel depths is the shear number of possible hiding bits. The amount of information that can be embedded in a 24-bit image is considerable as Figure 8 shows. Even these numbers are conservative since they assume only one bit per bytes is altered. In reality, two or three bits can be changed per byte without yielding significant distortion. Unfortunately, the larger file sizes might draw attention or suspicion if compression is not used.

Assumes 640x480 image.

$$Capacity = \frac{(\#\text{of pixels}) \times (\text{bytes per pixel})}{8 \text{ bits per message byte}}$$

```
= 38,400 bytes (8-bit cover)
= 115,200 bytes (24-bit cover)
```

Figure 8, Cover File Capacity

3.4.2 Grayscale Covers

Grayscale is important in steganography because the output from embedding messages in grayscale images has no visible distortion. Remember from the previous section that the embedding process most often results in a new pixel value that indexes a color above or below the original palette entry. Depending upon the original ordering of the palette, the color directly above or below another is not necessarily similar. By contrast, grayscale palettes are ordered sequentially by intensity level. Since grayscale has 256 levels, the difference between each adjacent byte value is too small to detect. In fact, the positive effects of ordering a palette are not restricted to grayscale palettes. Any time a palette is ordered in such a way that adjacent pixel values are similar, the amount of visible distortion is drastically reduced.

3.4.3 Message Compression and Encryption

In terms of improving the visible quality of the output, both compression and encryption of the message file accomplish similar results. The idea is to randomize the bytes in the underlying message to interrupt frequency patterns in the altered bits. Such patterns would appear as bands of distortion throughout the output image and can indicate specific underlying file types. For instance, text files could be characterized by blocks of unaltered bytes due to the amount of '0' bits in ASCII character codes, while

binary files could have less distortion due to the broader range of byte values. The lack of pronounced or definitive frequency patterns makes detection by byte frequency analysis less conclusive.

Encryption has the added benefit of providing additional security to the underlying message. When used together, steganography and cryptography are an effective way to secure covert communications channels.

3.4.4 Cover Image Compression

Unlike compression of the message file, compressing the cover image can be disastrous to the integrity of the embedded message. This is especially true if an incorrect compression technique is employed. Some techniques are lossless; that is, 100 percent of the original image data is available after compression. Examples of lossless compression techniques include run-length encoding (RLE) used in the Windows® compressed bitmap format and the Lempel-Ziv-Welch (LZW) algorithm used in the popular graphics interchange format (GIF). [12]

In contrast, other methods compress the image using transforms such as the discrete cosine transform. These methods are lossy because some of the original image data is rounded-off during the transform. When the image is expanded, the result is a best guess image that can appear visibly distorted due to propagated rounding errors. An example of lossy compression is the Joint Photography Experts Group (JPEG) standard. [12]

Cover image compression is an effective attack on steganographic channels. The same techniques that are able to maintain the original quality of the cover image can be applied repeatedly without excessive degradation.

3.4.5 Random Noise

One steganographic technique that is used successfully in a variety of covers is to embed the hidden message in the noise component of the cover signal. Unfortunately, if less-sophisticated methods are used to embed the message, the noise can be analyzed for covert channels. Truly random noise has no apparent pattern, but embedded information can induce a signal pattern that is both easily detected and recoverable. In order to avoid this problem, one would only need to model the noise component of the cover signal and embed the information in such a way that the randomness of the cover noise is not significantly violated. Such a principle is also applied to graphics cover files.

Sophisticated embedding techniques include a preprocessing step wherein the cover image is analyzed for areas with a greater variance of pixel values. Since these areas are characterized as busy areas in the image, distortion would be less noticeable than in areas of solid color or lower variance. The message is then embedded in the pixels that comprise these areas. These techniques are commonly employed with digital watermarks since the image is less distorted and the watermark is harder to detect. Such preprocessing is not often used in general image steganography.

A related method that is often used in steganographic tools is the insertion of random noise in the image data prior to embedding the message. This serves two purposes. First, it interferes with the underlying image byte frequencies under the assumption that a majority of the altered bits will be restored to their original values. The output appears less distorted than if the bits were not previously altered. Secondly, it masks the telltale demarcation between altered and unaltered pixels created when the data stream from the message file is exhausted. Such a line in the cover image would

certainly be noticeable and aid someone trying to retrieve only those bits that make up the embedded message.

3.5 Steganalysis

After reviewing requisite background information in graphics image files and steganography, several target-rich areas emerge for further examination into steganalysis strategies. This section presents three such areas that might warrant further scrutiny. They are:

- Examination of anomalies or signatures left by the steganography tool.
- Examination of the visible distortions to the image.
- Examination of the file characteristics.

3.5.1 Tool Anomalies

3.5.1.1 Flags

Of the three areas listed above, perhaps the most definitive metric indicating the presence of embedded information is a flag introduced by the embedding tool. If such a byte value or pattern could be readily identified, it would provide a definitive means of detection. It could even indicate the use of a particular tool or version of software. In this case, automated detection tools would almost certainly behave similar to virus detection tools.

The notion that a steganography tool would leave behind an explicit flag, though, contradicts the basic premise of steganography – to go unnoticed. As such, it is unlikely that such an obvious marking scheme is incorporated into any steganography tool worth using. In fact, the three tools used in this research do not mark files in such a way that the tool recognizes them as output files.

3.5.1.2 Message Recovery Data

On the other hand, tools must be able to recognize passwords and the selected hiding bits that were identified during the embedding process. This is often accomplished by hiding the message recovery information in predetermined locations within the image data. The choice of the area is certainly not arbitrary. For instance, embedding information within the header can render the file unreadable – again, drawing attention to the file. The palette of the Windows© BMP file, on the other hand, offers extensive space in the fourth byte of the RGB-tuple. However, even though this area is currently unused, it is typically initialized to zero. Hiding information here could, again, draw attention to the file.

The only other area left in the file is the image data, and it is in this area that most tools hide the message recovery data. The message recovery information varies among tools, but it usually consists of the message file's size and the random number seed used in the hiding bits' selection algorithm. More sophisticated tools might also include the tool's version number for compatibility between the embedding and recovery processes, the original message file's name, or even a user-selected password. The use of the password also varies among tools. It can be hashed into a number and used as the seed for selecting hiding bits, or it can be used as a key for encrypting the message file or message recovery data.

As expected, the specific location for storing the message recovery data within the image data also varies among tools. The location is most often hard-coded into the application, although other schemes are certainly possible. The properties of the cover file, such as the file size or creation date could be used as keys for locating the message

recovery data. However, regardless of the specific location or selection method used, if the location could be reliably determined, then the message recovery data could be retrieved and possibly exploited.

3.5.1.3 Application

Unfortunately, locating tool anomalies can be a lengthy process. It is also a process that is especially vulnerable to the traditional cat-and-mouse security game. When news of a tool's exploitation is announced, the exploit is usually fixed and a new version released. The process of determining new anomalies must be revisited each time a new tool is release. For this reason, using tool anomalies to detect the presence of embedded information generally has limited applications.

3.5.2 Image Distortions

The output from many eight-bit steganography tools can appear visibly distorted. But visually inspecting each image is time-consuming, if not impractical, especially when hundreds of images are involved. However, if the distortion is measured quantitatively, then a tool can be developed to process each image and look for the distortion in the image data. The first step in determining a distortion metric is to become acquainted with the types of distortion that can result during the embedding process.

3.5.2.1 Granularity

The most common distortion encountered is an increase in granularity of the output image, and its severity is affected by several factors. The most dramatic distortion occurs when the image's palette is ordered by a color's frequency of occurrence in the image. Recall from Section 3.3.1, the new value of an altered hiding byte indexes a color value directly above or below the original color. Since there is no guarantee that the new

color is similar to the old color, the new color value may be radically different than the surrounding pixels. The image will appear grainy. However, when the palette is ordered by luminance or grayscale level, the distortion is less noticeable since the new pixel value is similar to the old value. The figures below show the output from a typical steganography tool (*HideSeek v1.1 Win95*) using eight-bit images. Note that the most distorted output occurs when the palette was ordered by the color's frequency of occurrence in the image.

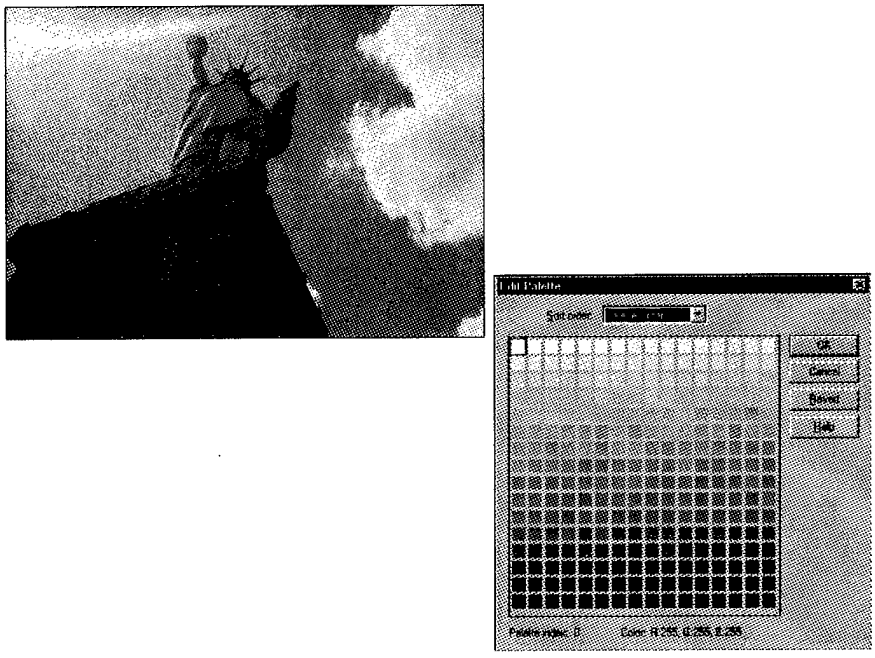


Figure 9, Stego-image with Luminance Ordered Palette

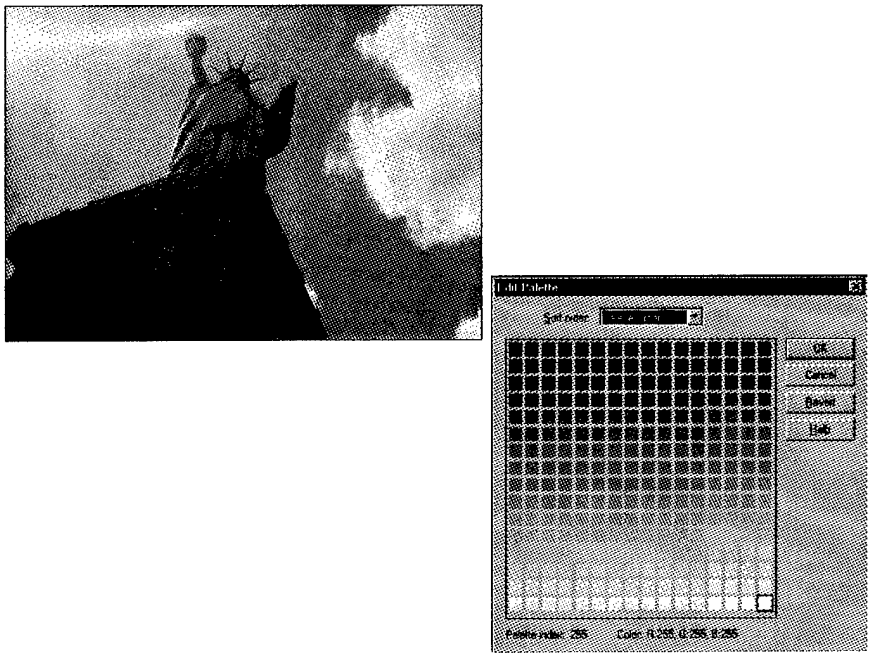


Figure 10, Stego-image with Grayscale (Intensity) Ordered Palette

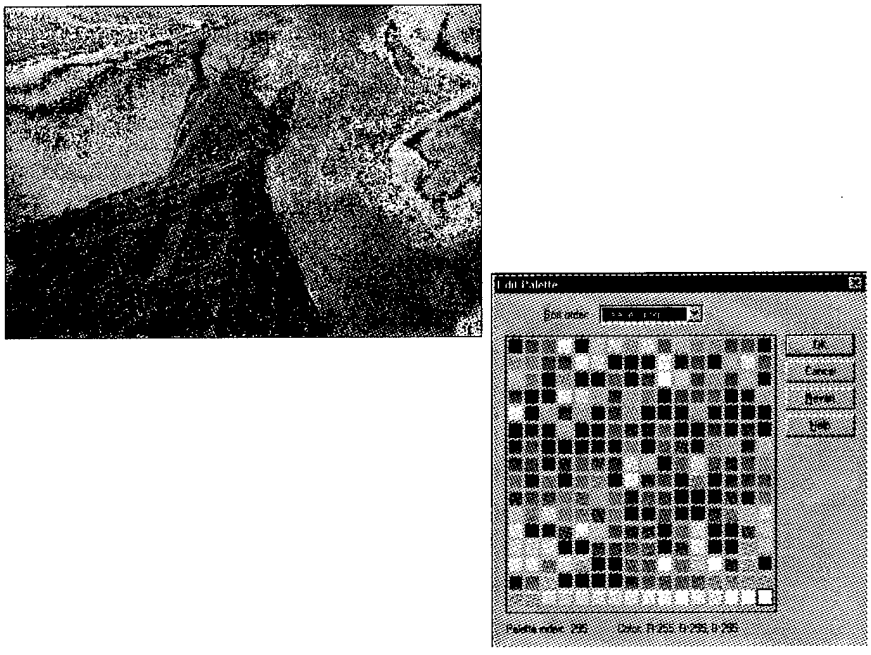


Figure 11, Stego-image with Frequency Ordered Palette

3.5.2.2 Pixel Stuffing and Cropping

A less common image distortion can occur when a cover file is either too large, or too small, for a given message file. This is undoubtedly a result of poor or incomplete programming techniques, and it is rarely found in current steganography tool releases. In the case where a cover image is too small for a message file, portions of the original image are replicated until the cover image is large enough to accommodate the image. Conversely, if the cover is too large, the image is cropped to the size necessary to accommodate the message. Both cases can result in an image that is grossly distorted; it is no wonder these tools are rarely used.

Another malady that plagued an early – if not prototypical – steganography tool (*HideSeek v5.0 for DOS*) was a fixed output image size. The program was designed for a single output file size; however, instead of replicating portions of the original image, it simply appended black pixels to the image. The resultant image possesses a noticeable black border. Figure 12 below shows the resultant 800 x 600-pixel image created using 512 x 512-pixel cover. [7]

3.5.3 File Characteristics

While examining images for distortion focuses on changes to the image, a third area for possible exploitation is examining the characteristics of the BMP file as a whole. Characteristics that deviate from normal files of the same class might include aberrations that affect composition of the palette entries, inconsistencies between header fields and the actual values, or the byte frequency distribution of the file.

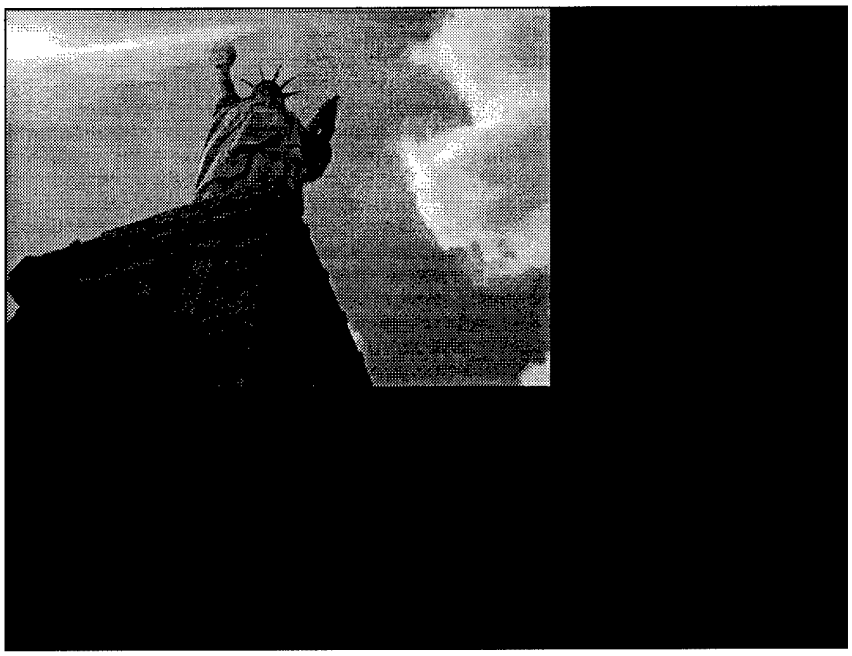


Figure 12, 800x600 Stego-image with Bit Stuffing

3.5.3.1 Palette Composition

Since palettes do not contain image data, some might conclude they are unimportant in steganography. However, this is certainly not the case. Many steganographic tools manipulate the palette to reduce visible distortion induced during the embedding process. Consequently, examination of an image's palette might offer clues to the presence of embedded information.

3.5.3.2 Header / Image Inconsistencies

Other possible indicators of embedded information are inconsistencies between the information in the header and the actual properties of the image. For instance, some steganographic techniques can result in an increase in the number of colors in an image. If the tool's author does not update the *NumColorsUsed* field in the header, then a simple comparison of the stored value and the number of colors in the image data would reveal

tampering. The Height and Width fields might also be used when the tool crops or pads the image data.

3.5.3.3 Byte Frequency Distribution

Examining the frequency of occurrence of a file's byte values might also offer clues to the presence of embedded information. It would seem reasonable that a file class could be categorized according to the distribution of the bytes that comprise the file. It is especially relevant when a significant portion of the file is byte-oriented like the image data in a BMP format file. In the case when a palette is ordered by a color's frequency of occurrence, the pixel values in the image data would be skewed toward lower values. Consequently, the file's average byte values would also be lower. If the embedding process alters the *typical* distribution of bytes, the presence of embedded information might be detectable.

IV Methods

4.1 Introduction

4.1.1 Problem Definition

Most steganography tools available today introduce changes to the cover file as a byproduct of the embedding process. Manually examining files for these subtle changes is a labor-intensive exercise, especially in the case where the characteristics of the original file are unknown. Automating this process would reap significant savings in time and manpower. An investigator could channel energies toward a subset of cover files that have a greater likelihood of containing embedded information.

No detailed information is available in the public domain that reports successful automated techniques for steganalysis of graphics files. An automated test or series of measurements is needed which can be used to provide a binary output indicating the presence of embedded information. A reliable *pass-fail* determination would successfully partition a larger search space into a more manageable one. It is also a logical first step in the recovery of the embedded information.

When eight-bit graphic images are used as cover, the resulting image is often visibly distorted. Therefore, an automated process that effectively recognizes characteristics of these distortions would be advantageous in combating the blind steganalysis problem. Another side effect of the bit-substitution process is a change in

byte values that comprise the image. Again, automating the process of determining the differences in the distribution of byte values for a given file class would also aid in blind steganalysis.

4.1.2 Problem Statement

This research attempts to determine strategies or tests that can be used to reliably determine the presence of embedded information in eight-bit graphics images. Emphasis is placed on techniques that can be readily employed in an automated tool or process, and methods that are minimally specific to a particular steganography application.

4.1.3 Scope

4.1.3.1 Selected Strategies

Three strategies were presented previously: examination of tool *signatures*, visible distortion, and file characteristics. The first strategy is too specific to be of considerable use in the general steganalysis case, since the release of new tools could require a lengthy re-investigation process. However, the latter strategies are applicable to a wider range of cases. This research explores methods that could exploit resulting image distortions and the differences in file byte frequencies between embedded and organic (untouched) image files.

4.1.3.2 File Format

As presented in Section 1.4, this research is limited to observing the effects of embedding information in eight-bit Windows © Bitmap (BMP) files. While many other formats exist, the BMP format was chosen due to its ease of use and small storage requirements. Despite the differences in format specifications, though, the image data and palette information are logically identical to other popular formats.

4.1.3.3 Image Library

The master image library consists of thirty bitmap images available on the Internet or scanned from personal photographs. They were not selected on the basis of their underlying picture composition. In order to standardize the image size and provide a level of control in the experiment, a target image size was arbitrarily set at 192,500 pixels. This equates to an image that is 550×350 pixels in size. Images were selected only if they were within five percent of the target pixel count. The minimum and maximum allowable image sizes were 182,875 and 202,125 pixels, respectively. In order to avoid *unnatural* distortions, the images were not subjected to shrinking, stretching, or other image processing techniques. Table 4 lists the individual file statistics.

4.1.3.4 Steganography Tools

Evaluation copies of the selected steganography tools were obtained from Internet sources. They were selected on the basis of the quality of their output, ease of use, and ability to use BMP format files as cover files.

HideSeek v1.1 for Windows95 – *HideSeek v1.1* for Windows 95 is a BMP-based steganography program created by Colin Moroney and available as shareware on the Internet. This version is a significant improvement over previous versions that were DOS-based and used CompuServe's GIF ® format.

The hiding technique is least-bit substitution, but it uses a pseudo-random process to flip non-hiding bits in order to make unauthorized recovery more difficult. *HideSeek* uses an optional password or phrase – up to 56-characters – to encrypt recovery information using the Blowfish encryption algorithm. Such information includes the

pseudo-random seed, the length of the file, and *HideSeek's* version number. This information is then inserted as a header at the beginning of the image data.

HideSeek does not provide an option to encrypt the message file. Due to *HideSeek's* pseudo-random bit-flipping process, the output stego-image appears distorted even when no data is hidden. Therefore, the size of the message file has little visible impact on the quality of the stego-image.

Steganos v1.0a for Win95 – *Steganos v1.0a* is a quasi-commercial cryptography and steganography program marketed by Fabian Hansmann / Sascha Wildgrube / Gabriel Yoran GbR (Deus Ex Machina Communications) of Frankfurt, Germany. Unlike *HideSeek*, *Steganos* uses a variety of cover file formats, including BMP, ASCII, HTML, and WAV audio files. However, like *HideSeek*, it uses least-bit substitution and pseudo-random *bit flipping* to embed message files.

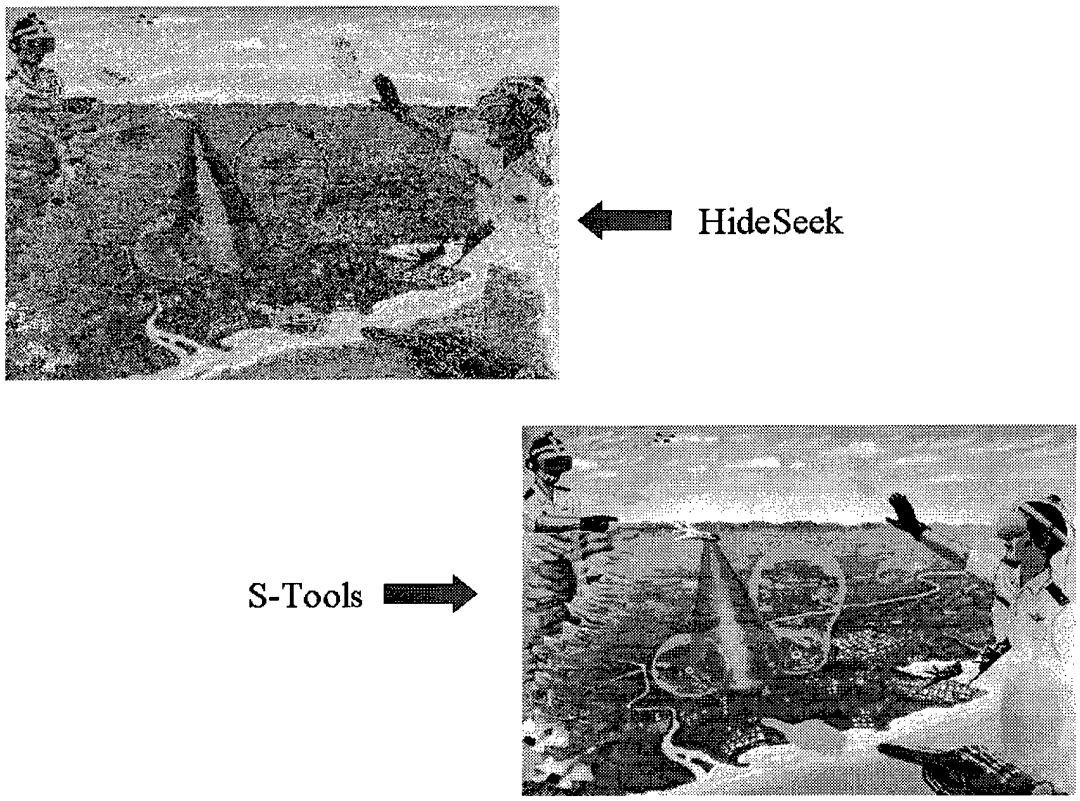
In addition, *Steganos* compresses the message file before embedding it using the Lempel-Ziv-Welch algorithm. This allows a considerable quantity of information to be hidden in a relatively small cover file. It also makes frequency analysis of the message file bytes more difficult. The user also has the option of encrypting the message file using the HWY1 algorithm, which is purported to be like RC4 ™.

The quality of the output stego-image is similar to *HideSeek's*. An interesting feature of *Steganos* is the option to convert 8-bit cover images to 24-bit prior to hiding. Although this creates a larger cover file, the quality of the output is practically indistinguishable from the original cover image.

Table 4, Master Image Library

Image	Width	Height	Image Size (Pixels)	Percent of Target Size
01	540	346	186,840	97.06%
02	540	365	197,100	102.39%
03	558	358	199,764	103.77%
04	524	372	194,928	101.26%
05	601	332	199,532	103.65%
06	558	362	201,996	104.93%
07	550	352	193,600	100.57%
08	572	324	185,328	96.27%
09	541	366	198,006	102.86%
10	540	360	194,400	100.99%
11	390	470	183,300	95.22%
12	436	453	197,508	102.60%
13	400	500	200,000	103.90%
14	526	357	187,782	97.55%
15	524	365	191,260	99.36%
16	388	487	188,956	98.16%
17	516	356	183,696	95.43%
18	522	359	187,398	97.35%
19	519	357	185,283	96.25%
20	524	355	186,020	96.63%
21	522	351	183,222	95.18%
22	528	357	188,496	97.92%
23	352	526	185,152	96.18%
24	520	354	184,080	95.63%
25	545	370	201,650	104.75%
26	536	364	195,104	101.35%
27	519	355	184,245	95.71%
28	536	364	195,104	101.35%
29	519	356	184,764	95.98%
30	523	355	185,665	96.45%
Average	512	378	191,006	99.22%
Max	601	526	201,996	104.93%
Min	352	324	183,222	95.18%

S-Tools v4.0 for Windows – *S-Tools v4.0* is a freely available steganography tool that hides files in BMP and GIF graphics files, and WAV audio files. *S-Tools* provided many user options including encryption and compression. Even though *S-Tools* uses least-bit substitution and pseudo-random dispersion of hiding bits, the quality of the output is extraordinarily good when compared to the previous tools. Figure 13 shows the visible output of *S-Tools* compared to that of *HideSeek.*



Artwork by Gene Lehman, Air Force Institute of Technology

Figure 13, Comparison of Stego-image: HideSeek vs. S-Tools

These remarkable results are due to a unique preprocessing technique employed by *S-Tools* on eight-bit color images. It is based upon the fact that an image with only 32 unique colors will never exceed 256 colors as a result of switching the least significant bits in each of the RGB components that comprise a color. Therefore, it first reduces the

number of colors in the original cover image to as close to 32 colors as it can. It does this through a proprietary color-reduction method based on a median-cut colormap generator. The help file that accompanies *S-Tools* contains more information and should be consulted for further reference.

After the number of colors has been reduced, *S-Tools* inserts other colors in the palette that are very similar to the reduced set. These newer colors are appended to the reduced list, and the palette grows in length. The outcome is a series of groups comprised from six to eight similar colors. In this way, flipping the least significant bit results in a new palette index that points to a color that is similar to the original. The output is similar to the results achieved using a grayscale or luminance–ordered palette.

Unfortunately, the new palette contains *blocks* of similar colors that vary by only a bit among the three RGB components. In Figure 14, a palette from a luminance-ordered file is shown. After the file is embedded using *S-Tools*, the palette is changed to that shown in Figure 15. The groups of very similar colors becomes more noticable when the *S-Tools* palette is reordered by luminance value, as shown in Figure 16. This signature is unique to *S-Tools* stego-images.

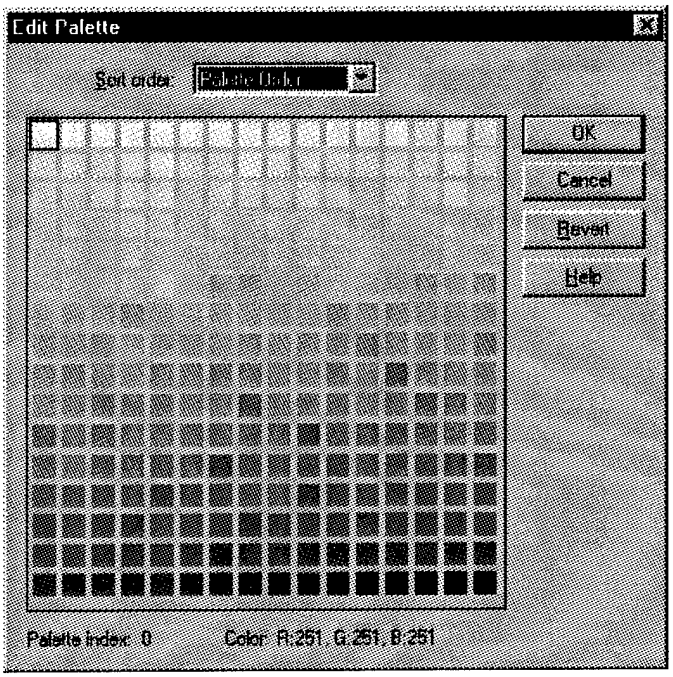


Figure 14, Original Palette

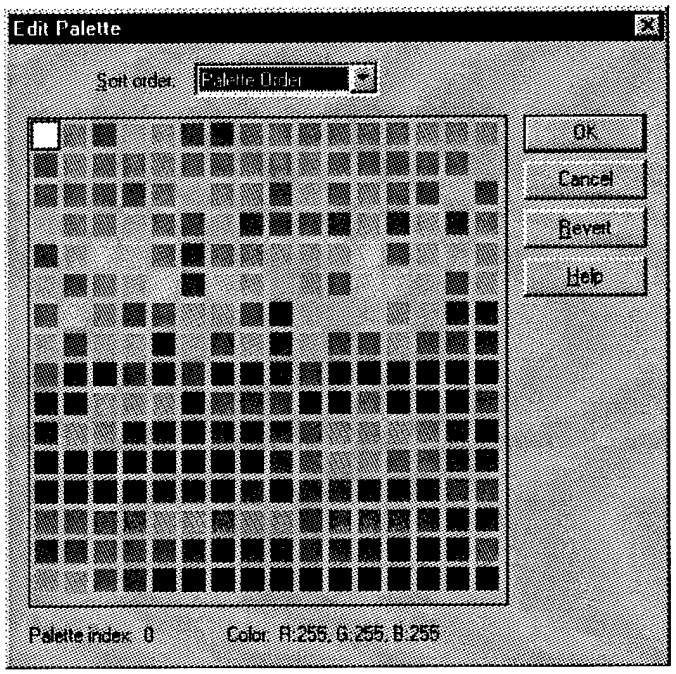


Figure 15, Palette after Embedding with S-Tools

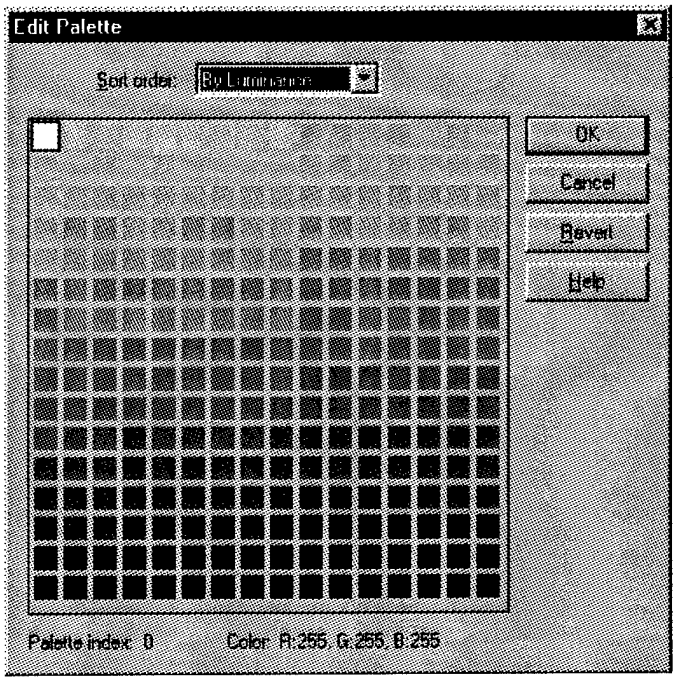


Figure 16, S-Tools Palette Sorted by Luminance

4.1.3.5 Test Parameters

A preliminary examination of factors that affect the output of steganography tools indicated three candidates for further scrutiny.

- method of ordering the palette
- characteristics of the underlying message file
- amount of embedded information (loading level)

These factors are the basis for partitioning the experiment into unique test cases. In order to vary the first parameter, the master image library was converted into two minor libraries. One library contained the master images with palettes sorted by luminance, and the other contained the same images with palettes ordered by frequency of occurrence in the image.

The second parameter was manipulated by selecting three different file types as message files. Each of these file types was selected because their byte frequency and distribution characteristics differ. The three classes of files are binary (executable), image, and text files.

The loading level parameter was achieved by selecting message files that were in the first and fourth quartile; that is, approximately greater than 75% and less than 25% of the average image size in pixels. These values represent high loading and low loading, respectively. Both large and small files were selected for each of the above message file types. Table 5 shows statistics for the message files. The value for the approximate loading level assumes one message byte requires eight cover pixels. Since a user cannot disable the compression option in *Steganos* v1.0a, the loading levels are not maintained during the trials using this tool to embed message files. Table 6 shows the actual loading levels for the *Steganos* trials.

Table 5, Statistics for Message Files

File Type	Size (bytes)	Approx. Loading (avg. file size = 191,006)	Approx. Loading (max. file size)	Approx. Loading (min. file size)
Large Binary	22,016	92.21%	87.19%	96.13%
Large Image	20,118	84.26%	79.68%	87.84%
Large Text	21,973	92.03%	87.02%	95.94%
Small Binary	5,120	21.44%	20.28%	22.36%
Small Image	5,800	24.29%	22.97%	25.32%
Small Text	5,754	24.10%	22.79%	25.12%

Table 6, Loading Levels Using Steganos Compression

File Type	Size (bytes)	Compressed File Size (bytes)	Approx. Loading (avg. file size = 191,006)	Approx. Loading (max. file size)	Approx. Loading (min. file size)
Large Binary	22,016	10805	45.26%	42.79%	47.18%
Large Image	20,118	14,542	60.91%	57.59%	63.49%
Large Text	21,973	9,823	41.14%	38.90%	42.89%
Small Binary	5,120	1,355	5.68%	5.37%	5.92%
Small Image	5,800	3,051	12.78%	12.08%	13.32%
Small Text	5,754	2,603	10.90%	10.31%	11.37%

4.1.3.6 Test Cases

The three parameters mentioned previously form the basis for partitioning the experiment, as shown in Figure 17. The two test partitions are further divided into 18 unique test cases that account for the application of three different steganography tools. Each tool is used to embed three types of information – binary, image, and text files – at low and high loading levels. Table 7 outlines the specific test cases for each partition. In all, 1,140 image files are examined in this research for each threshold tested.

4.2 Method of Evaluation

4.2.1 Process Overview

The selected steganalysis techniques are applied to each test case in the two partitions. The byte-frequency test is only conducted once on every test file and original image file. The results from the stego-files are compared to the non-stego image files. The point-noise thresholding test is performed once for every test case at each target

threshold. The order in which the tests are conducted is irrelevant, since they are independent of one another and do not alter the file being examined.

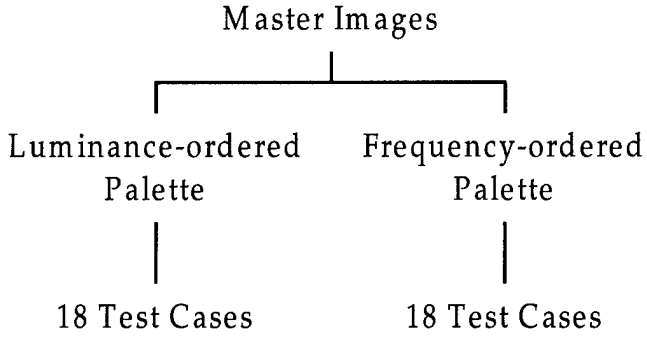


Figure 17, Experiment Partitions

4.2.2 Controls

4.2.2.1 Cover Files

The composition of the 36 test cases provides adequate samples from which comparisons of the effects of the different parameters on the steganalysis strategies can be made. In addition, the results can be compared against the results of the tests run against the images that do not contain embedded information. In this way, the effects of changing single and multiple parameters can be examined for differences from the norm.

4.2.2.2 Message Files

In addition to using the same cover files in each test case, identical message files are used among the test cases. This provides another level of control in the experiment,

and it permits examination of the effects of the characteristics of the message file on the steganalysis strategy.

Table 7, Test Cases for Sorted and Random Palette Partitions

	Steganography Tool			Loading Level		Message File Format		
Case	*HideSeek*	*S-Tools*	*Steganos*	High	Low	Binary	Image	Text
1	●			●		●		
2	●			●			●	
3	●			●				●
4	●				●	●		
5	●				●		●	
6	●				●			●
7		●		●		●		
8		●		●			●	
9		●		●				●
10		●			●	●		
11		●			●		●	
12		●			●			●
13			●	●		●		
14			●	●			●	
15			●	●				●
16			●		●	●		
17			●		●		●	
18			●		●			●

4.2.2.3 Steganography Tools

The three tools used in this research are unique. They have different interfaces, algorithms, and options. However, as mentioned previously, these tools provide a good sampling of the tools in use today. Settings and options used for each tool were identical

across test cases and are listed in Table 8. Details concerning each setting and other available options can be found in the documentation included with each tool.

Table 8, Tool Settings and Options

Tool	Settings and Options
HideSeek for Windows95	Password = "password" Message Encryption = (none available) Compression = (none available)
Steganos for Windows95	Password = "password" Message Encryption = Disabled Compression = Enabled (not configurable) 8-bit to 24-bit conversion = Disabled
S-Tools v4.0	Password = "password" Message Encryption = Enabled (not configurable) Compression = Disabled Encryption Algorithm = IDEA Median-cut Box Color Reduction = Center Dimension Choice = Large RGB Distance Floyd-Steinberg Dithering = Disabled

4.2.3 Point-noise Threshold Test

4.2.3.1 Overview

The threshold test is the more involved test of the two examined. The first step is to convert the raw bitmap image to an intensity map, or grayscale image. The palette must be reordered according to luminance, and the image data is renumbered to match the new palette. The next step is to apply the point-noise filter and create a map of the resulting pixel values. This map is then converted to a binary (*black-and-white*) image where each pixel is converted to a new value based upon a given threshold level. The final step calculates statistics of the binary image that shows the number of pixels significantly different from those around it. The output format from the threshold test is a

single ASCII text file that lists each file in a test case and the number of *hits*; that is, pixels with values above the threshold.

4.2.3.2 Initial Technique

Initially, a combination of C-programs and MATLAB scripts were written to accomplish the threshold test. The C utilities preprocess the bitmap image by extracting the image data from the file and converting it to a grayscale image. The palette is also extracted and converted to a luminance-ordered palette. A series of MATLAB scripts and user-developed functions are used to filter the image and perform the threshold procedure on the grayscale data. The resulting binary image can be displayed in MATLAB. The data file containing the percentage of *hits* in the threshold image is a tab-delimited text file that can be read into either MS-Excel or MATLAB for analysis.

4.2.3.3 Revised Technique

This multiple step process is slow due to the preprocessing requirement and the iterative approach to manipulating the image data in MATLAB. A search of the MATLAB web site revealed that a newer image processing toolkit was available that read different image formats and performed the grayscale conversion using MATLAB functions optimized for matrices. By replacing the previous C code with the newer MATLAB functions, the time involved decreases by nearly 50%. In addition, MATLAB provides a function that takes a user-defined filter and performs the filter operation on a given matrix. Using this function with the point-noise filter further reduced the time to perform the threshold test.

The result of combining the newer functions was a single, more efficient, user-developed MATLAB function. It works on each bitmap file in its directory and performs

the actions shown in Figure 18. The grayscale and binary images are stored to disk for supplementary examination, if needed.

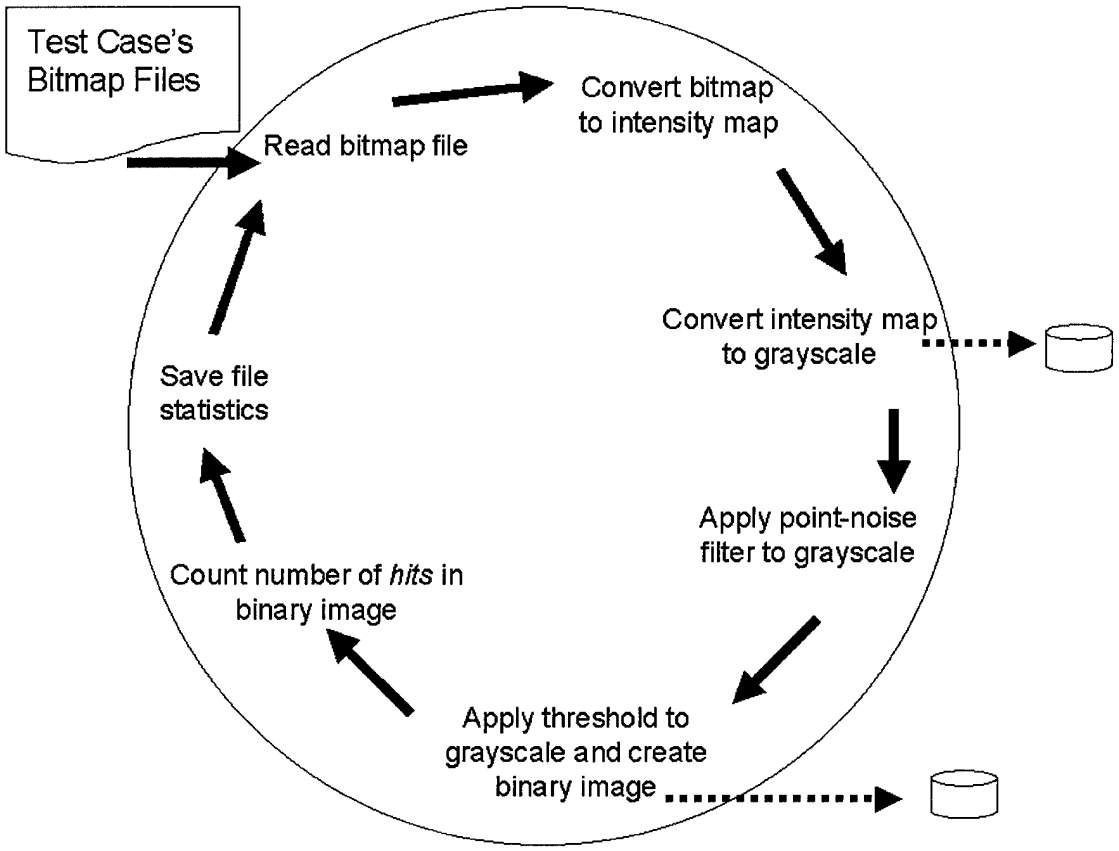


Figure 18, Point-noise Threshold Test Process

4.2.3.4 Bitmap to Intensity Map Conversion

As mentioned in Section 2.3, color is a three-dimensional description of light encompassing radiance, brightness, and luminance. In order to quantify a pixel's *value* relative to those around it, we use its luminance (amount) [6]. Converting the three-dimensional value to an associated intensity level is a matter of averaging the three

components that comprise the pixel's color. The intensity, I, of a pixel is given by the equation:

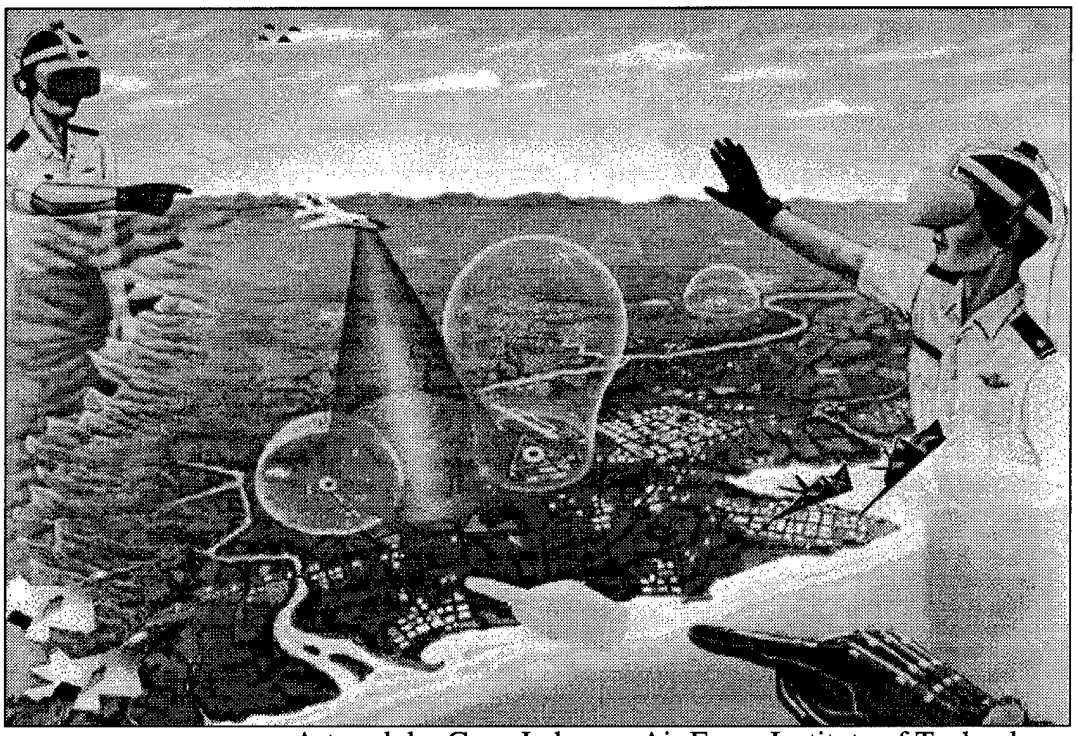
$$I = \frac{1}{3}(R + G + B)$$

In order to convert a bitmap file into a grayscale image, the palette is first converted to a grayscale palette. Each RGB component is replaced with the intensity of the color. (Recall from Section 2.3 that a grayscale palette has equivalent color values for each of the RGB bytes.) The palette is then ordered according to intensity level.

The final step involves renumbering the original image data to match the new palette indices. This step requires a vector to be constructed that matches the old color index to a corresponding new grayscale index. Once the image is renumbered, the vector can be discarded. The grayscale image is then saved with its new grayscale palette. Figure 19 shows a resulting grayscale image, or intensity map, of an originally color bitmap.

4.2.3.5 Point Detection In Images

The purpose of point detection is to identify pixels that are significantly different than those around it. Gonzalez and Woods [6] describe a *neighborhood* as those pixels around a central pixel, p, denoted by (x,y). Their coordinates are given by Figure 20. Point detection then becomes a two-step process involving spatial filtering and thresholding.



Artwork by Gene Lehman, Air Force Institute of Technology

Figure 19, Original Intensity Map (Grayscale Image)

(x-1, y+1)	(x, y+1)	(x+1, y+1)
(x-1, y)	(x, y)	(x+1, y)
(x-1, y-1)	(x, y-1)	(x+1, y-1)

Figure 20, Pixel Neighborhood

4.2.3.5.1 Spatial Filtering

In image processing, spatial filters are sometimes referred to as spatial masks. The idea behind using a mask in this research is to calculate a response to a given pixel such that the result is small if the pixel is similar in value to its neighbors, and large if the pixel is significantly different. A common implementation is a 3×3 mask with negative coefficients on the outside edge and a positive value in the center.

The response of a mask, R, is calculated by the equation

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

,where

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

and z_i is the intensity of the pixel associated with the mask coefficient w_i.

The filter used in this experiment is shown in Figure 21. The sum of the coefficients is zero so that if the mask is applied to an area of constant or slightly varying intensity, its response is zero or very small. The response of the mask is defined with respect to the center pixel. If the mask is centered on a pixel on the edge of the image, its response is calculated using only that portion of the mask that resides over pixels in the image. [6] See Appendix E, or consult Gonzalez and Woods [6], for more information on filters.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 21, 3x3 Point-detection Spatial Filter

4.2.3.5.2 Thresholding

Thresholding is used in this experiment to isolate those pixels that are significantly different from its neighbors. It measures the weighted-differences between the center pixel and its neighbors and is a straightforward process. A pixel is said to be *detected* if $|R| > T$, where T is a nonnegative threshold and R is the response of the mask. The result of thresholding is a binary image in which those pixels that are below the threshold are black, and those above are white.

Selecting an appropriate threshold for this experiment was accomplished through empirical observation. The idea is to determine a threshold where an image that does not have embedded information will result in very few *hits*. The results of a series of tests on the image in Figure 19 are shown in Table 9. The corresponding binary images are shown in Figure 22. (The images are shown in negative to enhance visibility during printing; the original images indicate *hits* as white.) Notice that the number of *hits* decreases as the threshold increases. An *increase* in the threshold is conceptually a *decrease* in the sensitivity of the spatial filter.

Table 9, Threshold Data on Original Intensity Map

Image	Total Pixels	Hits	Percent
Threshold = 250	194,400	8,935	4.5961 %
Threshold = 500	194,400	2,052	1.0555 %
Threshold = 750	194,400	158	0.0813 %
Threshold = 1000	194,400	8	0.0041 %

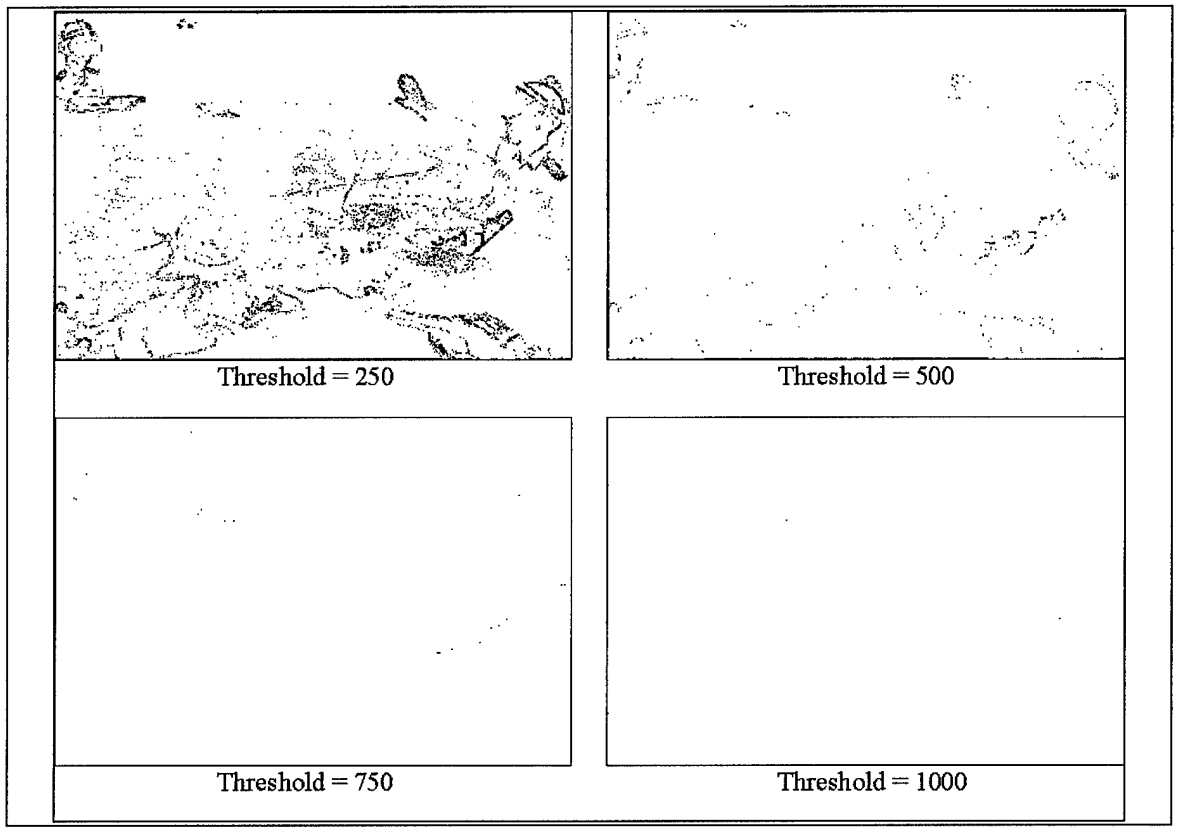


Figure 22, Results of Threshold on Original Intensity Map

4.2.3.6 Output of Point-noise Threshold Test

The output of this test is a (*threshold*, *percentage*) pairing that can be used in a binary test for embedded information. This is accomplished by comparing a total number of hits for a given threshold. The sensitivity of the test (threshold) and the minimum percentage of hits can be varied in order to achieve a binary test for the presence of embedded information. Table 10 displays the results of a sample trial showing the corresponding hits for the same grayscale image after information is embedded. In this case, differences exist between the two cases of palette ordering. It shows that luminance ordered palettes might be somewhat more resistant to the point-noise threshold test at the

same threshold level. The visible results of the sample test on the embedded images are shown in Figure 24 and Figure 25.

Table 10, Threshold Data on Embedded Intensity Maps (Threshold = 1000)

Image	Total Pixels	Hits	Percent
Luminance-ordered palette	194,400	7	0.0036 %
Frequency-ordered palette	194,400	1,082	0.5566 %

Plotting the percentage of hits as a simple bar chart can visually show the results of the threshold test. For example, the results of the test shown in Table 10 are compared against a similar threshold test done on the original image and displayed in Figure 23. This method is used to ascertain possible (*Threshold, Hit Percentage*) pairings for use in a binary test for embedded information.

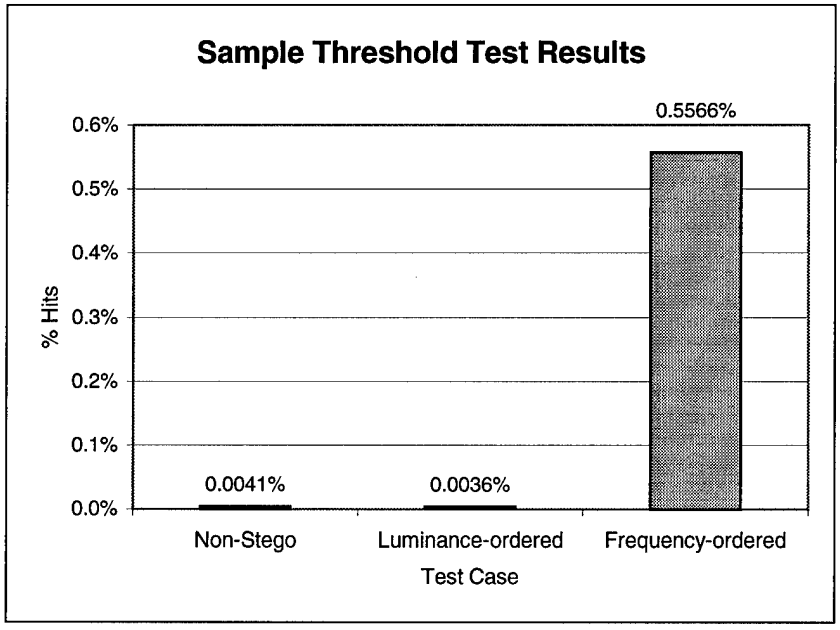
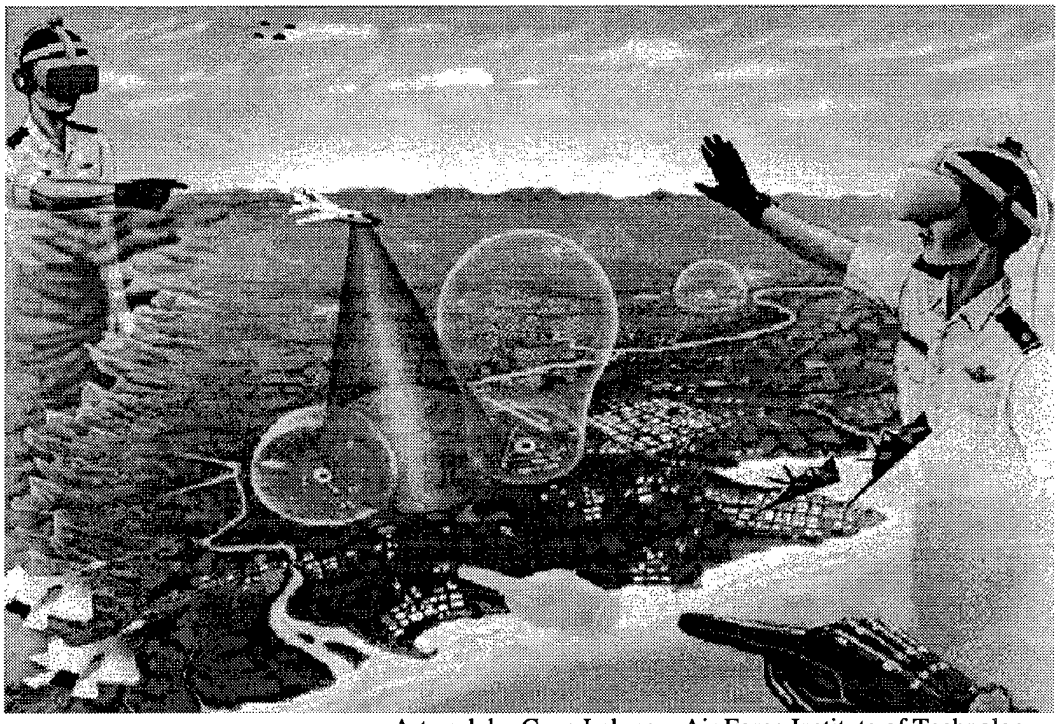


Figure 23, Sample Threshold Test Results



Artwork by Gene Lehman, Air Force Institute of Technology

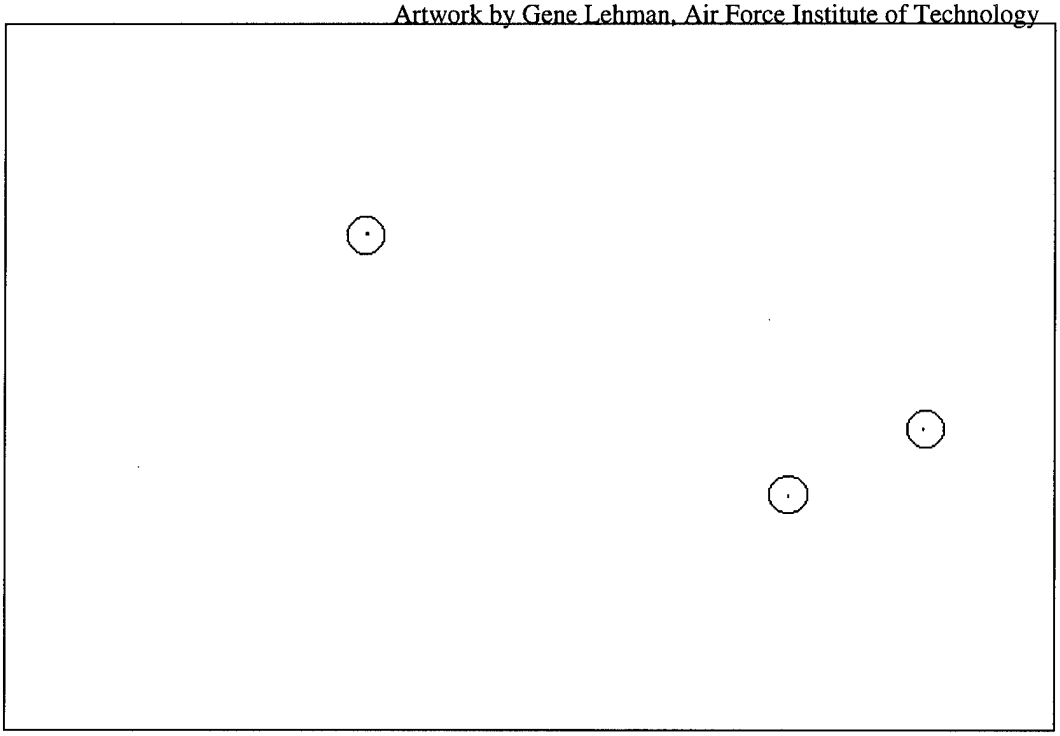
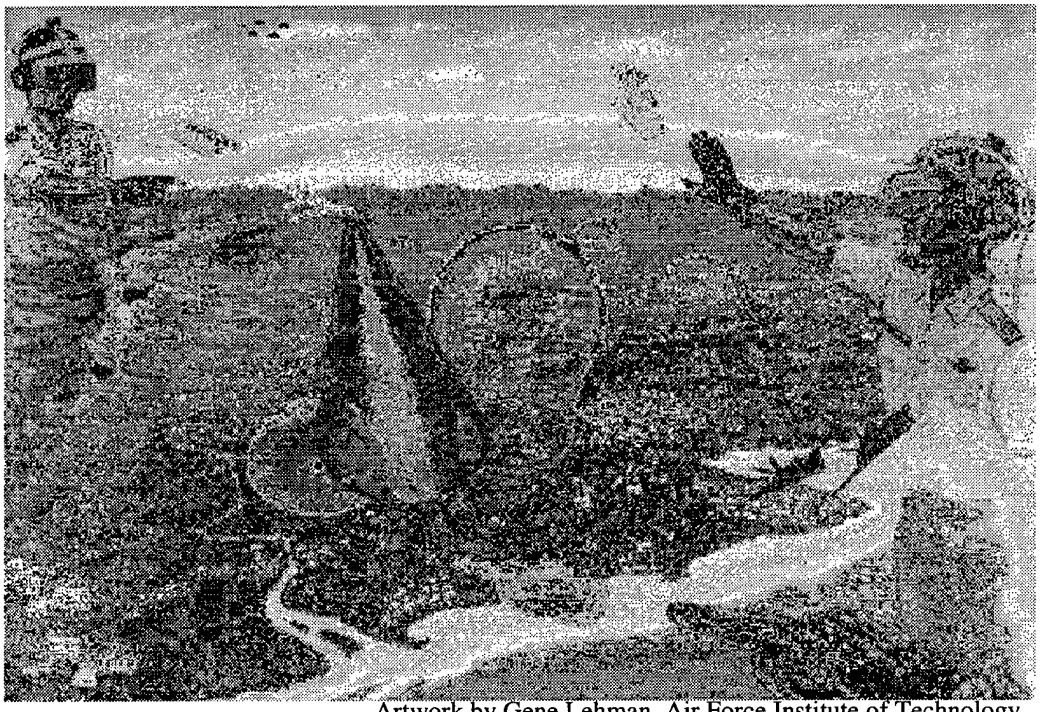


Figure 24, Stego Intensity Map and Threshold Image, T = 1000 (luminance-ordered)



Artwork by Gene Lehman, Air Force Institute of Technology

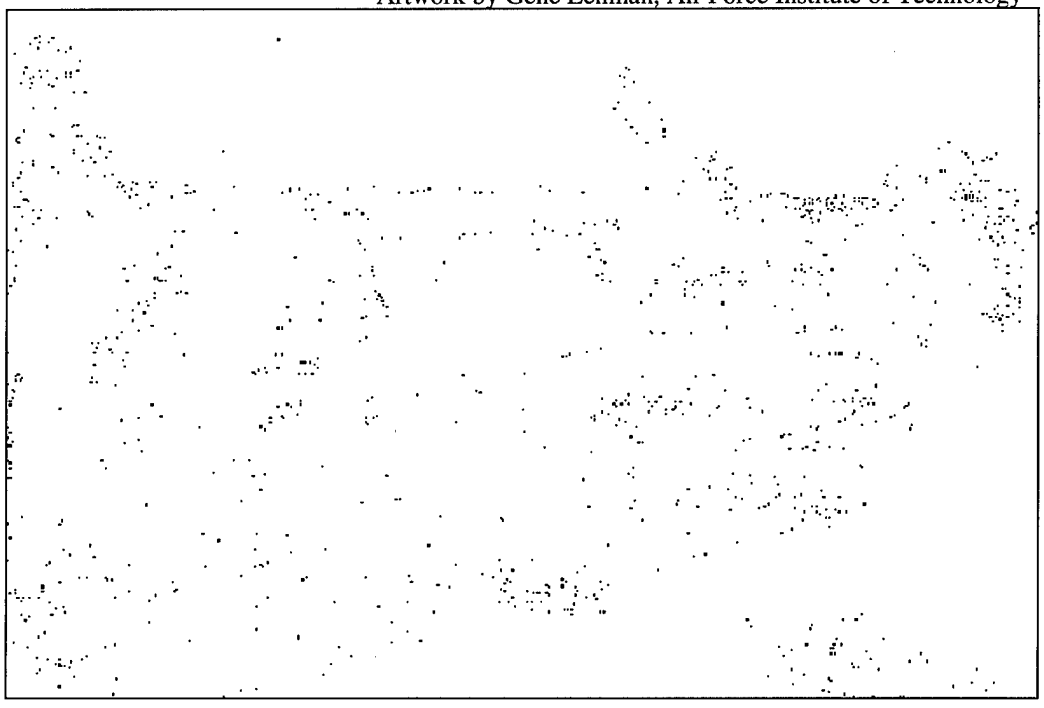


Figure 25, Stego Intensity Map and Threshold Image, T = 1000 (frequency-ordered)

4.2.4 Byte Frequency Analysis

4.2.4.1 Overview

The byte frequency test entails tallying each of the 256 possible byte values found in an image file. The output of this test is a single text file for each case that contains byte counts for each file. The output format is an ASCII file structured as a 256×30, comma-delimited matrix. From this output, the number of bytes of each value can be compared to the total number of bytes in the file. The data can be plotted and analyzed as a frequency chart or histogram.

4.2.4.2 Technique

The byte-frequency analysis is designed to examine what changes occur to a file when information is embedded in it. The basis for this analysis is the frequency and distribution of byte values in a file. Collecting the byte frequency data is accomplished with a user-developed utility written in C. The utility reads a list of files and scans each one, in turn. The user can select the output format through a command-line parameter. There are two options:

- comma-delimited (MS-Excel format)
- space-delimited (MATLAB format)

Before data is collected on embedded cover files, the frequency plots of non-stego files are examined to establish a baseline metric against which further data can be compared and differences detected.

4.2.4.3 Output of Byte Frequency Test

The output of this test is the percentage of the total bytes in a file that are above or below a certain byte value. As mentioned above, the output of the byte frequency test can be examined as a frequency plot. For example, Figure 26 shows the corresponding plot of the image in Figure 19. This particular version of the image has a frequency-ordered palette and its byte distribution is a result of the ordering of the palette. The majority (99.4%) of the file's byte values is in the image data area of the file, and the byte values that occur most frequently are indices to the most common colors in the image. Therefore, the colors appearing most often in the image equate to lower indices and, thus, lower byte values.

The byte frequency plot shown in Figure 26 is a discrete frequency plot with 256 bins where each bin represents the total number of bytes for a single byte value. Such a distribution is difficult to succinctly characterize, and its application to an automated tool may become cumbersome. However, reducing the number of bins provides an adequate partitioning of the results that still lends itself to a binary test for the presence of embedded information.

Three plots are shown in Figure 27, Figure 28, and Figure 29 below. They represent arbitrarily chosen bin counts of 16, 8, and 4 bins. By decreasing the number of bins, the relative differences in the distribution become more pronounced and are more readily discriminated. The reduced-bin, bar graph format is used in subsequent test cases to determine possible byte value ranges and distributions that might indicate the presence of embedded information.

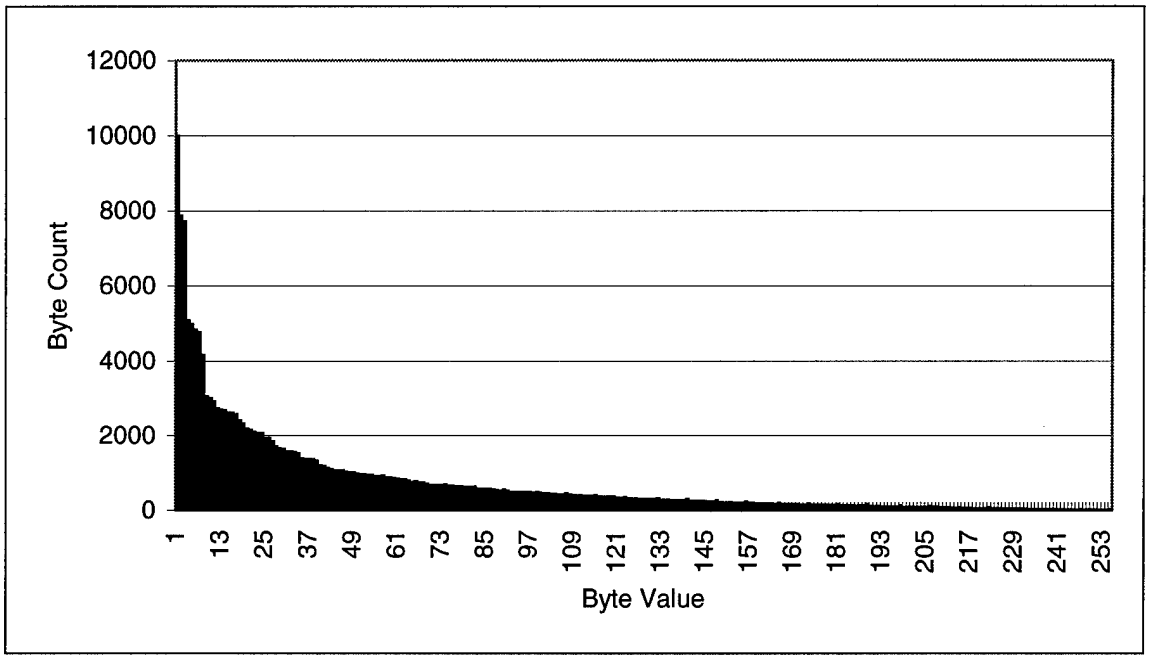


Figure 26, Original Image Byte Frequency Plot (frequency-ordered palette)

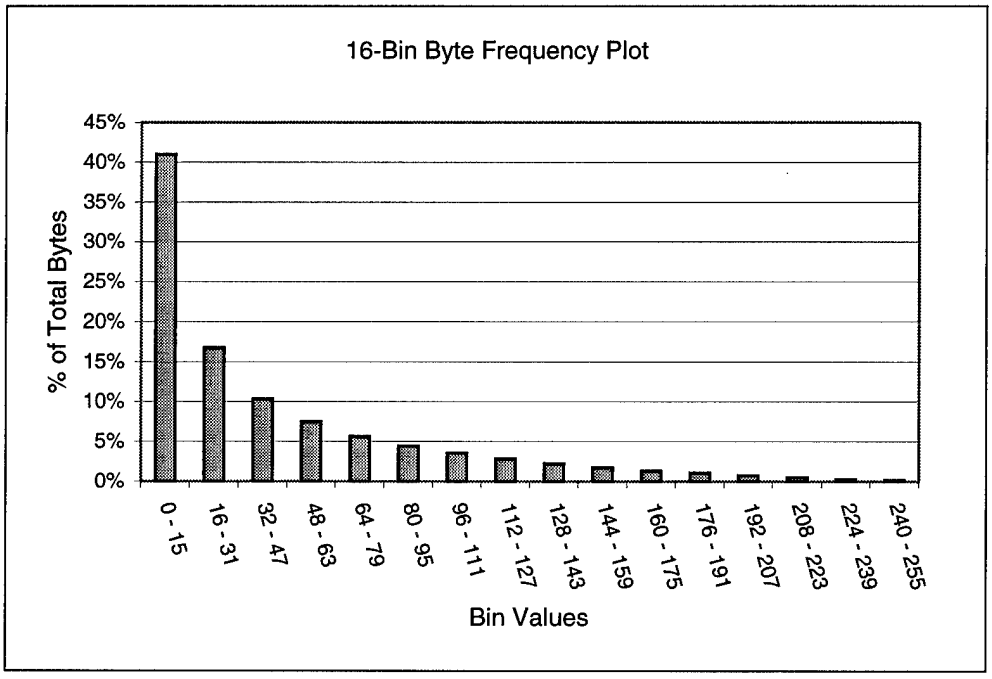


Figure 27, 16-Bin Byte Frequency Plot

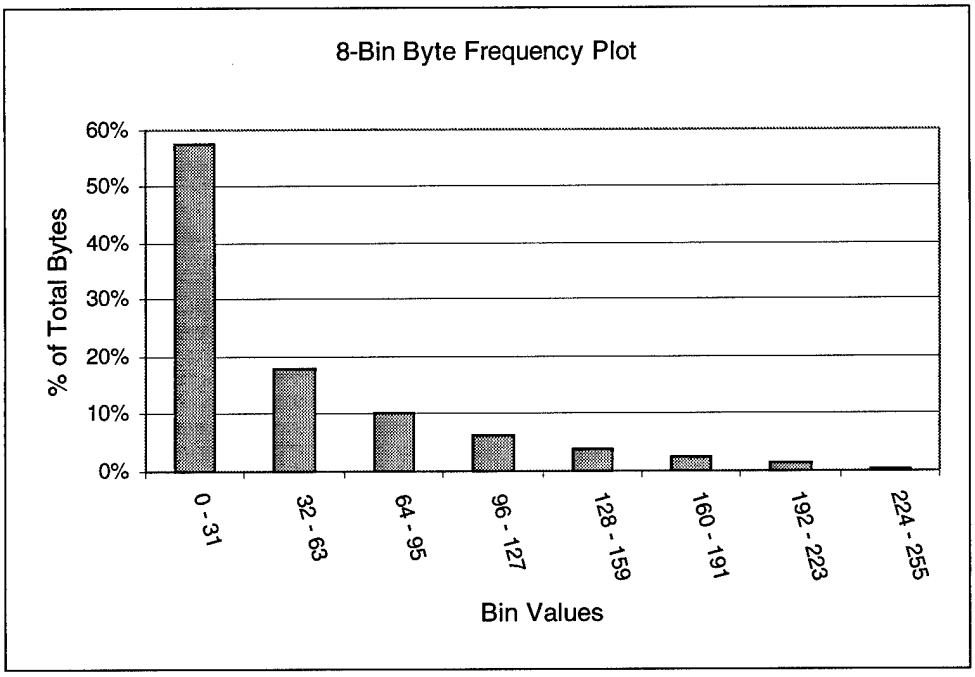


Figure 28, 8-Bin Byte Frequency Plot

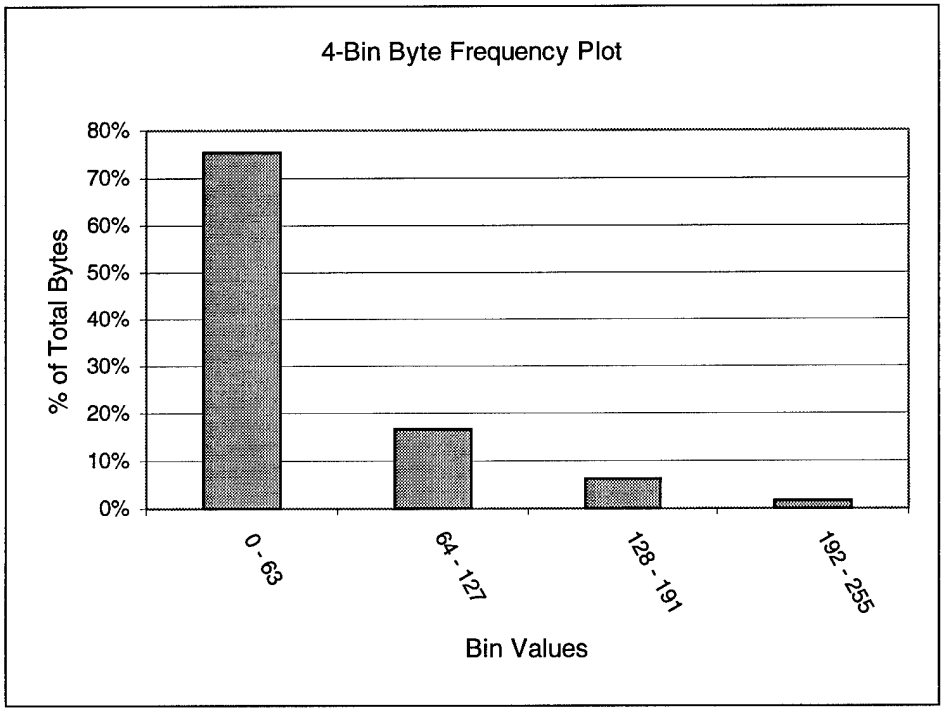


Figure 29, 4-Bin Byte Frequency Plot

It is important to note that subtle differences in the frequency distribution are lost as the number of bins is reduced. For example, Figure 30 shows the byte frequency plot of the same image used in Figure 26 after hiding information using *HideSeek for Windows 95*. Although the general shape of the original distribution is maintained, there are slight variances caused when byte values are changed to indices immediately above and below the original palette indices. These differences are essentially lost when the number of bins is reduced. A quick comparison of Figure 31 with Figure 29 demonstrates this problem.

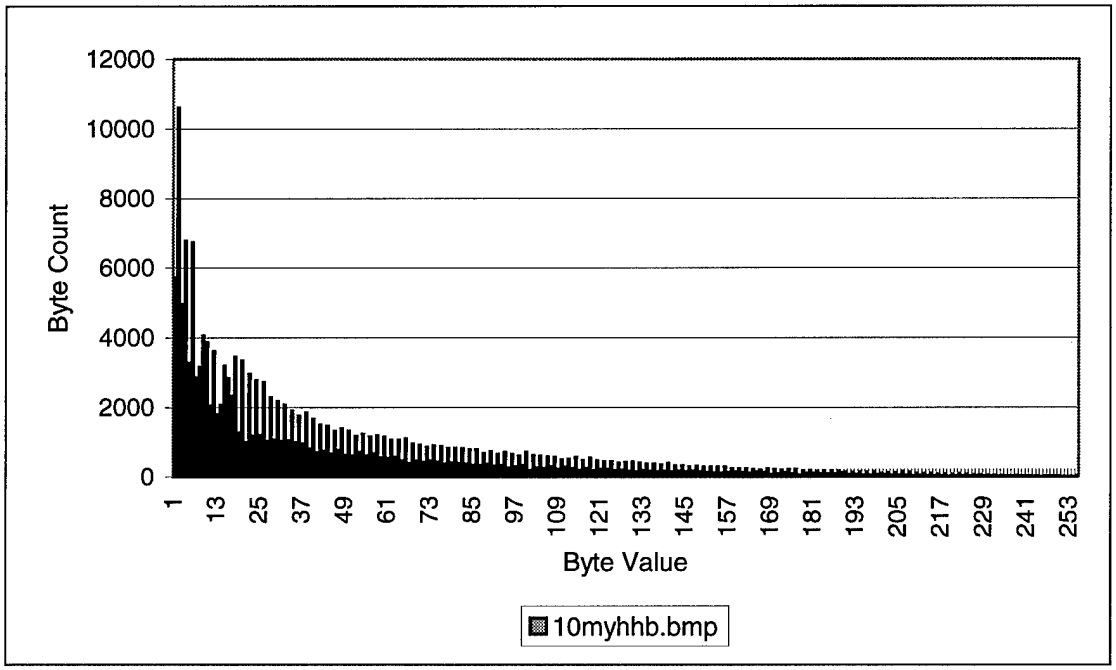


Figure 30, HideSeek Image Byte Frequency Plot (frequency-ordered palette)

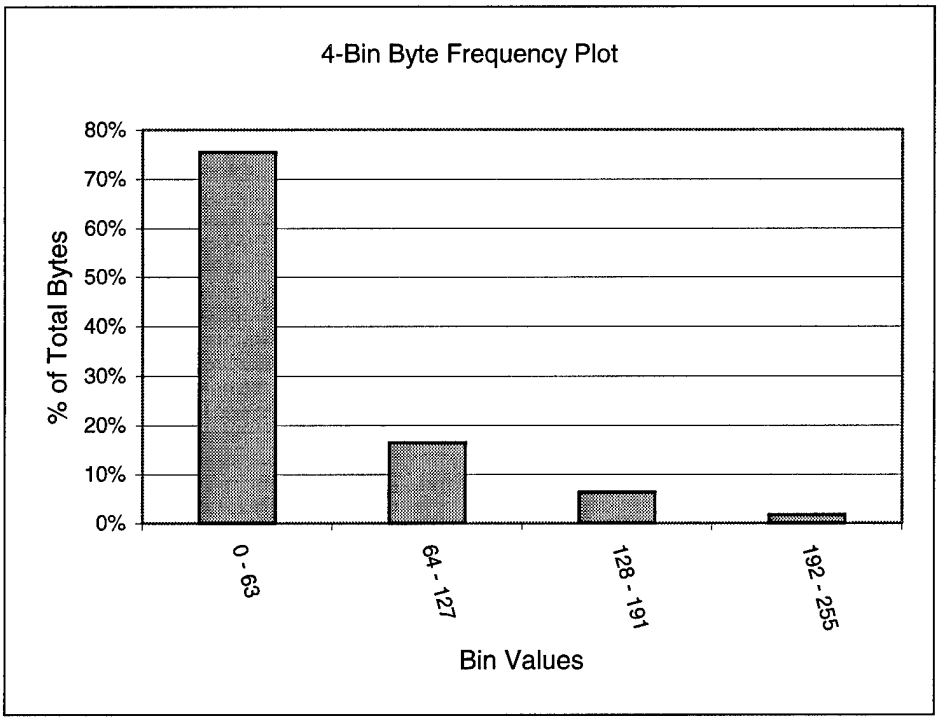


Figure 31, HideSeek Image 4-Bin Byte Frequency Plot (frequency-ordered palette)

In other cases, the differences in the frequency distributions between the test cases and the non-stego files are pronounced enough to make this test successful. Figure 32 shows the byte frequency plot of the same image after using *S-Tools v4.0.* The four-bin plot of this data, shown in Figure 33, reveals enough contrast to make this test worthy of further examination.

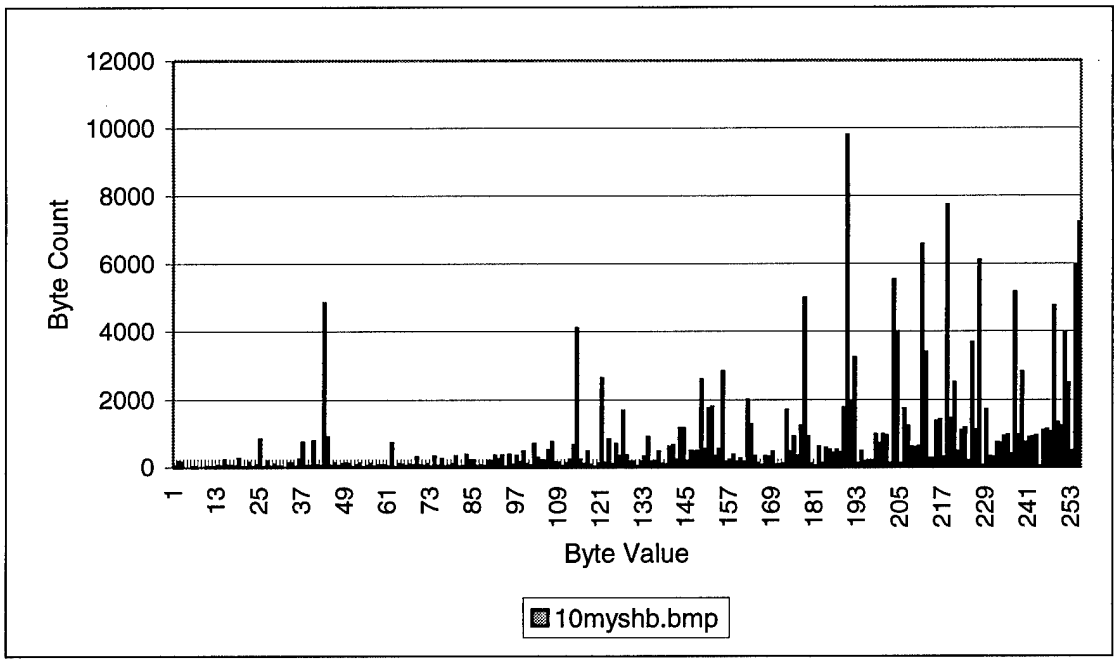


Figure 32, S-Tools Image Byte Frequency Plot (frequency-ordered palette)

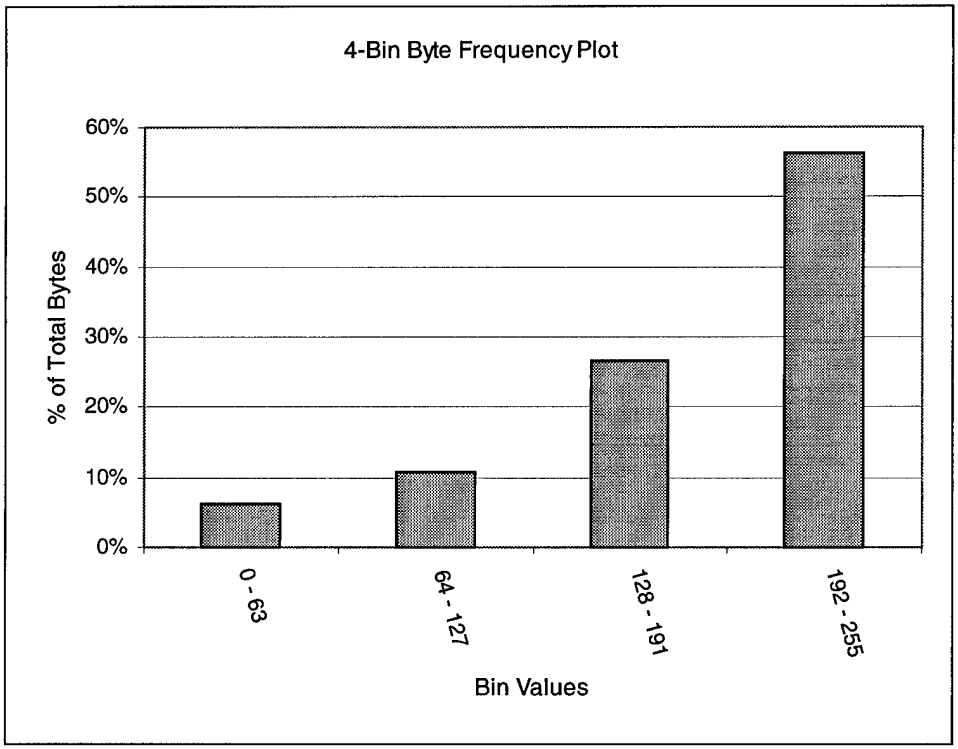


Figure 33, S-Tools Image 4-Bin Byte Frequency Plot (frequency-ordered palette)

V Results

5.1 Introduction

This chapter is divided into two main areas. Section 5.2 discusses the results of the point-noise threshold test, and Section 5.3 examines the outcome of the byte frequency analysis. Finally, ancillary results showing the effects of cover file loading levels and message file composition are discussed in Section 5.4.

5.2 Point-noise Threshold Test

The first test conducted on the library of test cases was the point-noise threshold test. This test operates on the image data in its logical spatial format. The results of the point-noise threshold test are presented below according to the two major partitions in the experiment: frequency-ordered and luminance-ordered palettes.

5.2.1 Frequency-ordered Partition

5.2.1.1 Overall Results

The initial results of the threshold test indicated differences in the median percentage of *hits* between the stego and non-stego images in each of the four thresholds, as shown in Table 11. The variability of the image data within the library of images was considerable, so the median statistic was used because it is less sensitive to outliers than the mean.

Table 11, Overall Median Percentage of Hits (frequency-ordered)

	250	500	750	1000
Non-stego	7.386%	1.009%	0.185%	0.007%
Stego	29.039%	9.698%	3.038%	0.962%

When the results in the stego cases of Table 11 are broken down further, as shown in Table 12, there is little difference in the median percentage of hits between the non-stego images and the images embedded by *S-Tools*. Therefore, the remaining tools – *HideSeek* and *Steganos* – accounted for the most significant differences. The chart in Figure 34 shows the relative differences in the median percentage of hits broken down by steganography tool. In each of the target thresholds, *S-Tools* produced results similar to the non-stego case, and both *HideSeek* and *Steganos* were significantly different.

Table 12, Tool-wise Median Percentage of Hits (frequency-ordered)

	250	500	750	1000
Non-stego	7.386%	1.009%	0.185%	0.007%
S-Tools	8.042%	1.156%	0.172%	0.022%
HideSeek	33.171%	12.304%	5.607%	2.272%
Steganos	47.400%	21.783%	8.451%	2.756%

5.2.1.2 Threshold Selection and Results

Since each tested threshold reveals significant differences between the {*Non-stego*, *S-Tools*} and {*HideSeek*, *Steganos*} test cases, the next step is to determine which *(Threshold, Hit Percentage)* setting would be best suited for an automated detection technique. Recall from Table 9 and Figure 22 that a threshold of 1000 on a non-stego image produced a signature with very little noise, in the form of *hits*, relative to the other thresholds. Proceeding under the assumption that a non-stego image must have a hit percentage of 0%, the appropriate acceptable hit percentage can be selected for each target threshold.

By plotting the median percentage of hits along with the maximum and minimum values in each data set, a reasonable estimate of the optimal threshold can be determined. The minimum hit percentages shown in Table 13 were gathered from the test results displayed in Figure 35 through Figure 38. The percentages are approximately equal to the maximum values of the non-stego test case. The assumption is that non-stego files should exhibit a negative response to the threshold filter since they do not contain embedded information. See Appendix B.4 for the overall maximum and minimum values for each threshold.

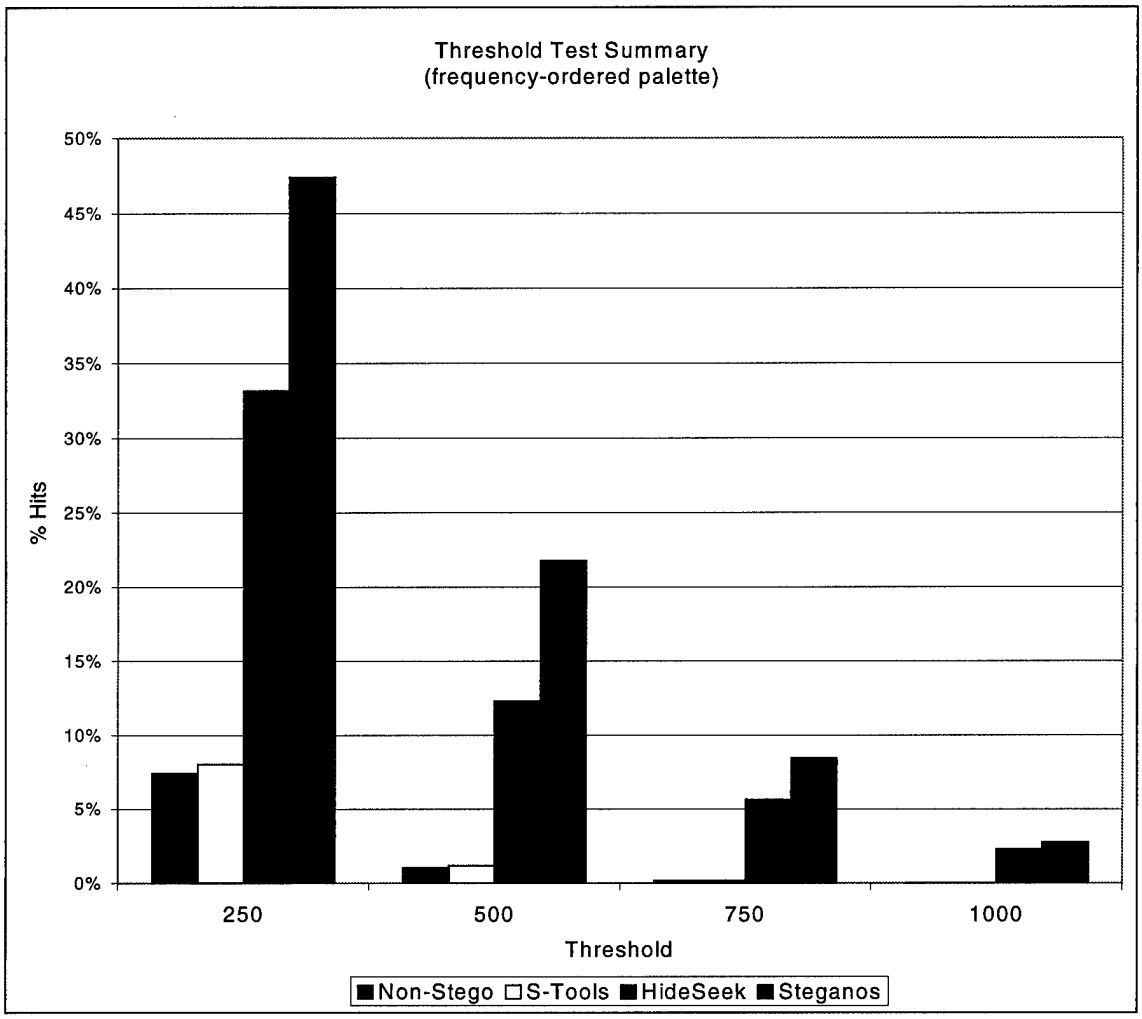


Figure 34, Threshold Test Summary (frequency-ordered palette)

Table 13, Minimum Hit Percentages by Threshold (frequency-ordered)

	250	500	750	1000
% Hits	> 28.00%	> 6.10%	> 1.00%	> 0.17%

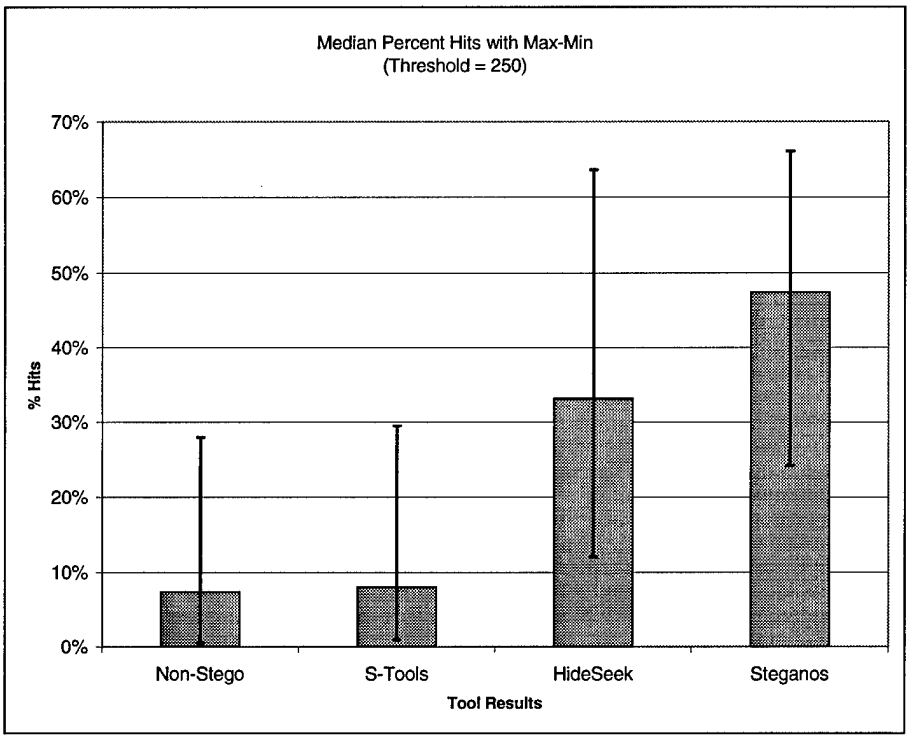


Figure 35, Median % Hits with Max-Min (T=250)

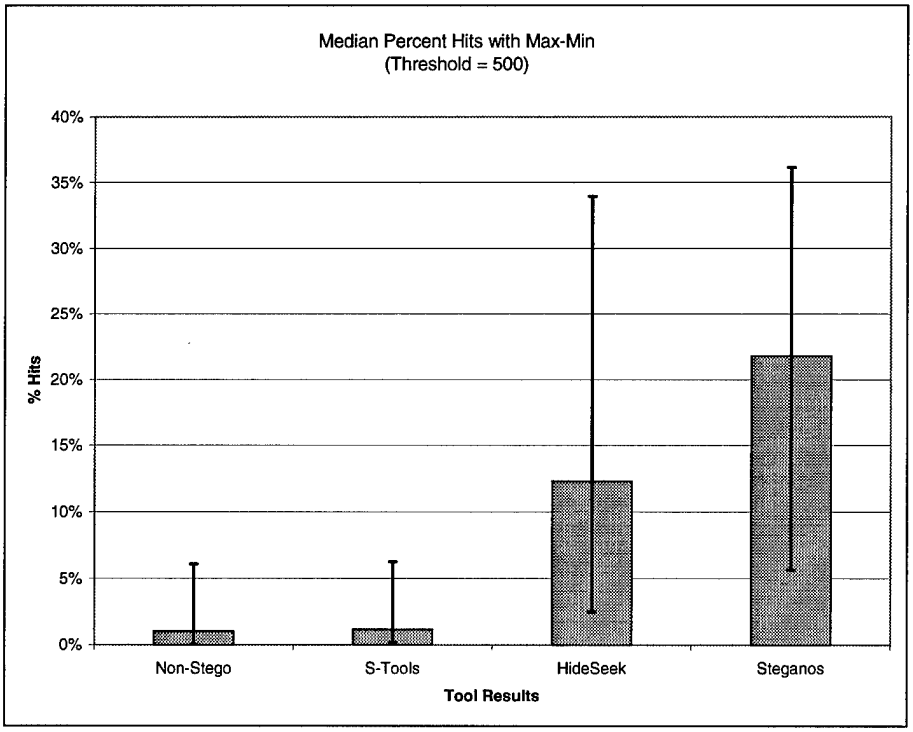


Figure 36, Median % Hits with Max-Min (T=500)

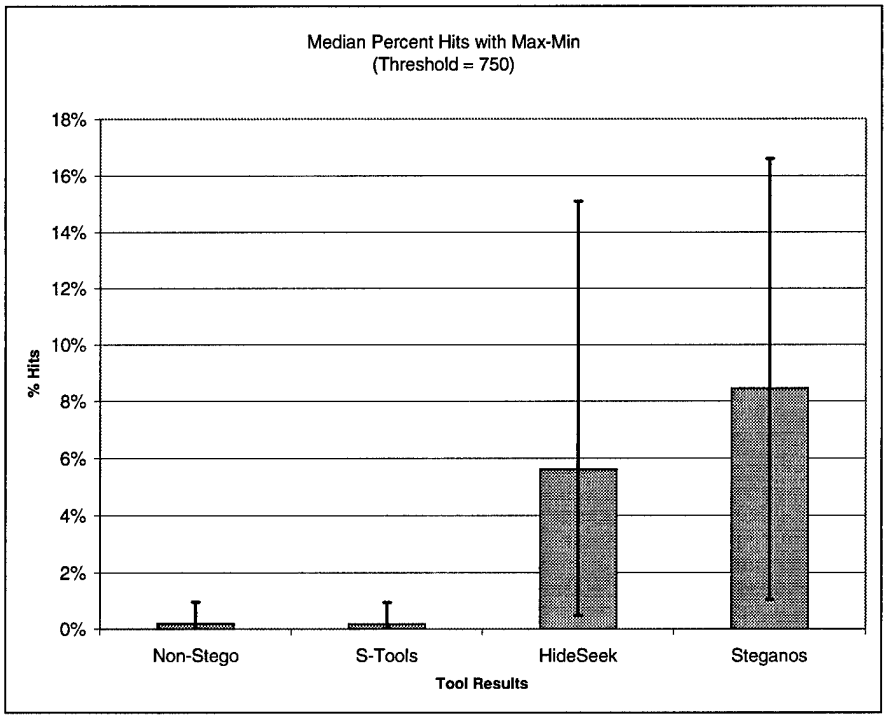


Figure 37, Median % Hits with Max-Min (T=750)

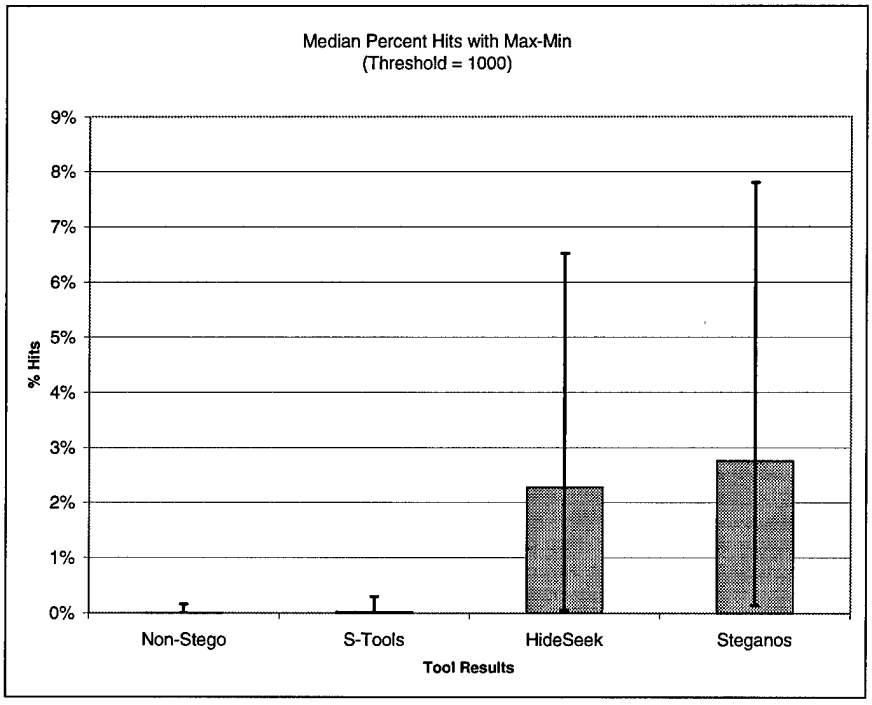


Figure 38, Median % Hits with Max-Min (T=1000)

Operating under this assumption, the success rate of the test on the frequency-ordered palette test cases can be calculated. A file is marked as *detected* if the hit percentage exceeds the minimum hit percentage from Table 13 for a given threshold. Table 14 shows the success rate for each test case. The success rate for the non-stego case is always 100% with no false positives.

From Table 14 it can be seen that *S-Tools* beats the threshold test in almost all test cases. If the *S-Tools* results are removed from the analysis, the combined success rate for the *HideSeek/Steganos* pairing jumps considerably, as shown in Table 15. The fact that the threshold test is easily defeated by *S-Tools* makes sense since the threshold test detects visible distortion, which is something *S-Tools* is careful to minimize.

Table 14, Threshold Test Success Rate (frequency-ordered)

Threshold	Test Case	Total *Detected*	Success Rate (*Detected* /# cases)	Failure Rate
250	**HideSeek**	112	62.22%	37.78%
> 28%	**Steganos**	168	93.33%	6.67%
	S-Tools	6	3.33%	96.67%
	Combined	286	**52.96%**	47.04%
500	**HideSeek**	159	88.33%	11.67%
> 6.1%	**Steganos**	174	96.67%	3.33%
	S-Tools	6	3.33%	96.67%
	Combined	339	**62.78%**	37.22%
750	**HideSeek**	174	96.67%	3.33%
> 1%	**Steganos**	180	100.00%	0.00%
	S-Tools	0	0.00%	100.00%
	Combined	354	**65.56%**	34.44%
1000	**HideSeek**	174	96.67%	3.33%
> 0.17%	**Steganos**	174	96.67%	3.33%
	S-Tools	10	5.56%	94.44%
	Combined	358	**66.30%**	33.70%

Table 15, HideSeek/Steganos Threshold Test Success Rate (frequency-ordered)

Threshold	Test Case	Total *Detected*	Success Rate (*Detected* /# cases)
250	**HideSeek/Steganos**	280	**77.78%**
500	**HideSeek/Steganos**	333	**92.50%**
750	**HideSeek/Steganos**	354	**98.33%**
1000	**HideSeek/Steganos**	348	**96.67%**

Maintaining a minimum hit percentage such that 100 percent of the non-stego images provide a negative response to the threshold test may not be the most realistic application of this strategy. The failure rates in Table 14 represent Type I errors; that is, not detecting a file that has embedded information. A Type II error – detecting a file that does not contain embedded information – is not considered in Table 14 since no non-stego images respond positively to the test. However, if a certain amount of Type II errors are allowed, the number of Type I errors will decrease. If we suppose law enforcement agency can tolerate *coin-flip* probabilities of detecting non-stego images, then the success rates of the threshold test improve. Table 16 shows the corresponding success rates and new minimum hit percentages for each threshold. Again, 50% of the non-stego images were detected. Table 17 shows the results of the *HideSeek/Steganos* pairing.

For the general case of determining an appropriate *(Threshold, Hit Percentage)* pairing, the results above can be combined to give the minimum hit percentage *curves* shown in Figure 40. The lines that connect the data points do not represent the actual population statistics, but do provide a reasonable estimate of the *(Threshold, Hit Percentage)* pairings between points based on the sample. Again, since the data points

represent lower bounds, any hit percentage above the data point would decrease the amount of Type II errors while increasing the chance of a Type I error.

Table 16, Coin-flip Threshold Test Success Rate (frequency-ordered)

Threshold	Test Case	Total *Detected*	Success Rate (*Detected* /# cases)	Failure Rate
250	**HideSeek**	180	100.00%	0.00%
> 8%	**Steganos**	180	100.00%	0.00%
	S-Tools	90	50.00%	50.00%
	Combined	450	**83.33%**	16.67%
500	**HideSeek**	180	100.00%	0.00%
> 1%	**Steganos**	180	100.00%	0.00%
	S-Tools	103	57.22%	42.78%
	Combined	463	**85.74%**	14.26%
750	**HideSeek**	180	100.00%	0.00%
> .1842%	**Steganos**	180	100.00%	0.00%
	S-Tools	79	43.89%	56.11%
	Combined	439	**81.30%**	18.70%
1000	**HideSeek**	180	100.00%	0.00%
> 0.009%	**Steganos**	180	100.00%	0.00%
	S-Tools	110	61.11%	38.89%
	Combined	470	**87.04%**	12.96%

Table 17, Coin-flip Threshold Test Success Rate (frequency-ordered)

Threshold	Test Case	Total *Detected*	Success Rate (*Detected* /# cases)
250	**HideSeek/Steganos**	360	**100.00%**
500	**HideSeek/Steganos**	360	**100.00%**
750	**HideSeek/Steganos**	360	**100.00%**
1000	**HideSeek/Steganos**	360	**100.00%**

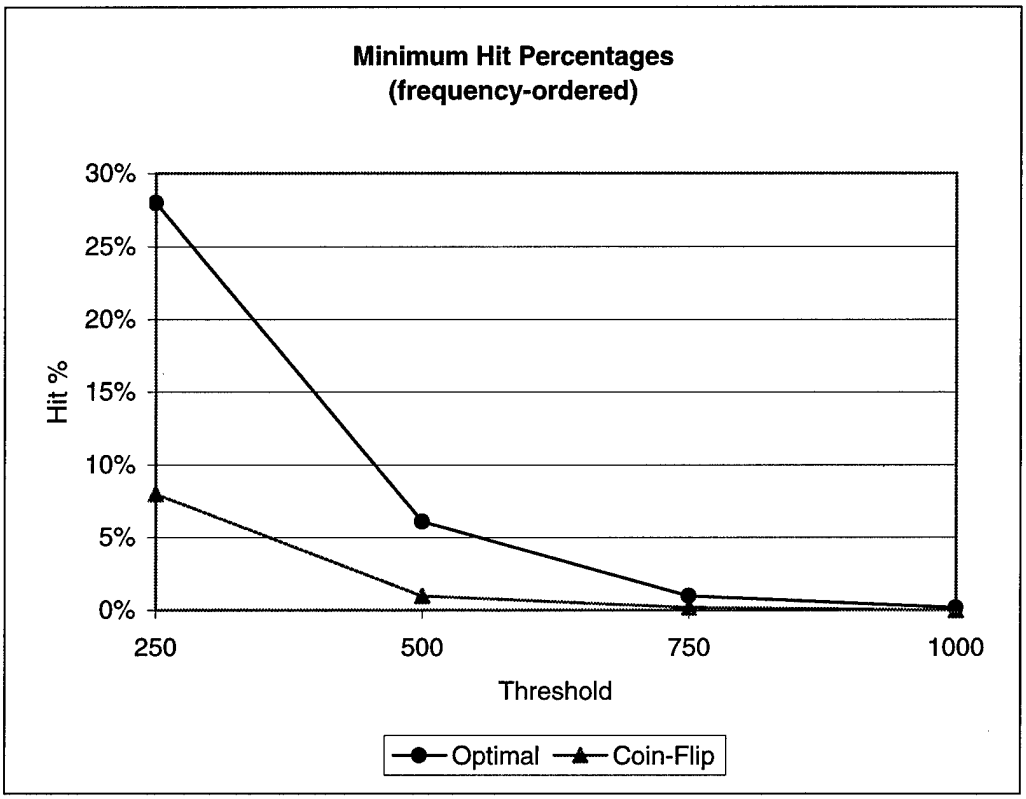


Figure 39, Minimum Hit Percentages (frequency-ordered)

5.2.2 Luminance-ordered Partition

5.2.2.1 Overall Results

Unlike the frequency-ordered test cases, luminance-ordered images were remarkably resistant to the threshold test. The data collected from the luminance-ordered test cases was the same as the previous test partition, and the analysis also proceeded along similar lines. Again, the potentially non-homogenous nature of the image data necessitated the use of the median statistic over the mean because it is less sensitive to outliers. The overall median percentage of hits is shown in Table 18, and Table 19 shows the tool-wise percentages.

Table 18, Overall Median Percentage of Hits (luminance-ordered)

	250	500	750	1000
Non-stego	7.375%	1.055%	0.283%	0.022%
Stego	7.947%	1.087%	0.264%	0.022%

Table 19, Tool-wise Median Percentage of Hits (luminance-ordered)

	250	500	750	1000
Non-stego	7.375%	1.055%	0.283%	0.022%
S-Tools	7.947%	1.174%	0.200%	0.024%
HideSeek	7.355%	1.053%	0.283%	0.022%
Steganos	7.359%	1.060%	0.272%	0.022%

It is evident from Table 19 that the differences between the non-stego and stego cases are negligible. If fact, in some test cases the non-stego image produced a slightly higher hit percentage than stego images. In either case, the differences were not significant enough to use as a discriminator. The results of the threshold test on the luminance-ordered palette test partition are shown Figure 40.

5.2.2.2 Threshold Selection and Results

Due to the results shown previously in Table 19, no appropriate *(Threshold, Hit Percentage)* setting can be obtained for luminance-ordered palette images. However, for the purposes of comparison, the same process used in the first test partition was applied. Remember the assumption from Section 5.2.1.2 that a non-stego image must have a hit percentage of 0% in order to select the appropriate acceptable hit percentage for each target threshold. Based on this assumption, the minimum hit percentages for the

luminance-ordered test partition are shown below in Table 20. The success rates based upon these minimum hit percentages are shown in Table 21. The corresponding results of the *HideSeek/Steganos* pairing are not presented here since the differences between all test cases were insignificant. See Appendix B.4 for the overall maximum and minimum values for each threshold.

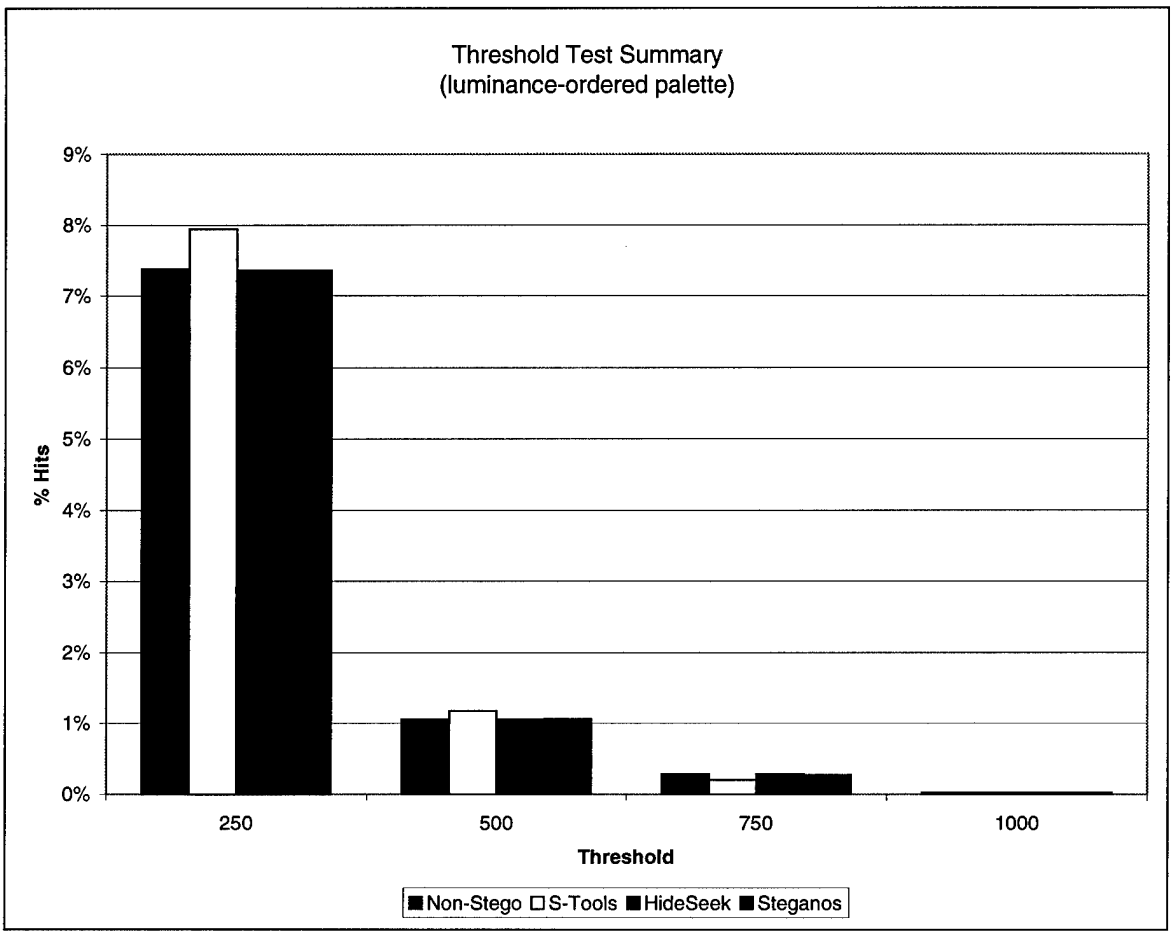


Figure 40, Threshold Test Summary (luminance-ordered palette)

Table 20, Minimum Hit Percentages by Threshold (luminance-ordered)

	250	500	750	1000
% Hits	> 28.00%	> 6.10%	> 1.00%	> 0.34%

Table 21, Threshold Test Success Rate (luminance-ordered)

Threshold	Test Case	Total *Detected*	Success Rate (*Detected* /# cases)	Failure Rate
250 > 28.00%	**HideSeek**	0	0.00%	100.00%
	Steganos	3	1.67%	98.33%
	S-Tools	6	3.33%	96.67%
	Combined	9	**1.67%**	98.33%
500 > 6.10%	**HideSeek**	0	0.00%	100.00%
	Steganos	0	0.00%	100.00%
	S-Tools	6	3.33%	96.67%
	Combined	6	**1.11%**	98.89%
750 > 1.00%	**HideSeek**	0	0.00%	100.00%
	Steganos	0	0.00%	100.00%
	S-Tools	0	0.00%	100.00%
	Combined	0	**0.00%**	100.00%
1000 > 0.34%	**HideSeek**	0	0.00%	100.00%
	Steganos	0	0.00%	100.00%
	S-Tools	1	0.56%	99.44%
	Combined	1	**0.19%**	99.81%

The success rates for the *coin-flip* case, where 50 percent of the non-stego images were detected, is shown in Table 22. Once again, the success rates are not significantly different than the non-stego case, and in many cases they are worse.

Table 22, Coin-flip Threshold Test Success Rate (luminance-ordered)

Threshold	Test Case	Total *Detected*	Success Rate (*Detected* /# cases)	Failure Rate
250	**HideSeek**	90	50.00%	50.00%
> 8.3%	**Steganos**	90	50.00%	50.00%
	S-Tools	90	50.00%	50.00%
	Combined	270	**50.00%**	50.00%
500	**HideSeek**	90	50.00%	50.00%
> 1.05%	**Steganos**	91	50.56%	49.44%
	S-Tools	102	56.67%	43.33%
	Combined	283	**52.41%**	47.59%
750	**HideSeek**	90	50.00%	50.00%
> .28%	**Steganos**	84	46.67%	53.33%
	S-Tools	73	40.56%	59.44%
	Combined	247	**45.74%**	54.26%
1000	**HideSeek**	89	49.44%	50.56%
> 0.0225%	**Steganos**	84	46.67%	53.33%
	S-Tools	92	51.11%	48.89%
	Combined	265	**49.07%**	50.93%

5.3 Byte Frequency Analysis

The second test performed on the library of test cases was the byte frequency analysis test. This test operates on the entire bitmap file rather than only the image data. The results of the byte frequency analysis test are presented below according to the two major partitions in the experiment: frequency-ordered and luminance-ordered palettes.

5.3.1 Frequency-ordered Partition

5.3.1.1 Overall Results

In Figure 26, the frequency distribution had a distinct shape that was concentrated at lower byte values. In fact, this frequency distribution was typical of images with frequency-ordered palettes. Since 99.4 percent of a file's bytes are palette indices in the

image data, the frequency-ordering nature of the palette will produce a greater number of lower index values. The median byte frequency distribution plots for each test case are shown below. The non-stego file results are shown in Figure 41.

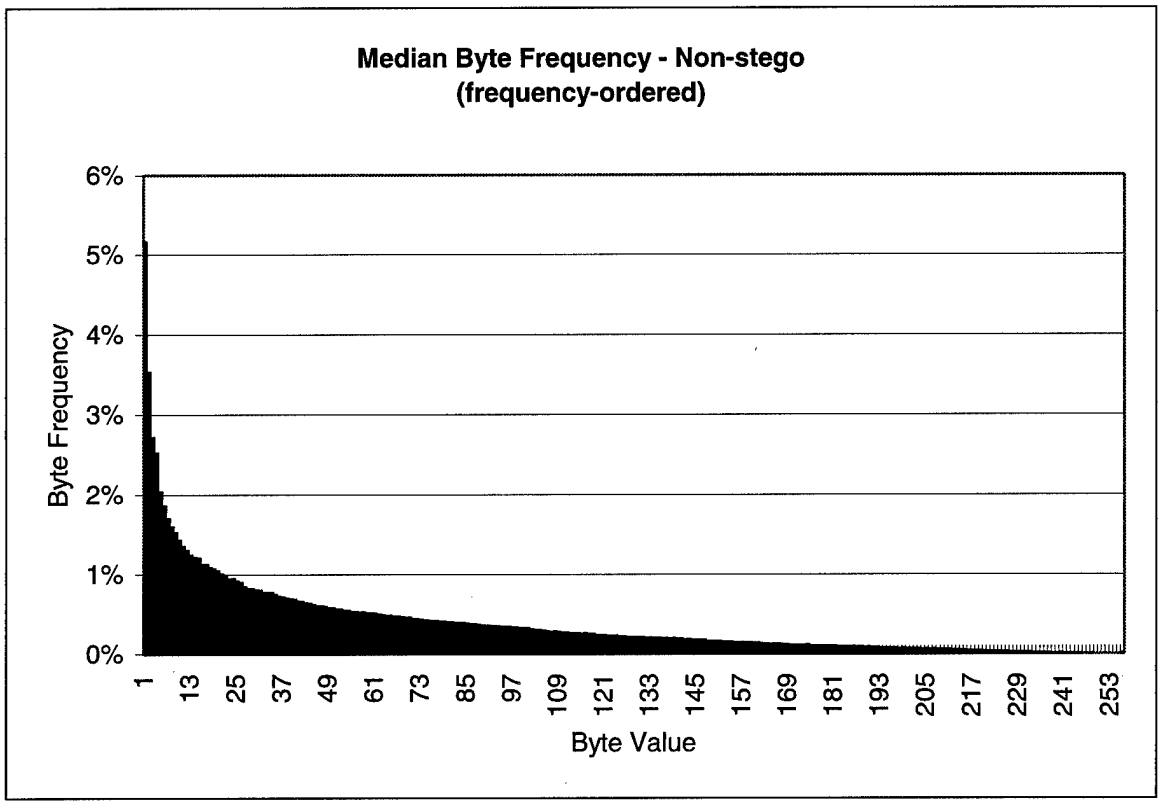


Figure 41, Median Byte Frequency – Non-stego (frequency-ordered)

Compare this plot to the plots shown below representing the *HideSeek* (Figure 42) and *Steganos* (Figure 43) embedded files. The distributions are, not surprisingly, quite similar. The technique employed in both *HideSeek* and *Steganos* is a simple least-bit substitution with no palette manipulation. Consequently, the resulting new indices are either above or below the old index, and by no more than a single value. Therefore, the resulting files have relatively similar byte frequency distributions as the original.

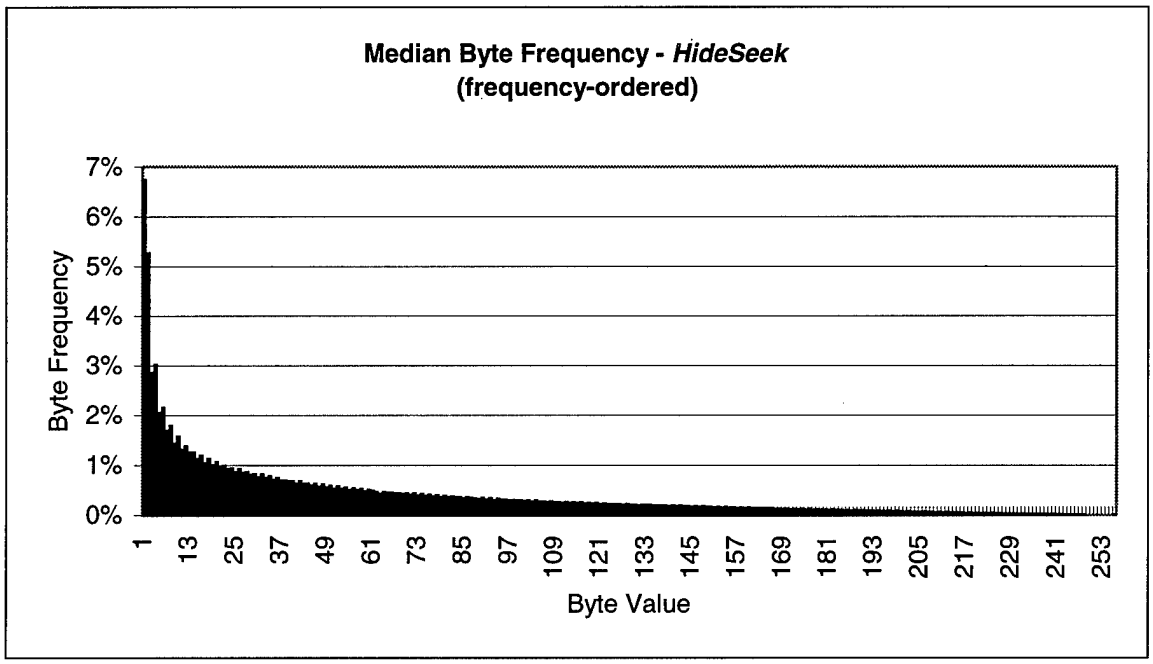


Figure 42, Median Byte Frequency – HideSeek (frequency-ordered)

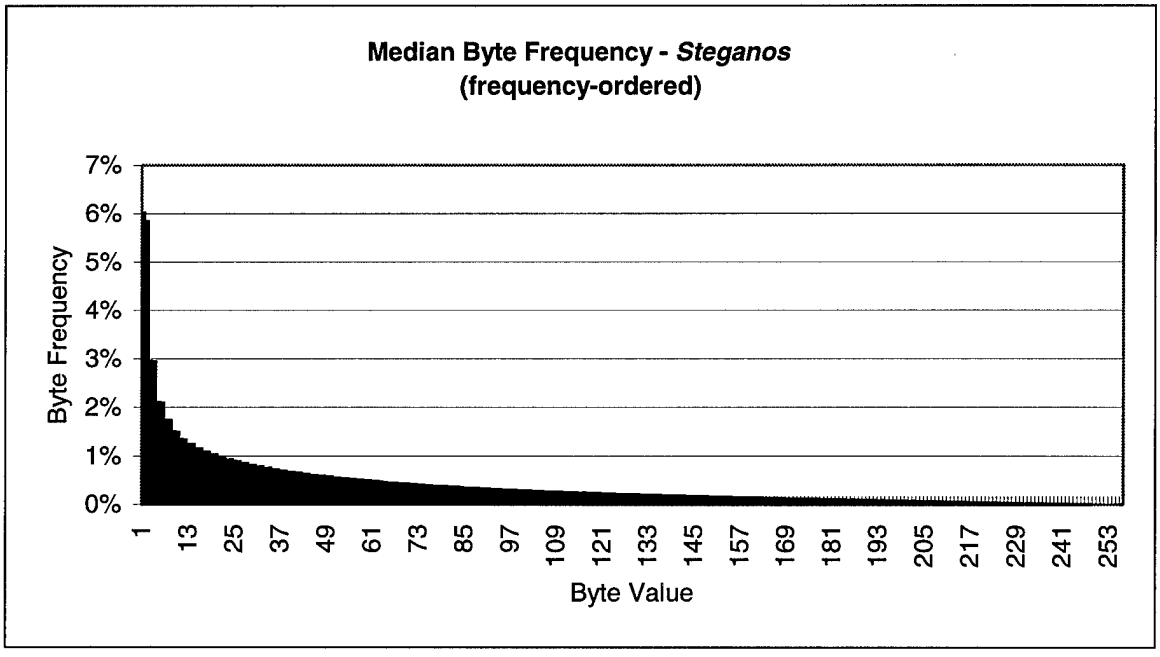


Figure 43, Median Byte Frequency – Steganos (frequency-ordered)

The results were radically different in the case of the *S-Tools* embedded files. The median byte frequency plot for this test case is shown below in Figure 44. The extreme variances at the higher byte values are a result of the unique manner in which *S-Tools* manipulates the palette. (See Section 4.1.3.4, *S-Tools*.) When the image is renumbered to match the new palette, the result is a marked increase in higher palette indices in the image to account for the increased use of the newer colors.

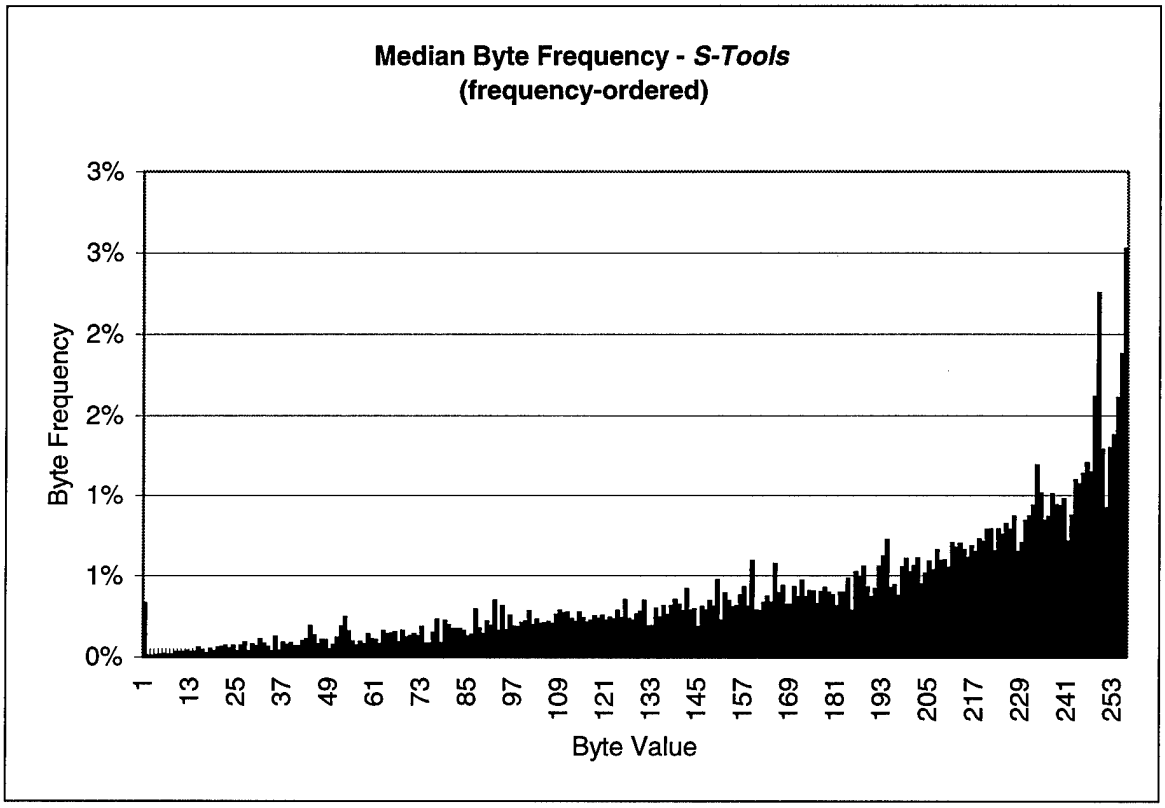


Figure 44, Median Byte Frequency – S-Tools (frequency-ordered)

5.3.1.2 Target Percentage Selection and Results

Using the technique described in Section 4.2.4, the median byte frequency values are combined into an eight-bin frequency distribution plot as shown in Figure 45. From

this figure, it is easy to see that *HideSeek* and *Steganos* do not exhibit any significant differences compared to the non-stego files.

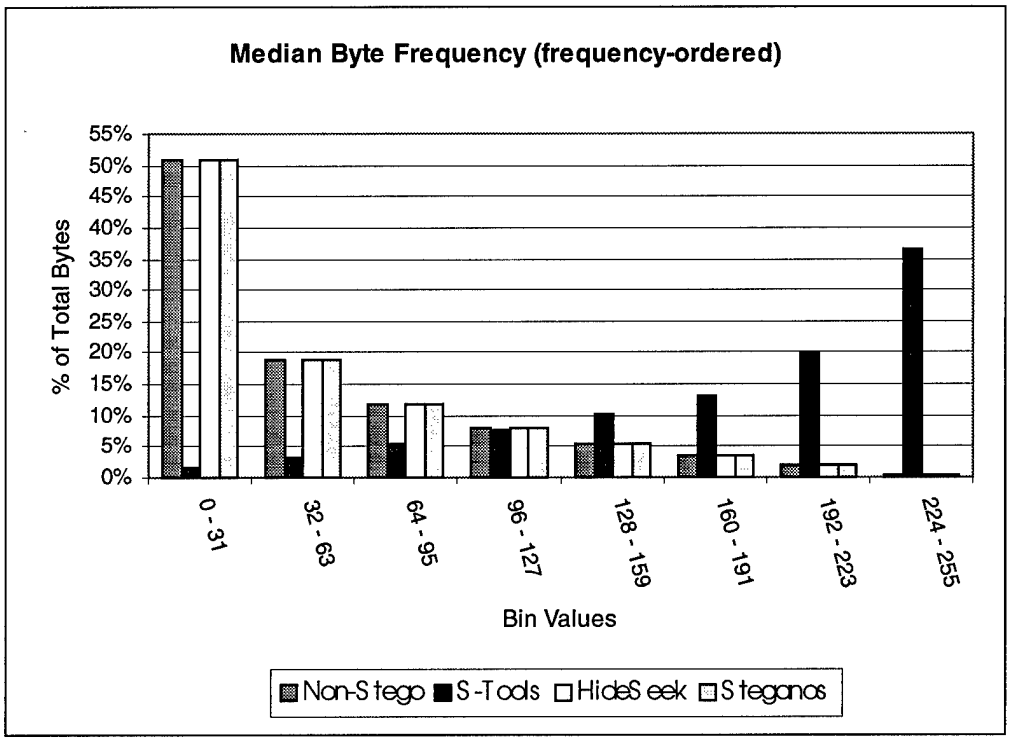


Figure 45, Combined Median Byte Frequency (frequency-ordered)

The next step in determining the utility of the byte frequency analysis test is selecting appropriate parameters to use in detecting the presence of embedded information. Since the results for *HideSeek* and *Steganos* were nearly identical to the non-stego cases, they are ignored. Only the *S-Tools* test case is used in this step since it produced results significantly different from the non-stego files. Consequently, this test is evaluated as a detection tool for *S-Tools* files from this point forward.

The desired metric in this step is a minimum cumulative percentage of bytes that exceed a particular threshold byte value or fall within a specified range. The results of the frequency-ordered palette test cases are shown in Figure 46. In this plot, the median

of the non-stego, *HideSeek*, and *Steganos* files are combined and compared against the *S-Tools*. An appropriate *(Target Percentage, Threshold Value)* pairing can be selected below the line connecting the *S-Tools* data points. For instance, one such pairing might be (60%, 127). In this case, if more than 60% of a file's byte values exceed 127, then the file is selected. Of course, the percentage shown is a median value, so any value less than 60% would also work with the threshold of 127. It is important to note that the results shown in Figure 46 are discrete data points and not a continuous distribution. The line connecting the points is displayed to highlight the trend of the data set.

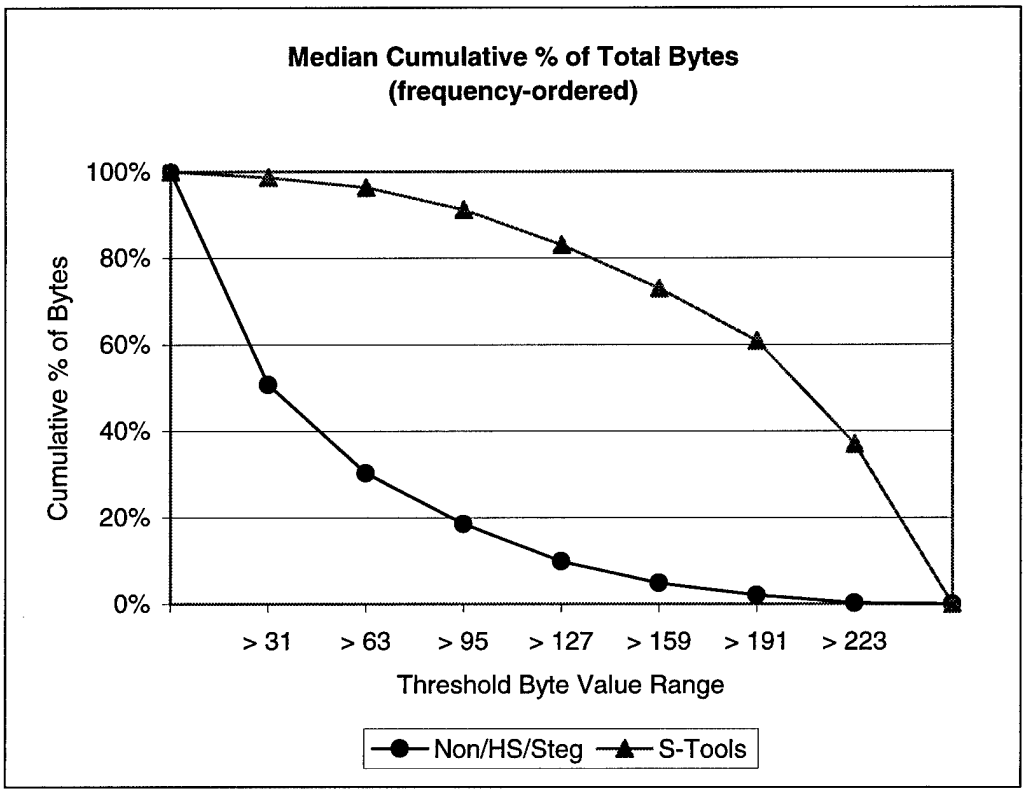


Figure 46, Cumulative Percentage of Total Bytes (frequency-ordered)

Although selecting a lower minimum percentage would increase the probability of success, a user's acceptable level of Type I and Type II errors must factors into selecting

the appropriate *(Target Percentage, Threshold Value)* pairing. Selecting a target percentage above the upper line would likely result in success rates worse than *coin-flip* probabilities, since the line indicates points where 50% of the population are above and 50% are below. Therefore, the optimal or target percentage would lie somewhere between the two so-called exceedence *curves*. As the target percentage decreases towards the lower line, the number of Type I errors decreases, but the number of Type II errors increases.

A reasonable technique to employ for obtaining an effective pairing is to split the difference between the two sets of data points. In Figure 47, the middle data points represent the differences between the upper and lower cumulative percentages. Using these values, the success rate of the byte frequency test on the frequency-ordered images is shown in Table 23.

Table 23, Target Percentage / Threshold Value Success Rates (frequency-ordered)

Target Percentage	Threshold Value	S-Tools	Non-Stego
74.71%	> 31	100.00%	0.00%
63.35%	> 63	100.00%	0.00%
54.88%	> 95	100.00%	0.00%
46.43%	> 127	100.00%	0.00%
38.90%	> 159	100.00%	0.00%
31.43%	> 191	99.44%	0.00%
18.64%	> 223	96.67%	0.00%

The lower set of data points in Figure 47 represents a reasonable lower bound for selecting a target percentage. The success rates using this new set of pairings is shown in

Table 24. The percentages listed in the non-stego column are the rates at which non-stego files are detected and represent Type II errors.

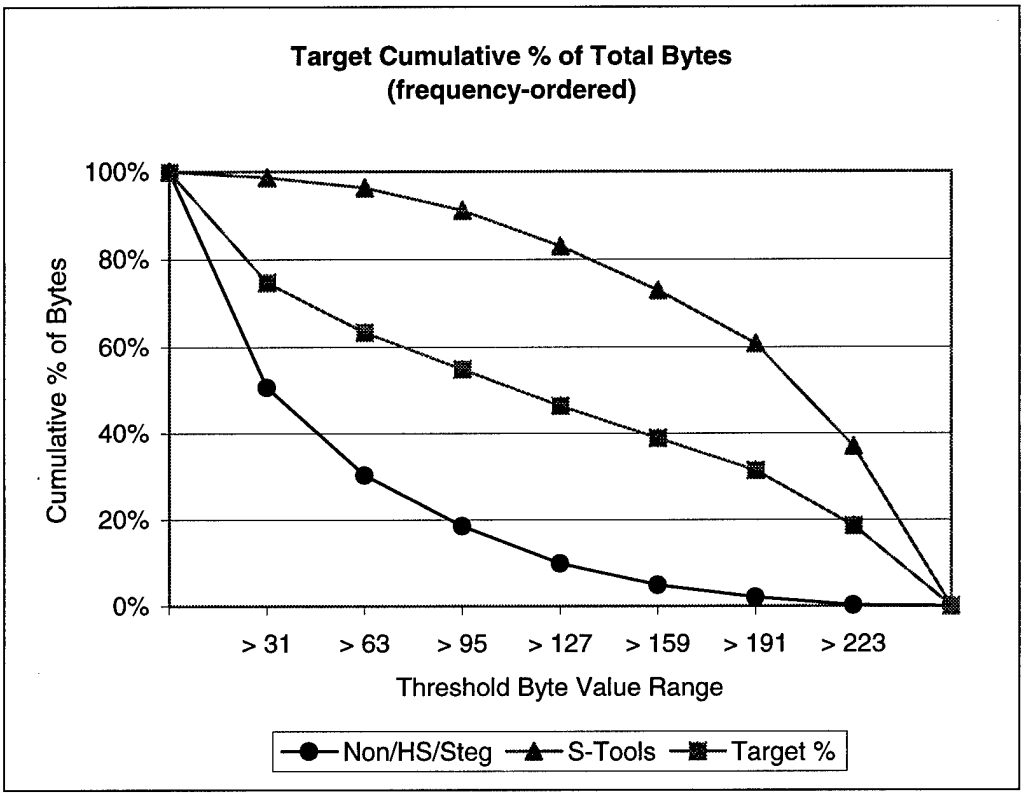


Figure 47, Target Percentage of Total Bytes (frequency-ordered)

Table 24, Target Percentage / Threshold Value Success Rates (frequency-ordered)

Target Percentage	Threshold Value	S-Tools	Non-Stego
50.72%	> 31	100.00%	50.00%
30.35%	> 63	100.00%	60.00%
18.58%	> 95	100.00%	46.67%
9.83%	> 127	100.00%	50.00%
4.88%	> 159	100.00%	50.00%
2.08%	> 191	100.00%	50.00%
0.25%	> 223	100.00%	100.00%

Conversely, if the upper set of data points are used, the rate of Type I errors increases as the rate of Type II errors decreases. The success rates using the upper set of data points as target percentages are shown in Table 25. The results from Table 23 support the claim that the optimal target percentages for given threshold values lie near the middle data points.

Table 25, Target Percentage / Threshold Value Success Rates (frequency-ordered)

Target Percentage	Threshold Value	S-Tools	Non-Stego
98.70%	> 31	45.00%	0.00%
96.35%	> 63	50.56%	0.00%
91.18%	> 95	45.56%	0.00%
83.02%	> 127	40.00%	0.00%
72.93%	> 159	43.89%	0.00%
60.78%	> 191	38.33%	0.00%
37.03%	> 223	43.89%	0.00%

5.3.2 Luminance-ordered Partition

5.3.2.1 Overall Results

Once again, a luminance-ordered palette resulted in a less definitive demarcation between non-stego images and those embedded using *HideSeek* and *Steganos*. Unlike the smooth distribution shown in Figure 41, ordering the palette by luminance value resulted in a less deterministic distribution of byte value frequencies. The byte value frequency distributions for the non-stego, *HideSeek*, and *Steganos* test cases are shown in Figure 48 through Figure 50, respectively.

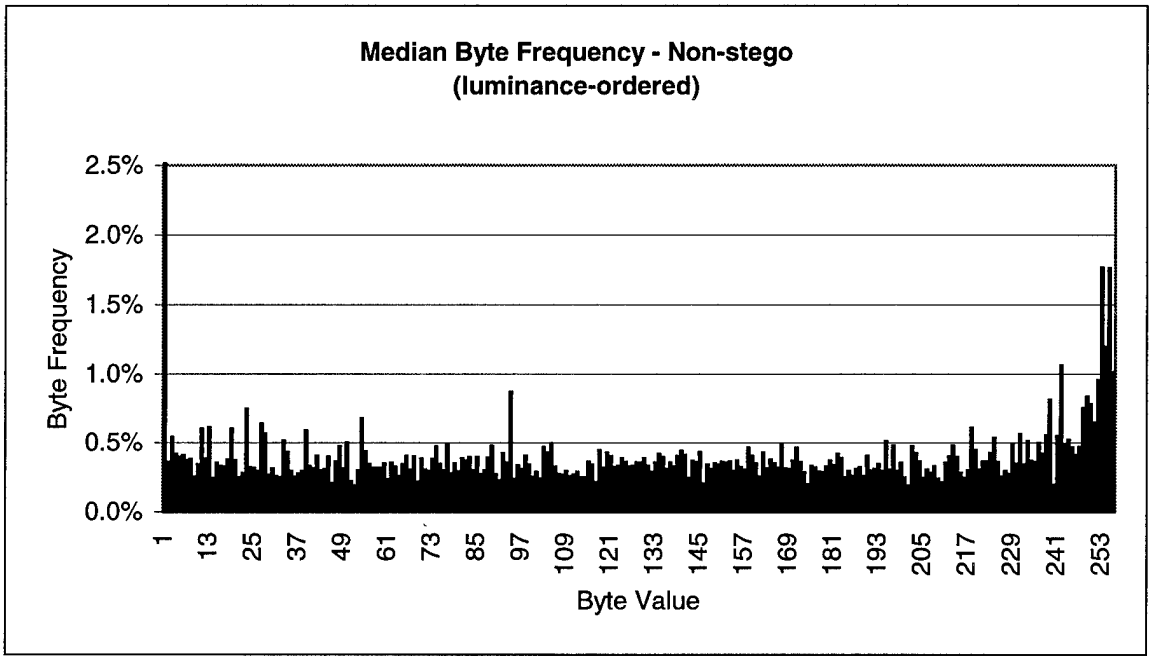


Figure 48, Median Byte Frequency – Non-stego (luminance-ordered)

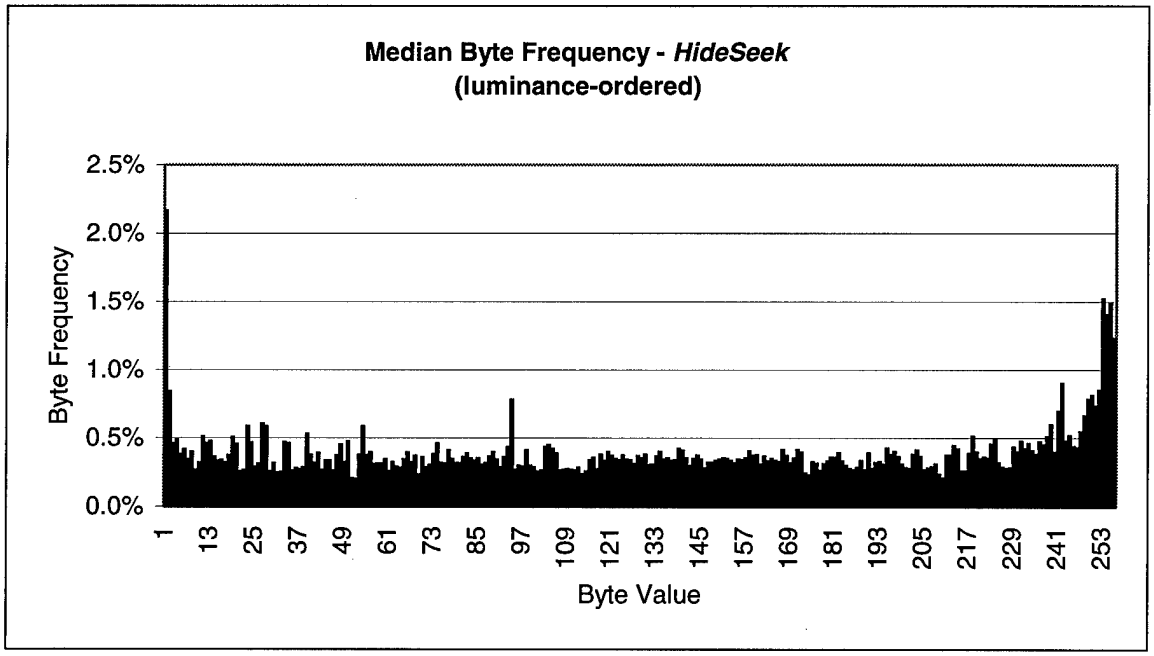


Figure 49, Median Byte Frequency – HideSeek (luminance-ordered)

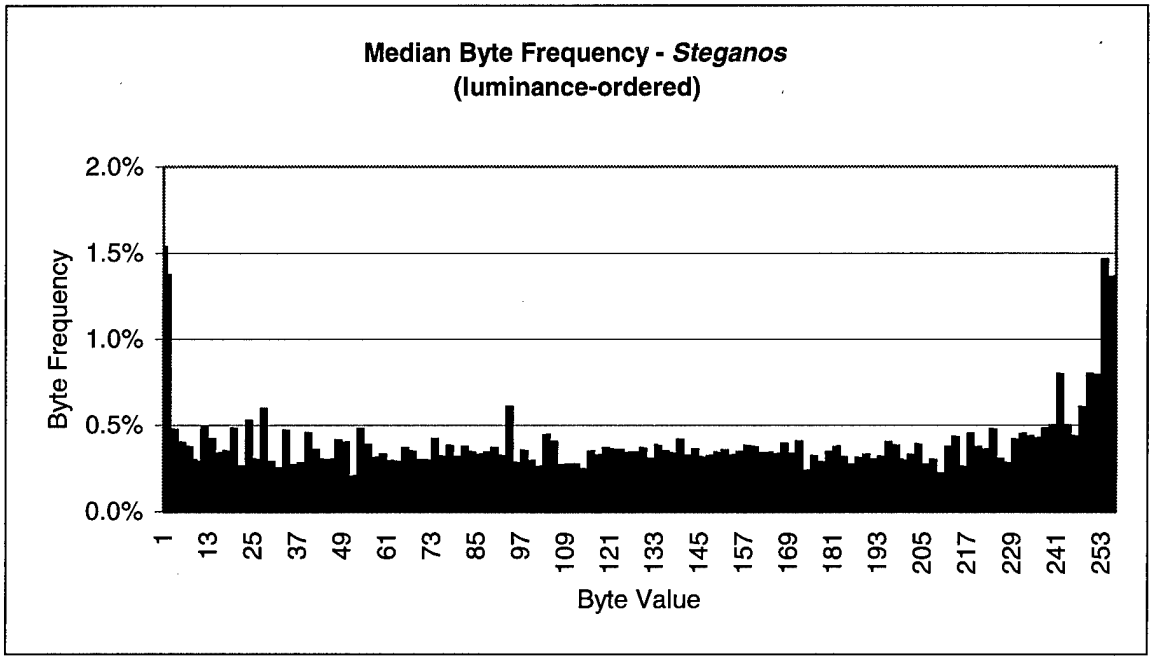


Figure 50, Median Byte Frequency – Steganos (luminance-ordered)

Once again, the byte frequencies almost mirror one another. The slight variations in the *HideSeek* and *Steganos* files from the non-stego files are due to the least-bit substitution method of embedding. The resulting new indices are either above or below the old indices, and by no more than a single byte value.

Surprisingly, the *S-Tools* files have nearly identical byte value frequency distributions as their frequency-ordered equivalents. This suggests that the method used by *S-Tools* to reduce and then repopulate the palette is relatively unaffected by the palette ordering scheme. The frequency distribution for the *S-Tools* files is shown in Figure 51.

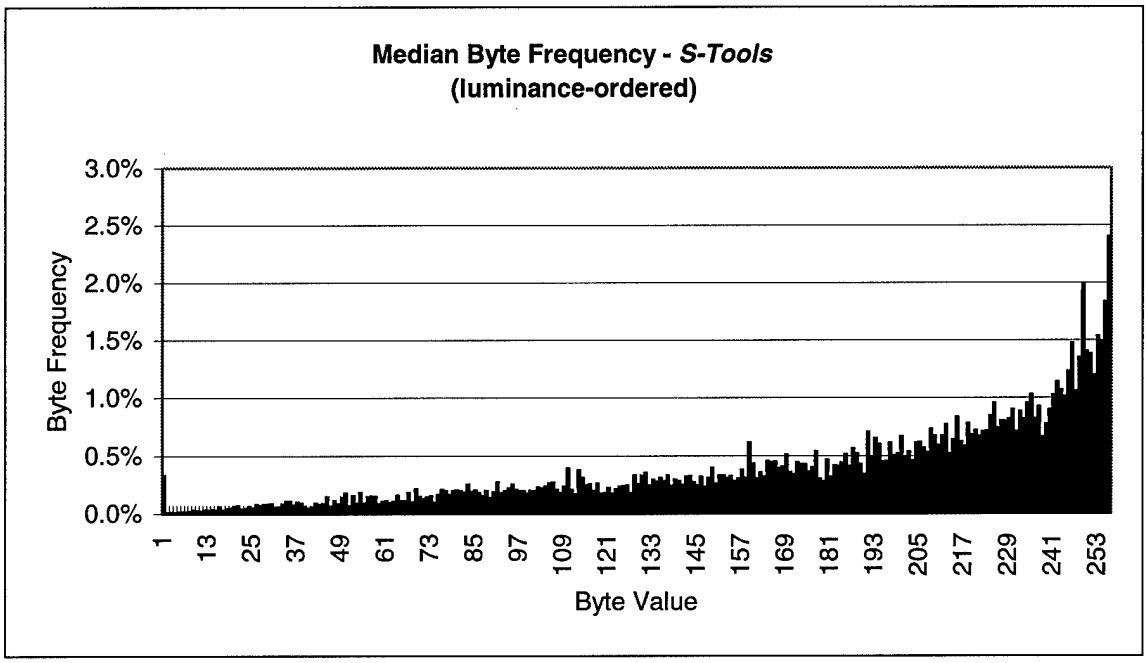


Figure 51, Median Byte Frequency – S-Tools (luminance-ordered)

5.3.2.2 Target Percentage Selection and Results

Using the techniques outlined in Section 5.3.1.2, the byte frequencies are combined into eight bins as shown in Figure 52. Once again, the negligible differences between the non-stego, *HideSeek*, and *Steganos* files are highlighted.

In order to obtain a minimum cumulative percentage of bytes that exceed a particular threshold byte value, the median percentages are plotted as shown in Figure 53. The median of the non-stego, *HideSeek*, and *Steganos* files are combined and compared against the *S-Tools* files.

In Figure 54, the differences between the upper and lower data points is plotted. The new mean cumulative percentages are then used to determine the success of the test on the luminance-ordered test library. The results, shown in Table 26, again support the

claim that the byte frequency test works well to determine *S-Tools* embedded images. However, the amount of Type II errors represented by the non-stego column is significantly higher than in the frequency-ordered test partition (shown in Table 23).

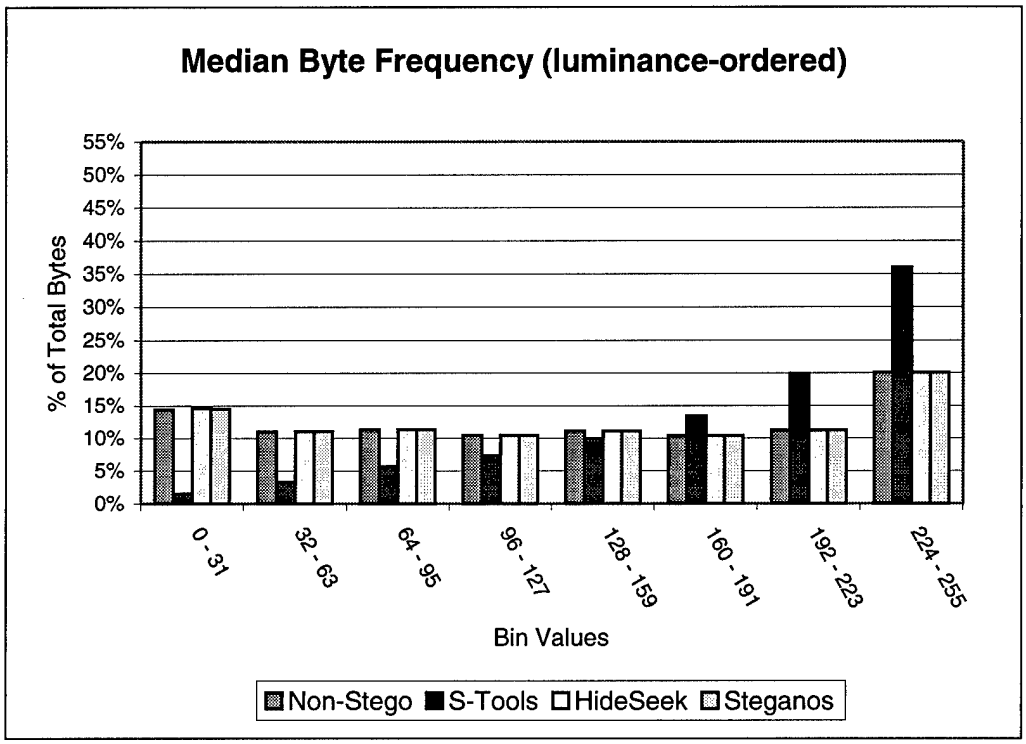


Figure 52, Combined Median Byte Frequency (luminance-ordered)

If the lower and upper sets of data points are used as target percentages, the test yields the success rates shown in Table 27 and Table 28, respectively. When the upper set of data points is used, the rate of Type I errors again increases as the rate of Type II errors decreases.

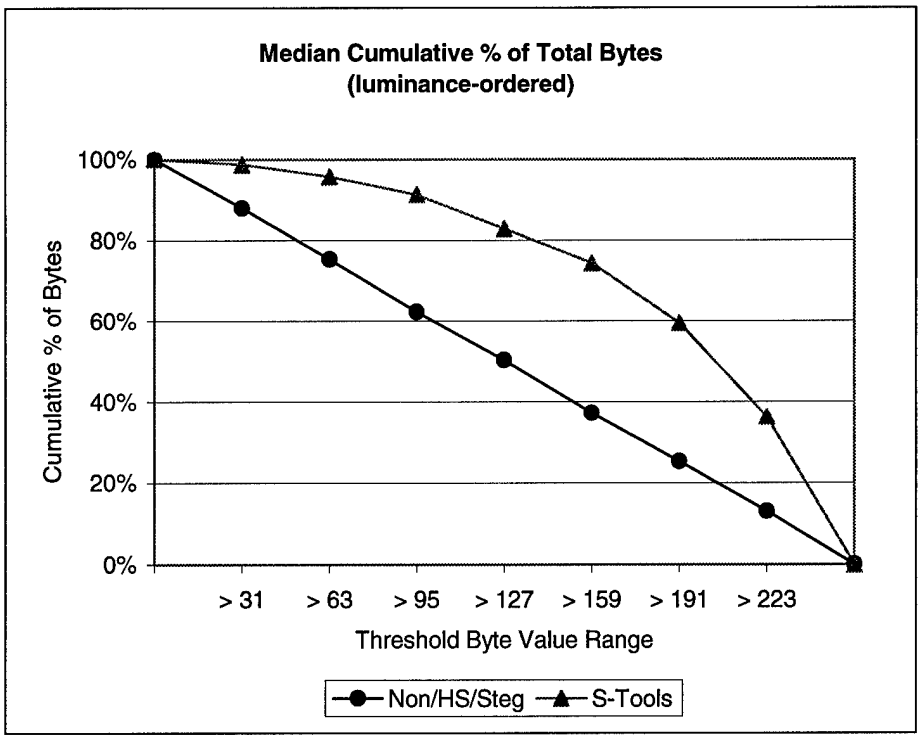


Figure 53, Cumulative Percentage of Total Bytes (luminance-ordered)

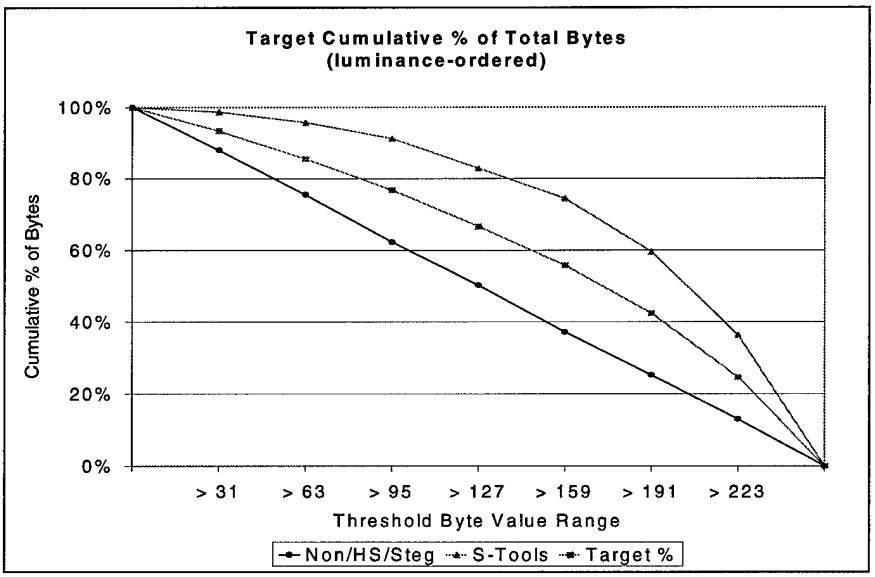


Figure 54, Target Percentage of Total Bytes (luminance-ordered)

The results below again show that the optimal target percentages for given threshold values lie near the middle data points.

Table 26, Target Percentage / Threshold Value Success Rates (luminance-ordered)

Target Percentage	Threshold Value	S-Tools	Non-Stego
93.34%	> 31	96.11%	13.33%
85.54%	> 63	92.78%	13.33%
76.74%	> 95	91.67%	16.67%
66.56%	> 127	89.44%	16.67%
55.77%	> 159	82.78%	16.67%
42.38%	> 191	83.33%	16.67%
24.64%	> 223	75.00%	26.67%

Table 27, Target Percentage / Threshold Value Success Rates (luminance-ordered)

Target Percentage	Threshold Value	S-Tools	Non-Stego
87.99%	> 31	100.00%	50.00%
75.39%	> 63	100.00%	60.00%
62.23%	> 95	100.00%	66.67%
50.26%	> 127	100.00%	60.00%
37.19%	> 159	100.00%	60.00%
25.24%	> 191	100.00%	60.00%
13.02%	> 223	100.00%	50.00%

Table 28, Target Percentage / Threshold Value Success Rates (luminance-ordered)

Target Percentage	Threshold Value	S-Tools	Non-Stego
98.69%	> 31	47.22%	0.00%
95.69%	> 63	48.89%	3.33%
91.25%	> 95	42.22%	3.33%
82.86%	> 127	43.33%	3.33%
74.35%	> 159	41.67%	6.67%
59.52%	> 191	42.22%	6.67%
36.26%	> 223	46.11%	13.33%

5.4 Ancillary Results

So far, only the differences between luminance and frequency ordered palettes have been considered as factors affecting detection. As indicated in Section 4.1.3.5, two other factors are also examined in an effort to discover rudimentary relationships and their impact on steganalysis. Each of these two factors – characteristics of the message file, and cover file loading levels – is discussed below within the context of the selected strategies.

5.4.1 Point-noise Threshold Test

5.4.1.1 Message File Composition

Three *types* of message files were used – binary, image, and text files. There were no significant differences in the hit percentages between the three file types for a given tool and threshold. This was true in both the luminance and frequency ordered palette test partitions. See Appendix B for results of each test case.

5.4.1.2 Cover File Loading Level

The two loading levels, designated *High* and *Low*, were described in Section 4.1.3.5. Noticeable differences in the hit percentages were detected only in the case of *HideSeek* embedded images, but not for the other two tools evaluated. This was true only for the frequency-ordered test partition; differences among luminance-ordered images remained negligible or non-existent. The median hit percentages in the *HideSeek* test cases for each threshold tested are shown in Figure 55.

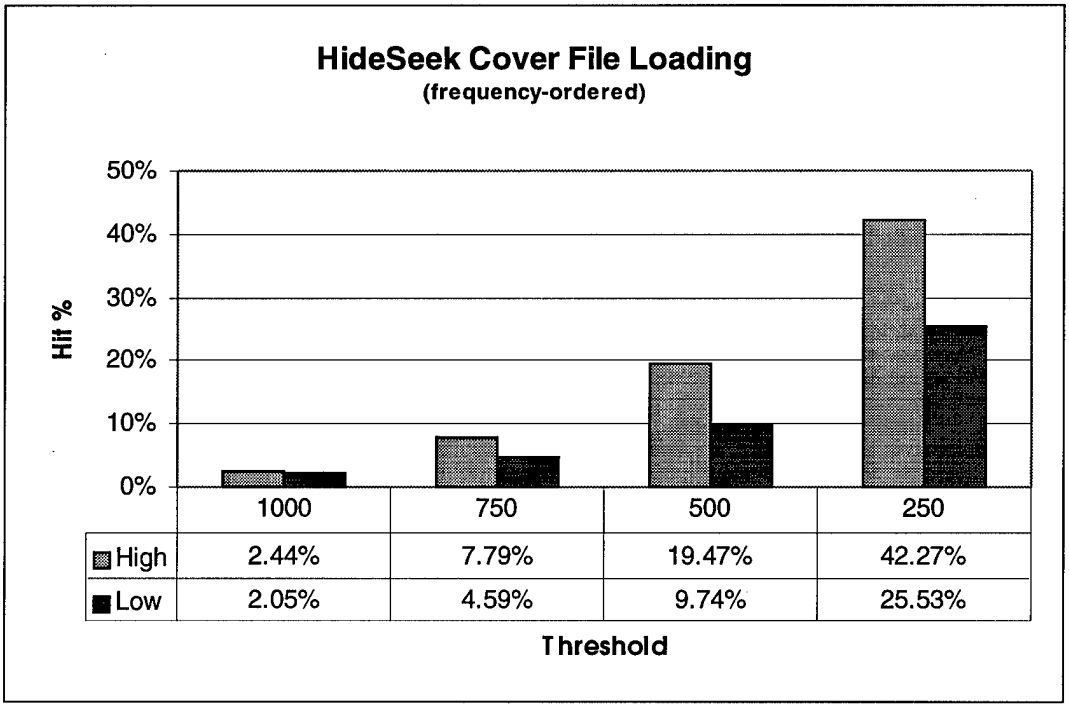


Figure 55, HideSeek due to Cover File Loading (frequency-ordered)

Since *Steganos* and *HideSeek* embed information using very similar techniques, it was surprising to discover *Steganos* did not display the dramatic differences between high and low loading. The most likely explanation lies with *Steganos's* use of compression. Since *Steganos* always compresses files prior to embedding them, the

results of this test for the loading parameter cannot be used for an unbiased comparison against *HideSeek*. The median hit percentages in the *Steganos* test cases for each threshold tested are shown in Figure 56. See Appendix B for detailed results of each test case.

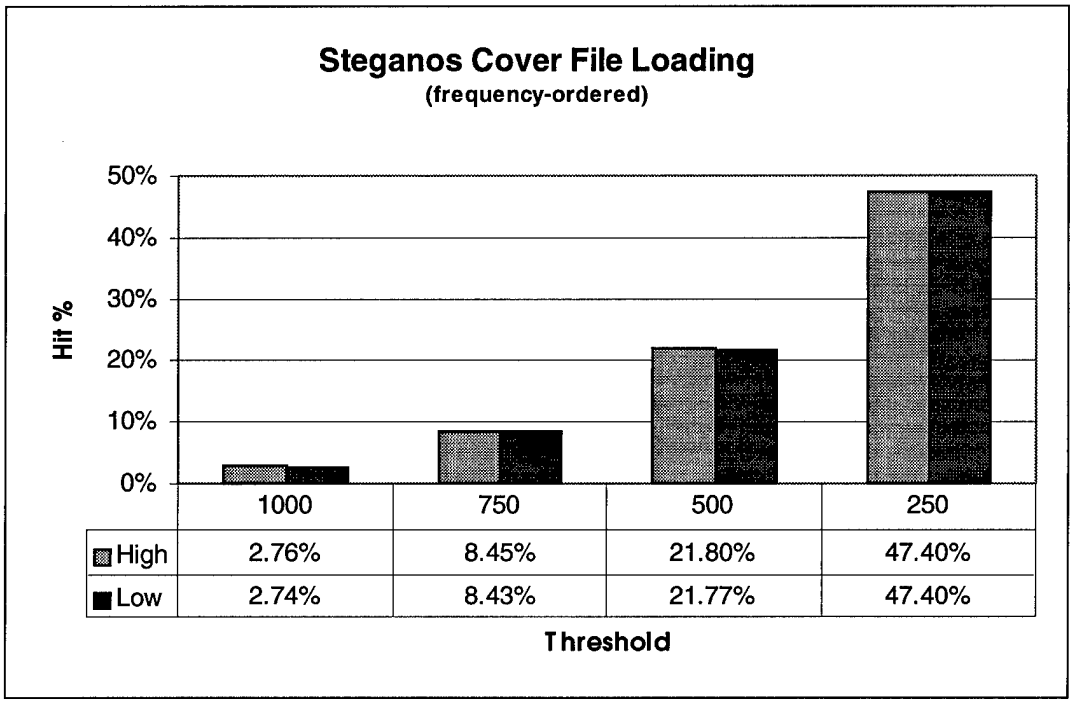


Figure 56, Steganos due to Cover File Loading (frequency-ordered)

5.4.2 Byte Frequency Analysis

5.4.2.1 Message File Composition

Once again, there were no appreciable differences in byte frequency distribution between the three message file types within the test cases. This was true across tool boundaries and palette ordering test partitions. See Appendix C.2.1 for the complete results of the byte frequency test with respect to message file composition.

5.4.2.2 Cover File Loading Level

The cover file's loading level noticeably affected only the *S-Tools* test cases in both test partitions. (See Figure 57, below.) The S-*Tools* files displayed a greater percentage of higher byte values in the low loading cases than they did during high loading. It is not surprising that both test partitions were affected similarly since *S-Tools* appears to be unaffected by palette ordering. The reasons for the differences due to the loading are not intuitive and require further investigation before they can be used as a discriminator in an automated detection process. See Appendix C.2.2 for the complete results of the byte frequency test with respect to cover file loading.

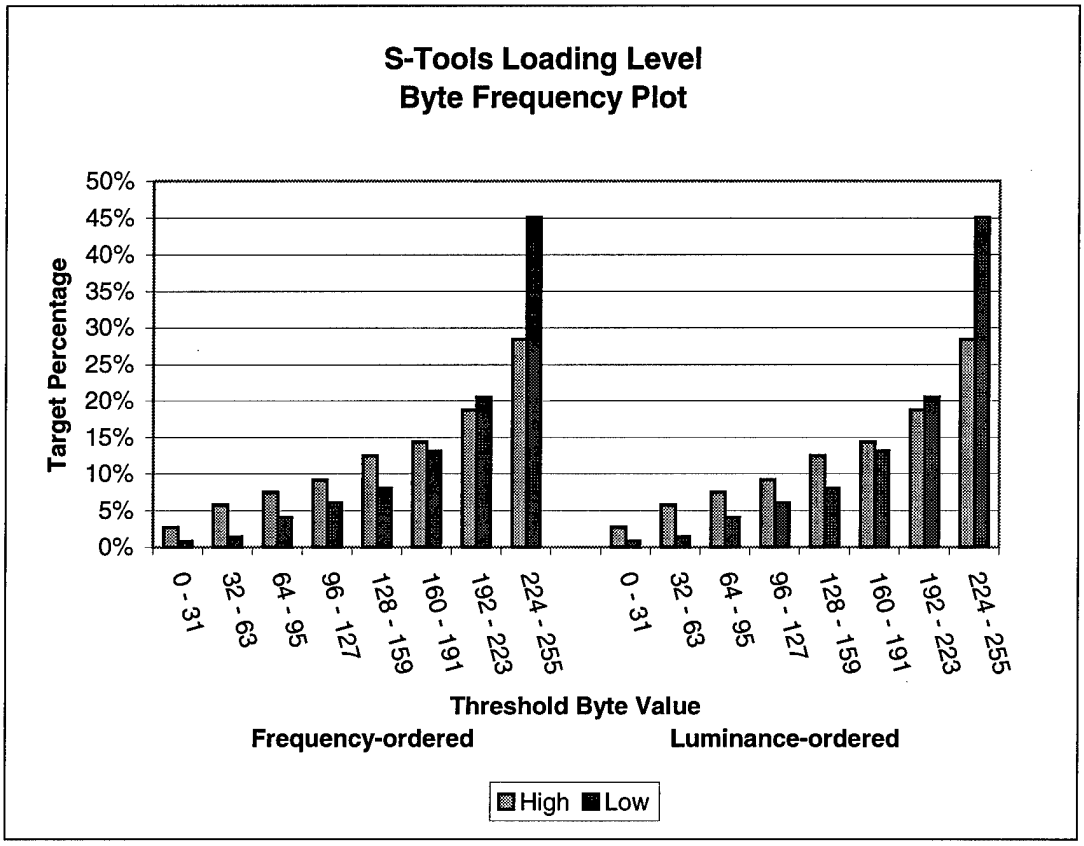


Figure 57, S-Tools due to Cover File Loading (combined test partition)

VI Conclusion and Recommendations

6.1 Conclusion

The tests developed and utilized in this research represent two viable strategies for detecting the presence of embedded information: through visible distortions and byte frequency analysis. Neither test proved to be a panacea; however, each proved effective in a subset of the overall problem space. A strategy was considered effective if the differences between identical statistics of a stego and non-stego file or image were large enough to make a detection rule from it. Minimal consideration was given to the amount of Type I and Type II errors, since it involves a tradeoff that is best left to the user's discretion. The effectiveness of each strategy is summarized in Table 29 according to test partition and steganography tool. Their combined effectiveness – that is, when the strategies are considered as complimentary tools – is shown in Table 30.

Table 29, Strategy Effectiveness

Strategy	**Frequency-ordered Palette**			**Luminance-ordered Palette**		
	HideSeek	*Steganos*	*S-Tools*	*HideSeek*	*Steganos*	*S-Tools*
Point-noise Threshold Test	✔	✔				
Byte Frequency Analysis			✔			✔

✔ Indicates an effective strategy

Table 30, Combined Effectiveness

Frequency-ordered Palette			Luminance-ordered Palette		
HideSeek	*Steganos*	*S-Tools*	*HideSeek*	*Steganos*	*S-Tools*
✔	✔	✔			✔

✔ Indicates an effective strategy

Furthermore, since *HideSeek* and *Steganos* represent similar file manipulation and embedding techniques, their results might be considered synonymous. *S-Tools*, on the other hand, is relatively unique in its methods and is representative of a different category of steganography tools. The effectiveness of the combined tool set when categories of tools are considered is shown in Table 31.

Table 31, Combined Effectiveness with Tool Characterization

Frequency-ordered Palette		Luminance-ordered Palette	
HS/Steg	*S-Tools*	HS/Steg	*S-Tools*
✔	✔		✔

✔ Indicates an effective strategy

6.2 Validity

Any discussion of effectiveness should consider the limitations of the research that produced the results. The problem space examined in this research consisted of eight-bit images embedded using the least-significant bit substitution (LSB) method. While not all-inclusive, it likely represents a preponderance of real-world steganography cases since eight-bit images and LSB are both widely used. Also, the steganography

tools evaluated here are continually being improved. The methods used in this research may become less effective as more advanced steganography tools are released. Even so, the strategies of looking for visible distortions and analyzing byte frequency distributions are likely to be part of any future automated detection process.

This research ultimately produced both expected and unexpected results. In the case of the point-noise threshold test, *HideSeek* and *Steganos* were expected to be vulnerable since both tools produce images that are visibly distorted when the palette is ordered by frequency of occurrence. However, the degree to which palette ordering impacts the results was more extreme than anticipated.

The byte frequency analysis test results were more unexpected, overall. Since palette ordering had such an extreme impact on the success of the previous test, it was anticipated to have similar influence on this test. However, luminance-ordered *S-Tools* images revealed similar vulnerabilities as frequency-ordered images. This result is significant because it means an entire category of steganography products, namely *S-Tools*, can be reliably detected regardless of palette ordering. If an investigator had other clues as to the method of embedding, such as *S-Tools* being loaded on the suspect system, then the target images could be retrieved for further examination.

With regard to the ancillary results, it was expected that cover file loading levels and underlying message file composition would have a greater impact than shown. Certainly, there are differences, but they are at a much finer level of detail than would be expected in an automated tool using these tests. Perhaps as reliable signatures of *normal* files are developed, the differences due to loading and message file composition will become more significant.

The goal of this research was to determine viable automated strategies for detecting the presence of embedded information in graphics files. From the results presented above, explicit parameters for detection are possible. For instance, from Figure 47 we can conclude a file likely contains embedded information if more than 50 percent of its bytes are above 127 in value. The results also comply with the blind steganalysis requirement since they do not use the original characteristics of the image or file for detection.

6.3 Application of Results

This research supports the claim that steganography which defeats the eye is susceptible to statistical analysis of the file, and vice versa. The point-noise threshold test is significant because it automates a process that is typically performed manually by viewing a rendered image. In that case, even the byte frequency analysis test reveals indicators that would normally be detected by viewing the image's palette. Consequently, any useful detection toolkit should include tests for both cases.

During this research, each test was performed in a rather labor-intensive manner. Data was collected using tools and scripts, but the analysis was conducted manually. However, now that detection parameters are known, the problem of creating an automated process is simply a matter of creating the interface.

Consideration should be given to increasing the speed of the point-noise thresholding process. The scripts used in this research were executed under MATLAB v5.2 and would not be as helpful as compiled applications. Before the utility of MATLAB was fully realized, several such routines were developed in C and used to manually prepare and convert the images for thresholding. Appendix D contains

instructions for obtaining C-code for a routine that converts an RGB bitmap to grayscale. Also available is code that converts a bitmap image to a frequency-ordered file, and conversely to a luminance-ordered file. Fortunately, MATLAB scripts and functions are C-like in structure and convention, so the conversion from MATLAB to a compiled routine should be trivial.

The output from each of the tests used in this research is a pairing of parameters – a parametric tuple, so to speak. The point-noise threshold test produced optimal *(Threshold, Hit Percentage)* pairings, and the byte frequency analysis test provided *(Target Percentage, Threshold Values).* Therefore, an automated tool should incorporate these findings through the use of a two-variable input mechanism. This concept is presented in Figure 58 and Figure 59 below. Armed with automated tools such as these and the results of this research, an investigator would be able to sift quickly and reliably through extraordinary amounts of evidence to segregate files of interest.

6.4 Recommendations for Future Work

Since this research reflects a high-level view on the prospects of detecting embedded information through the use of automated processes, surely more work remains. A number of recommendations for future efforts are presented here. Each respects the blind steganalysis problem, so they do not include such things as known-cover or known-message attacks. Also, they only deal with graphics file cover medium.

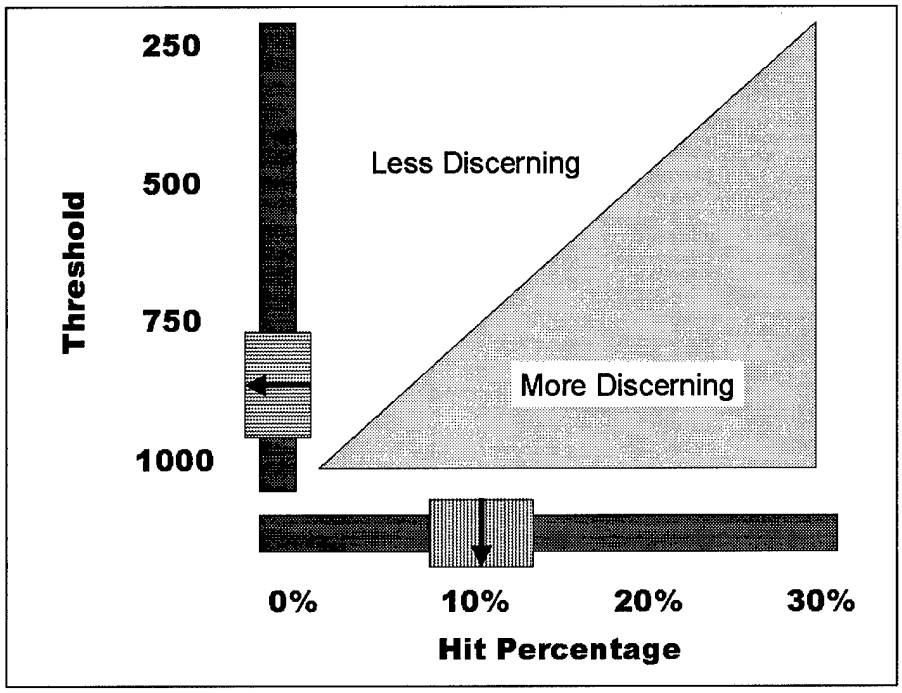


Figure 58, Point-noise Threshold Test Two-variable Input Mechanism

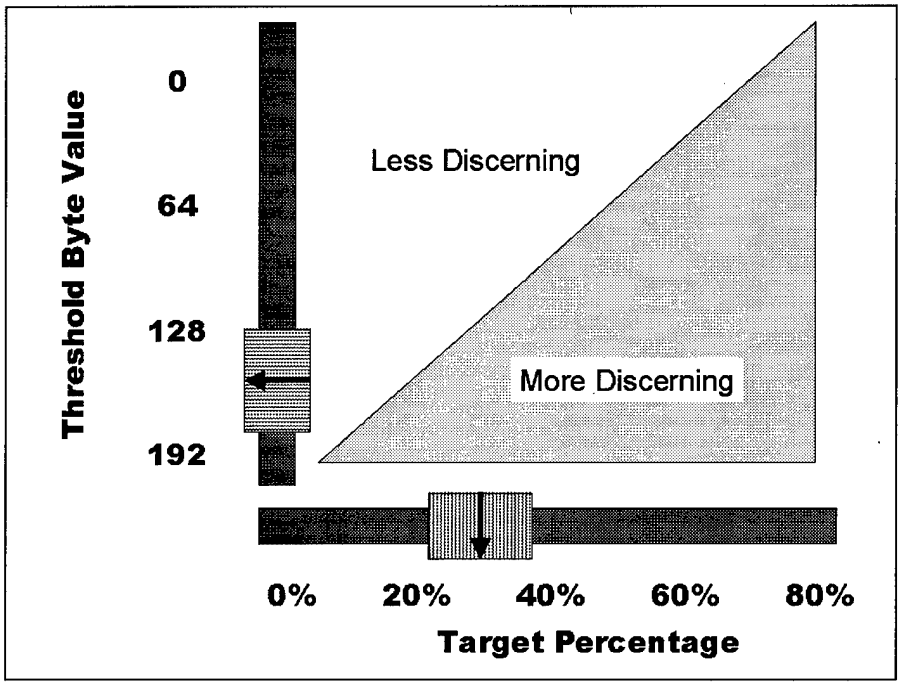


Figure 59, Byte Frequency Analysis Test Two-variable Input Mechanism

6.4.1 Characterization of Cover Images

The results of this research suggest that the images in the test library were *different;* that is, they do not represent a homogenous sample. Some images had predominantly dark colors, while others were light. Some were more uniformly colored, while others contained a broader mix of hues. Still others contained natural colors (photographs), while others consisted of artificial tones (artwork).

While no attempt was made to characterize the images *a priori*, the differences seemed to affect the statistics enough to decrease their predictability. Data points that at first appeared to be outliers may actually be valid for images that share a common set of characteristics.

The lack of image characterization made analysis of the mean statistic unfeasible. In fact, the reason the median statistic is used throughout is due to its decreased susceptibility to outliers. Very few files had statistics that were normally distributed, and the remaining files were not all equally distributed. This made an analysis of the variance of the mean impossible, as well as precluded the use of non-parametric tests.

Characterizing images and defining categories would serve two purposes. First, it may reveal a finer level of detail than was possible using a heterogeneous sample. Secondly, it would allow tests to be optimized for particular *types* of images. Filters, thresholds, and percentages would be matched to these categories and result in more reliable tests.

6.4.2 Population Statistics and File Signatures

Related to characterizing images is the idea of collecting enough sample statistics to better represent the true population statistics. Future research in this area should

contribute to the overall database of file statistics so that the sample is large enough to more reliably reflect the true population statistics. As a corollary to this, work should be done to accurately identify the statistics that represent a typical, unembedded file according to its kind. Bitmap files will have a different signature than word processing documents, and both are likely to be different from spreadsheet or CAD/CAM files. As these file signatures become more precise, detecting embedded information would be reduced to comparing a file against the signature of the file it purports to be.

6.4.3 Expanding the Problem Space

Certainly, a logical step in future research would be to apply the techniques of this research to a larger problem space. This includes examining their effectiveness against different steganography tools, as well as various pixel depths. Since the palette plays such a crucial role in the success of these tests, it would be interesting to see the results when they are applied to 24-bit images, which have no palettes, or grayscale images that have uniformly distributed palettes.

Future research might also consider if detection indicators exist in the image transform space. This is related to the idea of characterizing images, except the image is examined using common signal processing techniques such as the Fast Fourier Transform.

6.4.4 Parameter Modification

The tests conducted during this research were done with several fixed parameters. Subsequent research should study the affects of modifying these parameters. Such modifications include the following:

- Change the filter dimensions and/or coefficients used in the point-noise threshold test.
- Expand the byte frequency analysis to polybyte and polybit combinations. While this is likely to be necessary with 24-bit images since a pixel is represented by three bytes instead of a single byte palette index, it may also have utility applied to lower pixel depths.

Appendix A, Test Programs

A.1, Byte Count Utility (bytecnt.c)

A.1.1, Overview

The *bytecnt.c* program is used to collect data for the byte frequency analysis test. It takes as input a list of bitmap file names, minus their extensions, and processes each file in turn. Since there are only 256 possible byte values, the totals for each byte value are stored in a 256 record array. The byte value that is read from the bitmap file is used to index the array.

The output of this utility is a comma-delimited, ASCII text file named *bytecnt.dat.* In order to facilitate import into MS-Excel, each file is represented by a single column in the output file. Each line, therefore, contains the totals for a given byte value for each file processed. A portion of a sample output from this utility is shown in Section A.1.3.

A.1.2, Program Code

```
/*==================================================================
   FILE:     bytecnt.c
   AUTHOR:   Capt Chris Fogle
             AFIT, GCS-99M

   DESCRIPTION:  This program facilitates a basic byte analysis
   of a file.  It is a generic routine that counts the quantity
   of each byte value in a file and outputs the totals to a text
   file.  The text file can then be processed by Excel or MATLAB.

   ASSUMPTIONS:
   -- The file for reading is present in the current directory.
   -- The files for reading have a ".bmp" extension.
   -- The filenames to process are in a text file named
      "files.txt"
   -- Limited to 40 files.
==================================================================*/

#include <stdio.h>

#define NAMESIZE 40
#define MAXFILES 40

int main(int argc, char **argv) {

   /*  ---------------- Function Prototypes -------------------*/
   char * GetNextFileName(char *, FILE *);
   void   DoError();

   /*  --------------- Variables Definitions ------------------*/
   FILE *listfile = fopen("files.txt", "r");
   FILE *infile;      /* bitmap file   */
   FILE *outfile_byt = fopen("bytecnt.dat", "w");
                  /* output info   */

   char filename[NAMESIZE];
   char filenamebmp[NAMESIZE];
   char filenamebyt[NAMESIZE];

   long   byteCounts[256][MAXFILES];
   static int  i, j, fc;
   int    byteVal;
   static int  formatOption = 0;
                /* defaults to MATLAB output */
   /*  ----------------------------------------------------------*/

   /* check for command line parameters */
   if (argc > 2) DoError();
   if (argc == 2)
      if (strcmp(argv[1], "-X") == 0)
      formatOption = 1;
      else if (strcmp(argv[1], "-M") != 0)
           DoError();
```

```
    /* initialize the array space */
    for(i=0; i<256; i++)
       for (j=0; j<MAXFILES; j++)
             byteCounts[9][10] = 0;

    /*  process files  */
    printf("Creating output file... \n", filename);
    fc = 0;
    while( GetNextFileName(filename, listfile) != NULL ) {
       strcpy(filenamebmp, filename);
       infile = fopen((char *) strcat(filenamebmp, ".bmp"), "rb");
       printf("Processing data for    %s...\n", filename);
       while ( !feof(infile) ) {
          byteVal = fgetc(infile);
          byteCounts[byteVal][fc] ++;
       }
    close(infile);
    fc++;
    }

    printf ("Outputting byte count array in %s format...\n\n",
          (formatOption == 0 ? "MATLAB" : "EXCEL"));

    switch (formatOption) {
    case 0:  /* MATLAB output */
       for(i=0; i<256; i++) {
          for (j=0; j<fc; j++)
             fprintf(outfile_byt, "%d ", byteCounts[9][10]);
          fprintf(outfile_byt, "\n");
       }
       break;

    case 1:  /* Excel output */
       for (i=0; i<256; i++) {
          for (j=0; j<fc; j++) {
             fprintf(outfile_byt, "%d", byteCounts[9][10]);
             if (j<(fc-1)) fprintf(outfile_byt,",");
          }
          fprintf(outfile_byt, "\n");
       }
       break;
    }

    printf("Done.\n\n");
    close(listfile);
    close(outfile_byt);
}
/*============================================================*/
```

```
/* This routine extracts the next filename from the list of
/  files to process.  It assumes the file of filenames is already
/  opened.  It strips the '\n' (newline) character from the
/  filename so that it can be used for further processing.
*/

char * GetNextFileName(char * filename, FILE *listfile) {
   int  i=0;

   if (fgets(filename, NAMESIZE, listfile) != NULL ) {
      i=0;
      /* strip off the newline char */
      while (filename[9] != '\n') i++;
      filename[9] = '\0';
      return(filename);
   } else return(NULL);
}
/*=============================================================*/

void DoError()
{
    printf("\nImproper command.\n");
    printf("USAGE:  bytecnt [-X or -M]\n");
    printf("\n");
    printf("X -- output in Excel format, comma delimited, and row
            aligned\n");
    printf("M -- output in MATLAB matrix format, and column
            aligned\n\n");
    exit(-1);
}
/*=============================================================*/
```

A.1.3, Sample Output – bytecnt.c

Within the ASCII text file, the data appears as follows:

```
296,293,1010,296,1307,1075,1001,294,1416,296
13,1,25,10,1,13,6,5,14,153
13,2,36,98,2,16,118,4,27,1
2,2,10,13,1,28,25,4,33,3
14,1,25,97,0,41,12,9,19,1
11,11,5,542,5,2,26,52,189,3
77,25,3,530,1,73,12,25,187,10
24,62,118,102,3,6,6,14,183,7
17,60,6,9,26,60,151,112,178,14
2,51,148,27,2,41,6,119,152,157
66,20,46,59,120,22,41,5,993,820
```

Once the file is imported into an Excel spreadsheet, it is stored as shown below.

Each column represents a single test file:

Byte Value	File 01	File 02	File 03	File 04	File 05	File 06	File 07	File 08	File 09	File 10
00	296	293	1010	296	1307	1075	1001	294	1416	296
01	13	1	25	10	1	13	6	5	14	153
02	13	2	36	98	2	16	118	4	27	1
03	2	2	10	13	1	28	25	4	33	3
04	14	1	25	97	0	41	12	9	19	1
05	11	11	5	542	5	2	26	52	189	3
06	77	25	3	530	1	73	12	25	187	10
07	24	62	118	102	3	6	6	14	183	7
08	17	60	6	9	26	60	151	112	178	14
09	2	51	148	27	2	41	6	119	152	157
10	66	20	46	59	120	22	41	5	993	820

A.2, MATLAB Threshold Test Function (scandir.m, getstats.m)

A.2.1, Overview

These MATLAB functions work under MATLAB v5.2 and represent the top-level function in the point-noise threshold test. The first, *scandir*, relies on several vendor provided functions to perform the filter operation and various conversions of the bitmap image. The user-defined function, *getstats*, calculates the number of *hits* in the threshold image and the hit percentage.

The output from *scandir* is a space-delimited, ASCII text file. Each output data file represents a single test case. Each line in the data file represents the hit percentage statistics for a single file. In addition to compiling the hit percentage statistics, this function also saves the resulting grayscale and threshold images as bitmap files for further visual examination.

A.2.2, Function Code

A.2.2.1, scandir.m

```
function filtimg = scandir(mask, threshold, start, stop)

%  This function launches the data collection routines for the
%  point-noise filter trials. It takes as input a mask which
%  indicates which data files to operate on, and a threshold
%  value which is applied to the threshold routine in the
%  filter operation.
%
%  USAGE:  scandir(mask, threshold, start, stop)
%
%     mask      -> usually 'mY.bmp'
%     threshold -> filter threshold
%     start     -> file to start with (01 to 30)
%     stop      -> file to stop at (01 to 30)
%
%  Assumes:
%  -- all files begin with 01 to 30
%  -- the remainder of files' names are identical
%

mask
threshold

%point-noise filter
filtmask = [-1 -1 -1; ...
            -1  8 -1; ...
            -1 -1 -1];

dbstop if error

% initialize matrix that will hold values produced in this run
alldata = [ ];

for x = start:stop
    if x < 10
       % prepends a '0' for single digit numbers
       filename = ['0' int2str(x) mask];
    else
       filename = [int2str(x) mask];
    end

    disp(filename)
    disp('   Loading bitmap...')
    eval([ '[img pal] = imread(' '''' filename '''' ', '
 '''bmp'');' ]);

    % write back to file to fix any errors for next time
    eval([ 'imwrite(img, pal,' '''' filename '''' ', '
'''bmp'');' ]);
```

```
    % convert color bitmap to intensity map
    imap = ind2gray(img, pal);

    % convert intensity map to grayscale
    [img2 pal2] = gray2ind(imap, 256);

    % save the grayscale image to disk
    grayfilename = [ 'gray_' filename ];
    eval([ 'imwrite(img2, pal2,' '''' grayfilename '''' ', '
'''bmp'');' ]);

    % filter the image with point noise filter
    filtimg = filter2(filtmask, img2);

    % perform thresholding on the filtered image
    [maxrows, maxcols] = size(filtimg);

    disp('   Begin Threshold...')
    for r = 1:maxrows
       for c = 1:maxcols
       if abs(filtimg(r,c)) > threshold
           filtimg(r,c) = 256; %set to white
       else
           filtimg(r,c) = 1;   %set to black
       end
       end
    end


    % save threshold image as bitmap file
    threshfilename = [ int2str(threshold) 'filt_' filename ];
    eval([ 'imwrite(filtimg, pal2,' '''' threshfilename '''' ...
           ', ' '''bmp'');' ]);


    disp('   Begin Stats...')
    disp(' ')

    [pix,hits,pcnt,meanx,stddevx] = getstats(filtimg);

    disp(sprintf('   Pixels    = %7d',pix))
    disp(sprintf('   Threshold= %7d',threshold))
    disp(sprintf('   Hits      = %7d',hits))
    disp(sprintf('   Percent   = %7.6f',pcnt))
    disp(sprintf('   Mean      = %7.3f',meanx))
    disp(sprintf('   Std Dev   = %7.3f',stddevx))
    disp(' ')

    % add a new line of stats to the matrix of all files' stats
    newline = [ x pix hits pcnt meanx stddevx];
    alldata = [alldata; newline];
end

%  Write the run's data values to a text file.
datafile = [ 'Trial_' int2str(threshold) mask '.dat' ];
eval(['save ' datafile ' alldata -ascii -tabs' ]);
disp('Done.')
```

A.2.2.2, getstats.m

```
function [pixcnt,hits,percent,meanval,stdev] = getstats(x)

%  This function calculates the statistics of the output from
%   the filter/threshold operation.  It calculates the number of
%   hits in an image as a percentage of the total number of pixels
%   in the image.  It returns a matrix with  the following values:
%         [ pixelcount  hits  percentage  mean  stdev ]
%
%   USAGE:  [pixcnt,hits,percent,meanval,stdev] = getstats(img)
%

meanval = mean(x(:));
stdev = std(x(:));
pixcnt = size(x,1) * size(x,2);
[maxrows, maxcols] = size(x);
hits = 0;
for r = 1:maxrows
   for c = 1:maxcols
   if x(r,c) == 256
       hits = hits + 1;
   end
   end
end
percent = hits/pixcnt;
```

A.2.3, Sample Output – scandir.m

Within the ASCII text file, the data appears as follows:

```
1.0000000e+00    1.8684000e+05  1.6367000e+04  8.7599015e-02 2.3337749e+01  7.2091461e+01
2.0000000e+00    1.9710000e+05  4.8757000e+04  2.4737189e-01 6.4079833e+01  1.1002888e+02
3.0000000e+00    1.9976400e+05  2.0422000e+04  1.0223063e-01 2.7068811e+01  7.7252792e+01
4.0000000e+00    1.9492800e+05  5.4735000e+04  2.8079599e-01 7.2602976e+01  1.1459424e+02
5.0000000e+00    1.9953200e+05  4.6470000e+04  2.3289497e-01 6.0388218e+01  1.0778262e+02
6.0000000e+00    2.0199600e+05  1.1526000e+04  5.7060536e-02 1.5550437e+01  5.9149501e+01
7.0000000e+00    1.9360000e+05  1.4649000e+04  7.5666322e-02 2.0294912e+01  6.7438360e+01
8.0000000e+00    1.8532800e+05  2.2512000e+04  1.2147112e-01 3.1975136e+01  8.3302089e+01
9.0000000e+00    1.9800600e+05  1.8910000e+04  9.5502157e-02 2.5353050e+01  7.4946548e+01
1.0000000e+01    1.9440000e+05  1.7314000e+04  8.9063786e-02 2.3711265e+01  7.2633314e+01
1.1000000e+01    1.8330000e+05  3.1146000e+04  1.6991817e-01 4.4329133e+01  9.5768289e+01
1.2000000e+01    1.9750800e+05  3.3369000e+04  1.6895012e-01 4.4082280e+01  9.5550745e+01
1.3000000e+01    2.0000000e+05  2.3708000e+04  1.1854000e-01 3.1227700e+01  8.2428051e+01
1.4000000e+01    1.8778200e+05  3.0729000e+04  1.6364188e-01 4.2728680e+01  9.4337575e+01
1.5000000e+01    1.9126000e+05  5.9081000e+04  3.0890411e-01 7.9770548e+01  1.1782089e+02
1.6000000e+01    1.8895600e+05  2.9736000e+04  1.5736997e-01 4.1129342e+01  9.2858300e+01
1.7000000e+01    1.8369600e+05  6.4224000e+04  3.4962111e-01 9.0153384e+01  1.2159715e+02
1.8000000e+01    1.8739800e+05  4.3515000e+04  2.3220632e-01 6.0212612e+01  1.0767147e+02
1.9000000e+01    1.8528300e+05  3.7721000e+04  2.0358587e-01 5.2914396e+01  1.0267971e+02
2.0000000e+01    1.8602000e+05  2.6408000e+04  1.4196323e-01 3.7200624e+01  8.8998407e+01
2.1000000e+01    1.8322200e+05  6.5934000e+04  3.5985853e-01 9.2763926e+01  1.2238981e+02
2.2000000e+01    1.8849600e+05  4.4781000e+04  2.3757003e-01 6.1580357e+01  1.0852683e+02
2.3000000e+01    1.8515200e+05  4.6712000e+04  2.5229001e-01 6.5333953e+01  1.1075363e+02
2.4000000e+01    1.8408000e+05  3.5097000e+04  1.9066167e-01 4.9618726e+01  1.0017010e+02
2.5000000e+01    2.0165000e+05  5.3831000e+04  2.6695264e-01 6.9072923e+01  1.1280397e+02
2.6000000e+01    1.9510400e+05  6.1216000e+04  3.1376087e-01 8.1009021e+01  1.1832552e+02
2.7000000e+01    1.8424500e+05  5.2749000e+04  2.8629814e-01 7.4006025e+01  1.1526807e+02
2.8000000e+01    1.8476400e+05  4.9944000e+04  2.7031240e-01 6.9929662e+01  1.1325120e+02
3.0000000e+01    1.8566500e+05  6.2245000e+04  3.3525436e-01 8.6489861e+01  1.2038055e+02
```

After the file is imported into an Excel spreadsheet, it is stored as shown below. The statistic *mean pixel value* was used only as a general indication of the image's overall hit percentage. Since a black pixel had a value of one, darker threshold images would have mean pixel values closer to one, as well. The statistic *standard deviation* was calculated but not used in this analysis.

File Number	Total Pixels	Hits	Hit %	Mean Pixel	Std Dev
01	186840	16367	8.76%	23.34	72.09
02	197100	48757	24.74%	64.08	110.03
03	199764	20422	10.22%	27.07	77.25
04	194928	54735	28.08%	72.60	114.59
05	199532	46470	23.29%	60.39	107.78
06	201996	11526	5.71%	15.55	59.15
07	193600	14649	7.57%	20.29	67.44
08	185328	22512	12.15%	31.98	83.30
09	198006	18910	9.55%	25.35	74.95
10	194400	17314	8.91%	23.71	72.63
11	183300	31146	16.99%	44.33	95.77
12	197508	33369	16.90%	44.08	95.55
13	200000	23708	11.85%	31.23	82.43
14	187782	30729	16.36%	42.73	94.34
15	191260	59081	30.89%	79.77	117.82
16	188956	29736	15.74%	41.13	92.86
17	183696	64224	34.96%	90.15	121.60
18	187398	43515	23.22%	60.21	107.67
19	185283	37721	20.36%	52.91	102.68
20	186020	26408	14.20%	37.20	89.00
21	183222	65934	35.99%	92.76	122.39
22	188496	44781	23.76%	61.58	108.53
23	185152	46712	25.23%	65.33	110.75
24	184080	35097	19.07%	49.62	100.17
25	201650	53831	26.70%	69.07	112.80
26	195104	61216	31.38%	81.01	118.33
27	184245	52749	28.63%	74.01	115.27
28	184764	49944	27.03%	69.93	113.25
30	185665	62245	33.53%	86.49	120.38

Appendix B, Point-noise Threshold Test Results

This appendix contains summaries of each test case for the four thresholds examined. Only an example of the output is shown in Section B.2. Summary data for the frequency-ordered and luminance-ordered test partitions are shown in Sections B.3 and B.4 respectively.

B.1, Naming Conventions

A unique naming standard was employed to accurately track the files in the image library and various test cases. The filename elements are shown in Figure 60 and described in Table 32. References to an entire test case will omit the File ID element. Non-stego cases use only the first three elements and the bitmap extension.

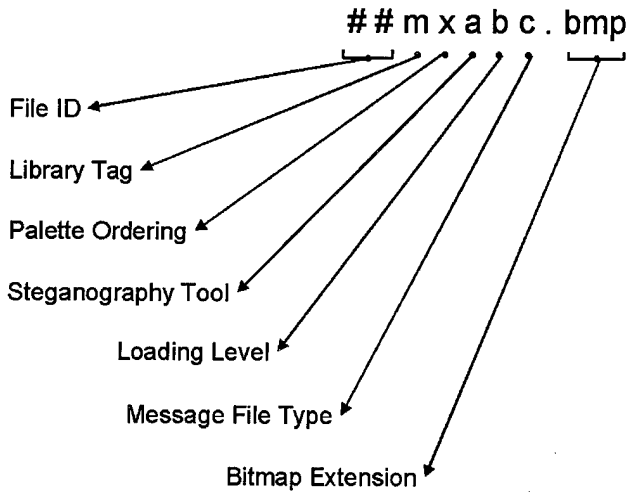


Figure 60, Naming Convention

Table 32, Naming Convention Element Descriptions

File ID: Two-digit numeric value that identified the cover file. Since there were 30 images in the master library, the values range from "01" to "30".
Library Tag: Designates which library the images were from. Since only one library (the master library) was utilized in this research, the only valid value was "m".
Palette Ordering: Designates the palette-ordering scheme. The valid values are: • "x" for luminance-ordered, • "y" for frequency-ordered palettes.
Steganography Tool: Single alphanumeric value designating the tool used to embed the message file. This element was not included in non-stego files. The valid values are: • "h" for *HideSeek*, • "g" for *Steganos*, • "s" for *S-Tools*.
Loading Level: Designates the loading level, the ratio of the total message bytes to image pixels. This element was not included in non-stego files. The valid values are: • "h" for high, • "l" for low.
Message File Type: Designates the underlying file type that is embedded in the cover file. This element was not included in non-stego files. The valid values are: • "b" for binary file, • "i" for image file, • "t" for text file.
Bitmap Extension: File extension for bitmap files.

B.2, Example Point-noise Threshold Test Results

The tables shown below are examples of the data collected in the point-noise threshold test. The entire data set can be obtained by contacting:

Dr. Henry Potoczny
Air Force Institute of Technology
2950 P. Street
Wright-Patterson AFB, OH 45433-1111

Table 33, Example Hit Percentages – Non-Stego, All Thresholds

Test Case				
File	my (T=1000)	my (T=750)	my (T=500)	my (T=250)
01	0.03%	0.19%	0.35%	1.14%
02	0.00%	0.00%	0.32%	0.90%
03	0.00%	0.63%	0.87%	3.21%
04	0.00%	0.09%	1.05%	10.08%
05	0.00%	0.00%	0.80%	1.40%
06	0.00%	0.01%	0.26%	3.76%
07	0.00%	0.00%	0.02%	0.52%
08	0.00%	0.02%	0.18%	1.96%
09	0.00%	0.00%	0.19%	1.15%
10	0.00%	0.08%	1.06%	4.60%
11	0.00%	0.03%	0.71%	6.23%
12	0.00%	0.03%	0.86%	5.57%
13	0.00%	0.00%	0.29%	1.31%
14	0.16%	0.31%	0.82%	2.66%
15	0.06%	0.48%	1.75%	10.76%
16	0.00%	0.05%	0.97%	11.09%
17	0.10%	0.96%	6.09%	27.94%
18	0.07%	0.50%	2.49%	10.05%
19	0.11%	0.44%	1.95%	8.54%
20	0.00%	0.00%	0.51%	1.81%
21	0.01%	0.20%	1.72%	13.29%
22	0.02%	0.26%	2.44%	15.21%
23	0.02%	0.30%	1.72%	8.67%
24	0.01%	0.18%	1.58%	11.41%
25	0.01%	0.18%	1.64%	8.62%
26	0.07%	0.82%	5.29%	23.72%
27	0.00%	0.02%	0.42%	1.23%
28	0.02%	0.24%	2.21%	11.99%
29	0.16%	0.77%	4.16%	20.63%
30	0.08%	0.68%	3.98%	19.28%
Median	0.01%	0.18%	1.01%	7.39%

Table 34, Example Hit Percentages Results– HideSeek, T=1000

	Test Case					
File	**myhhb**	**myhhi**	**myhht**	**myhlb**	**myhli**	**myhlt**
01	0.58%	0.61%	0.60%	0.49%	0.47%	0.48%
02	6.13%	6.38%	6.50%	4.65%	4.99%	4.99%
03	0.71%	0.66%	0.66%	0.46%	0.42%	0.43%
04	3.77%	4.01%	4.07%	2.88%	2.91%	2.75%
05	2.15%	2.06%	2.14%	2.31%	2.31%	2.27%
06	0.13%	0.10%	0.10%	0.06%	0.06%	0.07%
07	0.81%	0.98%	1.03%	0.78%	0.79%	0.82%
08	1.81%	1.92%	1.98%	1.53%	1.56%	1.51%
09	0.36%	0.42%	0.46%	0.29%	0.32%	0.33%
10	0.56%	0.65%	0.61%	0.54%	0.56%	0.54%
11	1.79%	1.92%	1.86%	1.54%	1.63%	1.64%
12	1.40%	1.50%	1.53%	1.32%	1.40%	1.37%
13	0.34%	0.33%	0.34%	0.32%	0.32%	0.32%
14	3.01%	2.83%	3.05%	2.17%	2.16%	2.12%
15	5.75%	6.08%	6.51%	3.23%	3.27%	3.19%
16	2.37%	2.35%	2.34%	1.52%	1.51%	1.51%
17	5.33%	4.82%	5.24%	3.04%	3.24%	3.06%
18	1.99%	2.06%	2.18%	1.43%	1.47%	1.44%
19	3.71%	3.59%	3.70%	2.28%	2.35%	2.31%
20	0.98%	0.86%	0.94%	0.62%	0.60%	0.60%
21	5.00%	4.83%	5.20%	3.55%	3.70%	3.78%
22	2.33%	2.08%	2.25%	1.48%	1.50%	1.54%
23	5.28%	5.32%	5.60%	3.12%	3.27%	3.34%
24	2.52%	3.11%	3.07%	1.79%	1.98%	1.98%
25	4.92%	4.54%	4.88%	2.88%	3.03%	2.96%
26	4.15%	4.12%	4.36%	2.75%	2.77%	2.74%
27	4.62%	4.59%	4.75%	2.76%	2.90%	3.00%
28	6.00%	5.82%	6.40%	2.86%	3.04%	3.07%
29	4.96%	4.79%	5.07%	2.55%	2.59%	2.59%
30	5.80%	6.20%	6.51%	3.30%	3.65%	3.78%
Median	2.44%	2.59%	2.70%	1.98%	2.07%	2.05%

B.3, Point-noise Threshold Test Summary – Frequency Ordered Palette

B.3.1, Threshold = 1000

Table 35, Hit Percentages – Summary, T=1000

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	0.03%	0.02%	0.54%	0.68%
02	0.00%	0.00%	5.61%	6.67%
03	0.00%	0.03%	0.56%	0.67%
04	0.00%	0.01%	3.40%	4.35%
05	0.00%	0.01%	2.21%	2.25%
06	0.00%	0.01%	0.09%	0.16%
07	0.00%	0.09%	0.87%	1.16%
08	0.00%	0.00%	1.72%	1.98%
09	0.00%	0.01%	0.36%	0.51%
10	0.00%	0.00%	0.58%	0.67%
11	0.00%	0.01%	1.73%	1.93%
12	0.00%	0.00%	1.42%	1.57%
13	0.00%	0.00%	0.33%	0.34%
14	0.16%	0.22%	2.56%	3.04%
15	0.06%	0.06%	4.67%	7.71%
16	0.00%	0.00%	1.93%	2.47%
17	0.10%	0.09%	4.12%	5.33%
18	0.07%	0.10%	1.76%	2.18%
19	0.11%	0.06%	2.99%	3.84%
20	0.00%	0.00%	0.77%	0.91%
21	0.01%	0.03%	4.34%	5.64%
22	0.02%	0.02%	1.86%	2.32%
23	0.02%	0.03%	4.32%	5.78%
24	0.01%	0.01%	2.41%	3.30%
25	0.01%	0.02%	3.87%	5.31%
26	0.07%	0.09%	3.48%	4.79%
27	0.00%	0.02%	3.77%	5.01%
28	0.02%	0.02%	4.53%	7.46%
29	0.16%	0.16%	3.76%	5.29%
30	0.08%	0.10%	4.87%	7.37%
Median	0.01%	0.02%	2.31%	2.76%

B.3.2, Threshold = 750

Table 36, Hit Percentages – Summary, T=750

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	0.19%	0.08%	2.12%	2.74%
02	0.00%	0.00%	9.80%	13.43%
03	0.63%	0.57%	2.39%	3.03%
04	0.09%	0.13%	8.27%	12.37%
05	0.00%	0.03%	6.32%	8.96%
06	0.01%	0.02%	0.69%	1.08%
07	0.00%	0.17%	1.91%	2.89%
08	0.02%	0.03%	3.91%	5.43%
09	0.00%	0.04%	1.96%	2.55%
10	0.08%	0.10%	1.96%	2.62%
11	0.03%	0.09%	4.70%	6.31%
12	0.03%	0.03%	4.26%	6.07%
13	0.00%	0.00%	1.74%	2.25%
14	0.31%	0.36%	5.49%	7.49%
15	0.48%	0.52%	9.06%	15.75%
16	0.05%	0.08%	4.32%	6.17%
17	0.96%	0.93%	11.24%	15.51%
18	0.50%	0.43%	6.37%	8.52%
19	0.44%	0.39%	7.44%	10.21%
20	0.00%	0.00%	3.32%	4.39%
21	0.20%	0.25%	10.85%	16.07%
22	0.26%	0.26%	6.25%	8.34%
23	0.30%	0.36%	8.53%	12.47%
24	0.18%	0.18%	5.55%	7.85%
25	0.18%	0.19%	8.70%	12.46%
26	0.82%	0.86%	8.83%	12.62%
27	0.02%	0.04%	8.50%	12.17%
28	0.24%	0.29%	9.25%	15.57%
29	0.77%	0.82%	8.50%	12.12%
30	0.68%	0.74%	10.61%	16.53%
Median	0.18%	0.18%	6.29%	8.43%

B.3.3, Threshold = 500

Table 37, Hit Percentages – Summary, T=500

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	0.35%	0.37%	6.04%	8.69%
02	0.32%	0.32%	17.12%	24.77%
03	0.87%	0.98%	8.97%	10.32%
04	1.05%	1.43%	18.90%	28.05%
05	0.80%	0.81%	14.84%	23.23%
06	0.26%	0.30%	3.72%	5.72%
07	0.02%	0.25%	5.53%	7.53%
08	0.18%	0.22%	8.20%	12.14%
09	0.19%	0.26%	6.34%	9.52%
10	1.06%	1.10%	6.73%	8.82%
11	0.71%	0.81%	11.51%	17.02%
12	0.86%	0.97%	10.54%	16.74%
13	0.29%	0.26%	8.42%	11.87%
14	0.82%	1.04%	11.18%	16.27%
15	1.75%	1.79%	18.83%	30.83%
16	0.97%	1.21%	10.78%	15.77%
17	6.09%	6.22%	26.21%	34.84%
18	2.49%	2.36%	17.27%	23.19%
19	1.95%	1.90%	14.48%	20.37%
20	0.51%	0.54%	10.04%	14.20%
21	1.72%	1.90%	24.11%	35.98%
22	2.44%	2.38%	17.41%	23.82%
23	1.72%	1.82%	17.32%	25.37%
24	1.58%	1.55%	13.34%	18.97%
25	1.64%	1.82%	18.49%	26.88%
26	5.29%	5.41%	22.63%	31.46%
27	0.42%	0.47%	19.16%	28.58%
28	2.21%	2.64%	18.91%	30.75%
29	4.16%	4.32%	19.68%	27.05%
30	3.98%	4.15%	23.14%	33.69%
Median	1.01%	1.16%	14.66%	21.78%

B.3.4, Threshold = 250

Table 38, Hit Percentages – Summary, T=250

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	1.14%	1.58%	20.15%	29.00%
02	0.90%	1.19%	32.78%	46.78%
03	3.21%	3.09%	34.10%	51.08%
04	10.08%	12.68%	43.05%	56.64%
05	1.40%	1.50%	33.19%	49.35%
06	3.76%	4.17%	18.65%	26.56%
07	0.52%	1.05%	19.34%	28.27%
08	1.96%	2.47%	17.60%	24.25%
09	1.15%	1.31%	20.08%	30.57%
10	4.60%	5.66%	23.51%	30.41%
11	6.23%	7.16%	30.81%	42.42%
12	5.57%	6.68%	30.38%	44.26%
13	1.31%	1.82%	28.98%	44.32%
14	2.66%	3.81%	23.35%	32.23%
15	10.76%	11.51%	42.70%	58.52%
16	11.09%	13.19%	32.08%	41.33%
17	27.94%	29.43%	54.87%	63.90%
18	10.05%	10.89%	38.39%	47.96%
19	8.54%	8.72%	32.84%	42.14%
20	1.81%	2.38%	26.77%	39.64%
21	13.29%	14.04%	50.81%	65.93%
22	15.21%	16.07%	42.87%	54.07%
23	8.67%	9.61%	39.50%	52.77%
24	11.41%	12.32%	34.03%	44.22%
25	8.62%	9.87%	38.37%	51.74%
26	23.72%	24.62%	51.64%	63.59%
27	1.23%	1.32%	41.07%	58.87%
28	11.99%	12.44%	42.52%	58.24%
29	20.63%	21.85%	46.03%	56.80%
30	19.28%	20.35%	49.59%	61.27%
Median	7.39%	7.94%	33.61%	47.37%

B.4, Point-noise Threshold Test Summary – Luminance Ordered Palette

B.4.1, Threshold = 1000

Table 39, Hit Percentages – Summary, T=1000

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	0.03%	0.02%	0.03%	0.03%
02	0.00%	0.00%	0.00%	0.00%
03	0.00%	0.06%	0.00%	0.00%
04	0.00%	0.01%	0.00%	0.00%
05	0.00%	0.06%	0.00%	0.00%
06	0.18%	0.09%	0.18%	0.16%
07	0.21%	0.07%	0.21%	0.21%
08	0.00%	0.00%	0.00%	0.00%
09	0.00%	0.00%	0.00%	0.00%
10	0.00%	0.00%	0.00%	0.00%
11	0.00%	0.00%	0.00%	0.00%
12	0.00%	0.00%	0.00%	0.00%
13	0.00%	0.00%	0.00%	0.00%
14	0.33%	0.31%	0.33%	0.33%
15	0.06%	0.06%	0.06%	0.06%
16	0.00%	0.00%	0.00%	0.00%
17	0.10%	0.09%	0.10%	0.10%
18	0.07%	0.12%	0.07%	0.07%
19	0.11%	0.10%	0.11%	0.10%
20	0.00%	0.00%	0.00%	0.00%
21	0.20%	0.09%	0.20%	0.20%
22	0.02%	0.02%	0.02%	0.02%
23	0.02%	0.03%	0.02%	0.02%
24	0.01%	0.01%	0.01%	0.01%
25	0.02%	0.02%	0.01%	0.01%
26	0.07%	0.09%	0.07%	0.07%
27	0.12%	0.06%	0.12%	0.12%
28	0.02%	0.02%	0.02%	0.02%
29	0.17%	0.16%	0.17%	0.17%
30	0.16%	0.17%	0.15%	0.16%
Median	0.02%	0.03%	0.02%	0.02%

B.4.2, Threshold = 750

Table 40, Hit Percentages – Summary, T=750

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	0.19%	0.08%	0.19%	0.19%
02	0.00%	0.00%	0.00%	0.00%
03	0.63%	0.51%	0.63%	0.63%
04	0.09%	0.13%	0.09%	0.10%
05	0.30%	0.18%	0.31%	0.30%
06	0.29%	0.21%	0.29%	0.26%
07	0.30%	0.16%	0.30%	0.31%
08	0.02%	0.03%	0.02%	0.02%
09	0.28%	0.00%	0.28%	0.28%
10	0.08%	0.10%	0.09%	0.09%
11	0.03%	0.09%	0.03%	0.03%
12	0.03%	0.03%	0.03%	0.03%
13	0.00%	0.00%	0.00%	0.00%
14	0.67%	0.45%	0.69%	0.67%
15	0.48%	0.52%	0.48%	0.48%
16	0.05%	0.08%	0.05%	0.06%
17	0.96%	0.94%	0.95%	0.95%
18	0.77%	0.43%	0.77%	0.76%
19	0.62%	0.47%	0.62%	0.61%
20	0.00%	0.00%	0.00%	0.00%
21	0.40%	0.40%	0.41%	0.40%
22	0.26%	0.26%	0.26%	0.26%
23	0.30%	0.36%	0.30%	0.29%
24	0.18%	0.18%	0.18%	0.18%
25	0.46%	0.32%	0.45%	0.31%
26	0.82%	0.85%	0.82%	0.82%
27	0.21%	0.12%	0.20%	0.21%
28	0.24%	0.29%	0.26%	0.26%
29	0.80%	0.88%	0.77%	0.78%
30	0.86%	0.89%	0.86%	0.86%
Median	0.28%	0.20%	0.28%	0.27%

B.4.3, Threshold = 500

Table 41, Hit Percentages – Summary, T=500

	Test Case			
File	Non-Stego	S-Tools	HideSeek	Steganos
01	0.35%	0.37%	0.35%	0.35%
02	0.32%	0.32%	0.32%	0.32%
03	0.87%	0.96%	0.88%	0.88%
04	1.05%	1.43%	1.05%	1.05%
05	0.82%	0.84%	0.83%	0.82%
06	0.55%	0.54%	0.57%	0.58%
07	0.60%	0.29%	0.60%	0.60%
08	0.18%	0.22%	0.18%	0.18%
09	0.54%	0.46%	0.54%	0.54%
10	1.06%	1.10%	1.06%	1.07%
11	0.72%	0.83%	0.72%	0.72%
12	0.86%	0.96%	0.86%	0.86%
13	0.29%	0.26%	0.29%	0.29%
14	1.41%	1.19%	1.41%	1.42%
15	1.75%	1.79%	1.75%	1.75%
16	0.97%	1.21%	0.97%	0.97%
17	6.09%	6.21%	6.07%	6.06%
18	2.49%	2.34%	2.48%	2.48%
19	1.95%	1.85%	1.94%	1.94%
20	0.51%	0.54%	0.51%	0.51%
21	1.89%	2.01%	1.90%	1.90%
22	2.44%	2.38%	2.44%	2.44%
23	1.72%	1.82%	1.72%	1.72%
24	1.58%	1.54%	1.58%	1.57%
25	1.64%	1.80%	1.65%	1.65%
26	5.29%	5.38%	5.31%	5.32%
27	0.57%	0.52%	0.57%	0.57%
28	2.21%	2.64%	2.22%	2.22%
29	4.16%	4.68%	4.16%	4.16%
30	4.01%	4.12%	4.00%	4.00%
Median	1.06%	1.20%	1.05%	1.06%

B.4.4, Threshold = 250

Table 42, Hit Percentages – Summary, T=250

Test Case				
File	Non-Stego	S-Tools	HideSeek	Steganos
01	1.14%	1.57%	1.42%	1.53%
02	0.90%	1.19%	0.91%	0.92%
03	3.21%	3.17%	3.18%	3.17%
04	10.08%	12.65%	10.08%	10.08%
05	1.42%	1.56%	1.43%	1.42%
06	3.95%	4.31%	4.43%	4.75%
07	1.12%	1.19%	1.41%	1.62%
08	1.96%	2.47%	1.96%	1.97%
09	1.69%	1.91%	1.71%	1.71%
10	4.60%	5.65%	4.63%	4.65%
11	6.21%	7.28%	6.22%	6.22%
12	5.57%	6.63%	5.56%	5.58%
13	1.31%	1.82%	1.31%	1.31%
14	3.24%	3.95%	3.25%	3.25%
15	10.76%	11.52%	10.78%	10.80%
16	11.09%	13.19%	11.11%	11.13%
17	27.94%	29.37%	27.98%	28.00%
18	10.05%	10.89%	10.02%	10.01%
19	8.54%	8.55%	8.50%	8.49%
20	1.81%	2.38%	1.80%	1.80%
21	13.29%	14.02%	13.32%	13.35%
22	15.21%	16.06%	15.28%	15.27%
23	8.67%	9.60%	8.67%	8.68%
24	11.41%	12.25%	11.44%	11.44%
25	8.62%	9.70%	8.63%	8.62%
26	23.72%	24.59%	23.76%	23.75%
27	1.23%	1.35%	1.23%	1.24%
28	11.99%	12.42%	12.02%	12.00%
29	20.63%	22.73%	20.63%	20.63%
30	19.28%	20.13%	19.30%	19.31%
Median	7.37%	7.92%	7.36%	7.36%

B.5, Point-noise Threshold Test Max/Min Values

Table 43, Max-Min Hit Percentages (frequency-ordered)

Frequency Ordered Palette					
Tool		Threshold			
		1000	750	500	250
Non-Stego	Max	0.16%	0.96%	6.09%	27.94%
	Min	0.00%	0.00%	0.02%	0.52%
S-Tools	Max	0.30%	0.95%	6.25%	29.48%
	Min	0.00%	0.00%	0.17%	0.91%
HideSeek	Max	6.51%	15.10%	33.95%	63.60%
	Min	0.06%	0.48%	2.51%	12.05%
Steganos	Max	7.80%	16.59%	36.14%	66.07%
	Min	0.15%	1.04%	5.67%	24.16%

Table 44, Max-Min Hit Percentages (luminance-ordered)

Luminance Ordered Palette					
Tool		Threshold			
		1000	750	500	250
Non-Stego	Max	0.33%	0.96%	6.09%	27.94%
	Min	0.00%	0.00%	0.18%	0.90%
S-Tools	Max	0.36%	0.95%	6.25%	29.48%
	Min	0.00%	0.00%	0.21%	1.06%
HideSeek	Max	0.33%	0.96%	6.08%	28.00%
	Min	0.00%	0.00%	0.18%	0.90%
Steganos	Max	0.33%	0.96%	6.07%	28.02%
	Min	0.00%	0.00%	0.18%	0.92%

Appendix C, Byte Frequency Test Results

The data collected on each file consisted of a 256-element array for each file. Each test case consisted of 30 files, and there were 18 test cases in each of 2 test partitions. The shear volume of the data collected in the byte frequency analysis test makes it incompatible with printed media. The complete byte frequency data set used in this research is available upon request. (See Appendix B.2) The following sections present the cumulative median byte frequency test results.

C.1, Cumulative Median Byte Frequency

Table 45, 16-Bin Cumulative Median Byte Frequency (frequency-ordered)

	16 Bins			
Bin Range	**Non-Stego**	**S-Tools**	**HideSeek**	**Steganos**
0 - 15	36.13%	0.67%	36.27%	36.13%
16 - 31	14.72%	0.94%	14.73%	14.72%
32 - 47	10.62%	1.42%	10.62%	10.62%
48 - 63	8.23%	1.88%	8.19%	8.23%
64 - 79	6.56%	2.29%	6.56%	6.56%
80 - 95	5.30%	3.21%	5.30%	5.30%
96 - 111	4.33%	3.67%	4.33%	4.33%
112 - 127	3.60%	3.94%	3.61%	3.60%
128 - 143	3.00%	4.67%	3.00%	3.00%
144 - 159	2.44%	5.51%	2.44%	2.44%
160 - 175	1.96%	6.17%	1.96%	1.96%
176 - 191	1.55%	6.85%	1.55%	1.55%
192 - 207	1.14%	8.81%	1.14%	1.14%
208 - 223	0.73%	11.10%	0.73%	0.73%
224 - 239	0.35%	14.35%	0.35%	0.35%

Table 46, 8-Bin Cumulative Median Byte Frequency (frequency-ordered)

8 Bins				
Bin Range	**Non-Stego**	**S-Tools**	**HideSeek**	**Steganos**
0 - 31	50.85%	1.62%	50.99%	50.85%
32 - 63	18.86%	3.30%	18.81%	18.86%
64 - 95	11.86%	5.50%	11.86%	11.86%
96 - 127	7.93%	7.61%	7.93%	7.93%
128 - 159	5.43%	10.18%	5.43%	5.43%
160 - 191	3.51%	13.02%	3.51%	3.51%
192 - 223	1.87%	19.91%	1.87%	1.87%
224 - 255	0.44%	36.53%	0.44%	0.44%

Table 47, 4-Bin Cumulative Median Byte Frequency (frequency-ordered)

4 Bins				
Bin Range	**Non-Stego**	**S-Tools**	**HideSeek**	**Steganos**
0 - 63	69.71%	4.92%	69.81%	69.71%
64 - 127	19.79%	13.11%	19.79%	19.79%
128 - 191	8.95%	23.19%	8.94%	8.95%
192 - 255	2.30%	56.44%	2.30%	2.30%

Table 48, 16-Bin Cumulative Median Byte Frequency (luminance-ordered)

16 Bins				
Bin Range	Non-Stego	S-Tools	HideSeek	Steganos
0 - 15	8.37%	0.65%	8.55%	8.44%
16 - 31	6.06%	0.90%	6.06%	6.06%
32 - 47	5.64%	1.32%	5.64%	5.64%
48 - 63	5.35%	1.95%	5.35%	5.35%
64 - 79	5.46%	2.45%	5.46%	5.46%
80 - 95	5.90%	3.17%	5.90%	5.90%
96 - 111	5.10%	3.51%	5.10%	5.10%
112 - 127	5.33%	3.78%	5.33%	5.33%
128 - 143	5.65%	4.62%	5.65%	5.65%
144 - 159	5.47%	5.32%	5.47%	5.47%
160 - 175	5.37%	6.35%	5.37%	5.37%
176 - 191	5.03%	7.07%	5.03%	5.03%
192 - 207	5.34%	8.94%	5.34%	5.34%
208 - 223	5.86%	10.96%	5.86%	5.86%
224 - 239	6.55%	13.43%	6.55%	6.55%

Table 49, 8-Bin Cumulative Median Byte Frequency (luminance-ordered)

8 Bins				
Bin Range	Non-Stego	S-Tools	HideSeek	Steganos
0 - 31	14.43%	1.55%	14.61%	14.50%
32 - 63	10.99%	3.27%	10.99%	10.99%
64 - 95	11.36%	5.62%	11.36%	11.36%
96 - 127	10.43%	7.28%	10.43%	10.43%
128 - 159	11.12%	9.95%	11.12%	11.12%
160 - 191	10.40%	13.42%	10.40%	10.40%
192 - 223	11.20%	19.90%	11.20%	11.20%
224 - 255	20.05%	35.98%	20.03%	20.03%

Table 50, 4-Bin Cumulative Median Byte Frequency (luminance-ordered)

	4 Bins			
Bin Range	Non-Stego	S-Tools	HideSeek	Steganos
0 - 63	25.43%	4.82%	25.60%	25.50%
64 - 127	21.79%	12.90%	21.79%	21.79%
128 - 191	21.51%	23.37%	21.51%	21.51%
192 - 255	31.25%	55.87%	31.23%	31.23%

C.2, Cumulative Parameter-wise Median Byte Frequency

C.2.1, Message File Type

Table 51, 16-Bin Cumulative Median Byte Frequency (HideSeek, frequency-ordered)

	16 Bins		
Bin Range	Binary	Image	Text
0 - 15	36.02%	36.01%	35.96%
16 - 31	14.59%	14.59%	14.59%
32 - 47	10.52%	10.52%	10.52%
48 - 63	8.11%	8.12%	8.16%
64 - 79	6.50%	6.50%	6.50%
80 - 95	5.25%	5.25%	5.25%
96 - 111	4.29%	4.29%	4.29%
112 - 127	3.57%	3.57%	3.57%
128 - 143	2.97%	2.97%	2.97%
144 - 159	2.42%	2.42%	2.42%
160 - 175	1.94%	1.94%	1.95%
176 - 191	1.53%	1.53%	1.54%
192 - 207	1.13%	1.13%	1.13%
208 - 223	0.69%	0.68%	0.68%
224 - 239	0.35%	0.35%	0.35%
240 - 255	0.08%	0.08%	0.08%

Table 52, 16-Bin Cumulative Median Byte Frequency (Steganos, frequency-ordered)

	16 Bins		
Bin Range	Binary	Image	Text
0 - 15	35.80%	35.80%	35.80%
16 - 31	14.59%	14.59%	14.59%
32 - 47	10.52%	10.52%	10.52%
48 - 63	8.16%	8.16%	8.16%
64 - 79	6.50%	6.50%	6.50%
80 - 95	5.25%	5.25%	5.25%
96 - 111	4.29%	4.29%	4.29%
112 - 127	3.57%	3.57%	3.57%
128 - 143	2.97%	2.97%	2.97%
144 - 159	2.42%	2.42%	2.42%
160 - 175	1.95%	1.95%	1.95%
176 - 191	1.54%	1.54%	1.54%
192 - 207	1.13%	1.13%	1.13%
208 - 223	0.69%	0.69%	0.69%
224 - 239	0.35%	0.35%	0.35%
240 - 255	0.08%	0.08%	0.08%

Table 53, 16-Bin Cumulative Median Byte Frequency (S-Tools, frequency-ordered)

	16 Bins		
Bin Range	Binary	Image	Text
0 - 15	0.76%	0.78%	0.78%
16 - 31	1.12%	1.03%	1.02%
32 - 47	1.46%	1.45%	1.53%
48 - 63	2.16%	1.89%	2.19%
64 - 79	2.20%	2.53%	2.55%
80 - 95	3.30%	3.13%	3.31%
96 - 111	4.01%	3.20%	4.14%
112 - 127	3.95%	4.33%	3.82%
128 - 143	4.56%	5.05%	4.90%
144 - 159	5.30%	5.71%	5.58%
160 - 175	6.42%	6.14%	6.60%
176 - 191	7.38%	7.12%	7.28%
192 - 207	9.28%	8.82%	8.34%
208 - 223	10.22%	10.57%	10.56%
224 - 239	14.45%	14.17%	14.40%
240 - 255	22.65%	23.15%	22.28%

Table 54, 16-Bin Cumulative Median Byte Frequency (HideSeek, luminance-ordered)

	16 Bins		
Bin Range	Binary	Image	Text
0 - 15	8.53%	8.53%	8.53%
16 - 31	6.05%	6.05%	6.05%
32 - 47	5.64%	5.64%	5.64%
48 - 63	5.34%	5.34%	5.34%
64 - 79	5.45%	5.45%	5.45%
80 - 95	5.89%	5.89%	5.89%
96 - 111	5.09%	5.09%	5.09%
112 - 127	5.32%	5.32%	5.32%
128 - 143	5.64%	5.64%	5.64%
144 - 159	5.46%	5.46%	5.46%
160 - 175	5.36%	5.36%	5.36%
176 - 191	5.02%	5.02%	5.02%
192 - 207	5.34%	5.34%	5.34%
208 - 223	5.45%	5.34%	5.36%
224 - 239	6.54%	6.54%	6.54%
240 - 255	13.45%	13.45%	13.45%

Table 55, 16-Bin Cumulative Median Byte Frequency (Steganos, luminance-ordered)

	16 Bins		
Bin Range	Binary	Image	Text
0 - 15	8.43%	8.43%	8.43%
16 - 31	6.05%	6.05%	6.05%
32 - 47	5.64%	5.64%	5.64%
48 - 63	5.34%	5.34%	5.34%
64 - 79	5.45%	5.45%	5.45%
80 - 95	5.89%	5.89%	5.89%
96 - 111	5.09%	5.09%	5.09%
112 - 127	5.32%	5.32%	5.32%
128 - 143	5.64%	5.64%	5.64%
144 - 159	5.46%	5.46%	5.46%
160 - 175	5.36%	5.36%	5.36%
176 - 191	5.02%	5.02%	5.02%
192 - 207	5.34%	5.34%	5.34%
208 - 223	5.38%	5.38%	5.38%
224 - 239	6.54%	6.54%	6.54%
240 - 255	13.45%	13.45%	13.45%

Table 56, 16-Bin Cumulative Median Byte Frequency (S-Tools, luminance-ordered)

	16 Bins		
Bin Range	**Binary**	**Image**	**Text**
0 - 15	0.74%	0.66%	0.76%
16 - 31	1.09%	1.07%	1.03%
32 - 47	1.46%	1.39%	1.70%
48 - 63	2.10%	2.05%	2.11%
64 - 79	2.41%	2.63%	2.42%
80 - 95	3.27%	3.19%	3.37%
96 - 111	3.81%	3.38%	3.81%
112 - 127	3.84%	4.05%	3.94%
128 - 143	4.61%	4.86%	4.90%
144 - 159	5.44%	5.37%	5.56%
160 - 175	6.25%	6.54%	7.00%
176 - 191	7.39%	7.31%	6.79%
192 - 207	9.22%	9.21%	8.96%
208 - 223	10.42%	10.24%	10.87%
224 - 239	14.17%	14.22%	13.52%
240 - 255	22.94%	22.77%	22.51%

Table 57, 16-Bin Cumulative Median Byte Frequency (HideSeek, frequency-ordered)

	16 Bins	
Bin Range	**High**	**Low**
0 - 15	36.03%	35.96%
16 - 31	14.59%	14.59%
32 - 47	10.52%	10.52%
48 - 63	8.10%	8.16%
64 - 79	6.50%	6.50%
80 - 95	5.25%	5.25%
96 - 111	4.29%	4.29%
112 - 127	3.57%	3.57%
128 - 143	2.97%	2.97%
144 - 159	2.42%	2.42%
160 - 175	1.94%	1.95%
176 - 191	1.53%	1.54%
192 - 207	1.13%	1.13%
208 - 223	0.69%	0.68%
224 - 239	0.35%	0.35%
240 - 255	0.08%	0.08%

Table 58, 16-Bin Cumulative Median Byte Frequency (Steganos, frequency-ordered)

	16 Bins	
Bin Range	High	Low
0 - 15	35.80%	35.80%
16 - 31	14.59%	14.59%
32 - 47	10.52%	10.52%
48 - 63	8.16%	8.16%
64 - 79	6.50%	6.50%
80 - 95	5.25%	5.25%
96 - 111	4.29%	4.29%
112 - 127	3.57%	3.57%
128 - 143	2.97%	2.97%
144 - 159	2.42%	2.42%
160 - 175	1.95%	1.95%
176 - 191	1.54%	1.54%
192 - 207	1.13%	1.13%
208 - 223	0.69%	0.69%
224 - 239	0.35%	0.35%
240 - 255	0.08%	0.08%

Table 59, 16-Bin Cumulative Median Byte Frequency (S-Tools, frequency-ordered)

	16 Bins	
Bin Range	High	Low
0 - 15	1.14%	0.41%
16 - 31	1.69%	0.43%
32 - 47	2.23%	0.73%
48 - 63	3.44%	0.72%
64 - 79	3.74%	1.12%
80 - 95	3.60%	2.90%
96 - 111	4.54%	3.03%
112 - 127	4.82%	3.25%
128 - 143	6.54%	3.14%
144 - 159	6.20%	4.86%
160 - 175	6.42%	6.34%
176 - 191	7.86%	6.66%
192 - 207	8.54%	9.09%
208 - 223	9.67%	11.23%
224 - 239	10.95%	17.73%
240 - 255	18.07%	27.31%

Table 60, 16-Bin Cumulative Median Byte Frequency (HideSeek, luminance-ordered)

	16 Bins	
Bin Range	High	Low
0 - 15	8.53%	8.53%
16 - 31	6.05%	6.05%
32 - 47	5.64%	5.64%
48 - 63	5.34%	5.34%
64 - 79	5.45%	5.45%
80 - 95	5.89%	5.89%
96 - 111	5.09%	5.09%
112 - 127	5.32%	5.32%
128 - 143	5.64%	5.64%
144 - 159	5.46%	5.46%
160 - 175	5.36%	5.36%
176 - 191	5.02%	5.02%
192 - 207	5.34%	5.34%
208 - 223	5.45%	5.31%
224 - 239	6.54%	6.54%
240 - 255	13.45%	13.45%

Table 61, 16-Bin Cumulative Median Byte Frequency (Steganos, luminance-ordered)

	16 Bins	
Bin Range	High	Low
0 - 15	8.43%	8.43%
16 - 31	6.05%	6.05%
32 - 47	5.64%	5.64%
48 - 63	5.34%	5.34%
64 - 79	5.45%	5.45%
80 - 95	5.89%	5.89%
96 - 111	5.09%	5.09%
112 - 127	5.32%	5.32%
128 - 143	5.64%	5.64%
144 - 159	5.46%	5.46%
160 - 175	5.36%	5.36%
176 - 191	5.02%	5.02%
192 - 207	5.34%	5.34%
208 - 223	5.38%	5.38%
224 - 239	6.54%	6.54%
240 - 255	13.45%	13.45%

Table 62, 16-Bin Cumulative Median Byte Frequency (S-Tools, luminance-ordered)

	16 Bins	
Bin Range	**High**	**Low**
0 - 15	1.03%	0.41%
16 - 31	1.70%	0.42%
32 - 47	2.43%	0.60%
48 - 63	3.34%	0.83%
64 - 79	3.88%	1.10%
80 - 95	3.60%	2.95%
96 - 111	4.40%	2.93%
112 - 127	4.78%	3.10%
128 - 143	6.19%	3.39%
144 - 159	6.27%	4.64%
160 - 175	6.64%	6.55%
176 - 191	7.74%	6.58%
192 - 207	9.22%	9.04%
208 - 223	9.53%	11.49%
224 - 239	10.61%	17.33%
240 - 255	17.75%	27.72%

Appendix D, Miscellaneous C-code

The source code for the following C/C++ routines can be obtained by contacting:

Dr. Henry Potoczny
Air Force Institute of Technology
2950 P. Street
Wright-Patterson AFB, OH 45433-1111

D.1, RGB to Grayscale Conversion (rgb2gray.c)

D.2, Frequency-ordered Palette Conversion (palfreq.c)

D.3, Luminance-ordered Palette Conversion (pallum.c)

D.4, Bitmap I/O Header File (bmpio.h)

Appendix E, Filtering and Thresholding

This appendix provides additional information on calculation of the response of the point-noise filter and the subsequent creation of a binary image using thresholding. The image shown in Figure 61will be used to demonstrate each step. Note that the light pixels have a value of 127, and the dark pixels have a value of 255.

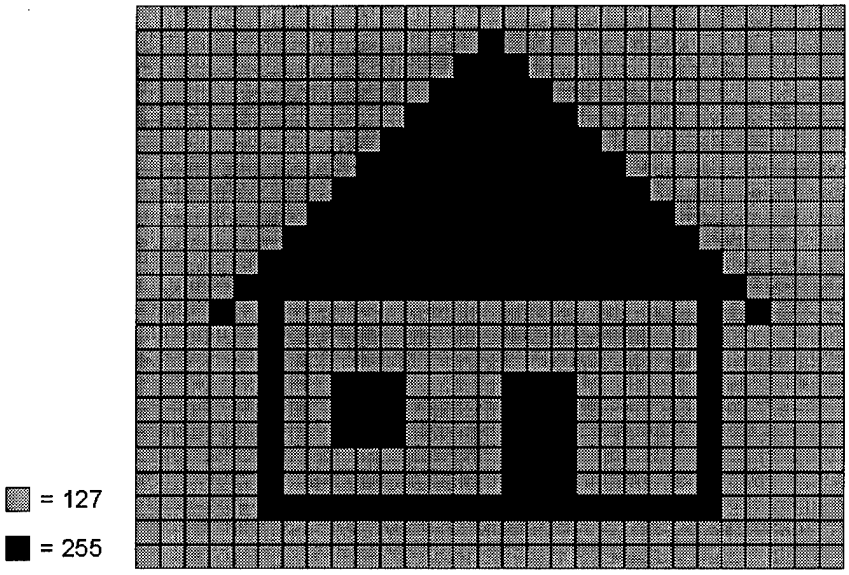


Figure 61, Initial Image

For the purposes of this demonstration, we will only calculate the response of the image area that represents the window of the house as shown below in Figure 62. The 3x3 matrix of coefficients is the point-noise filter used in this research, and it is applied to the image data in a row-major fashion. The center coefficient is applied to the target pixel, while the surrounding coefficients are applied to its neighbors.

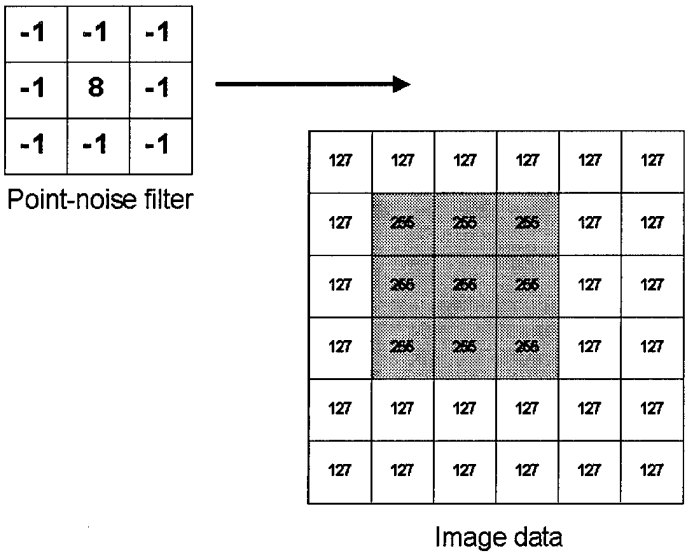


Figure 62, Original Image Data and Filter

The first step is to calculate the response of the filter on the image. The response, R, is calculated by the equation

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i$$

,where

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

and z_i is the intensity of the pixel associated with the coefficient w_i. Obviously, the edges of the image are an area of concern since the filter overlaps the image in these

areas. There are several methods of handling the boundary pixels. The pixels at the boundaries can be ignored so that the first row of calculations is row two, and the first column is column two. Such a method is easy to implement and works well with large images where the number of boundary pixels is small compared to the total pixels in the image.

Another method would be to calculate a partial response along the edges. This involves reducing the center coefficient such that it equals one less than the number of pixels covered by the filter.

Still another method involves padding the image with additional rows and columns. While several methods exist to determine the value of the new boundary pixels, the easiest is to duplicate the existing boundary rows and columns. This method is shown in Figure 63 and is used for the remainder of this example. The new boundary pixel values reflect the old boundary.

The response of the filter on the first (top left) pixel in the original image is given by:

$$\begin{aligned} R = {} & (-1)(127) + (-1)(127) + (-1)(127) + \\ & (-1)(127) + (8)(127) + (-1)(127) + \\ & (-1)(127) + (-1)(127) + (-1)(255) + \\ = {} & -128 \end{aligned}$$

The results of the remaining calculations are shown below in Figure 64. Note how the response of the filter goes to zero when the target pixel is identical to all of its neighboring pixels.

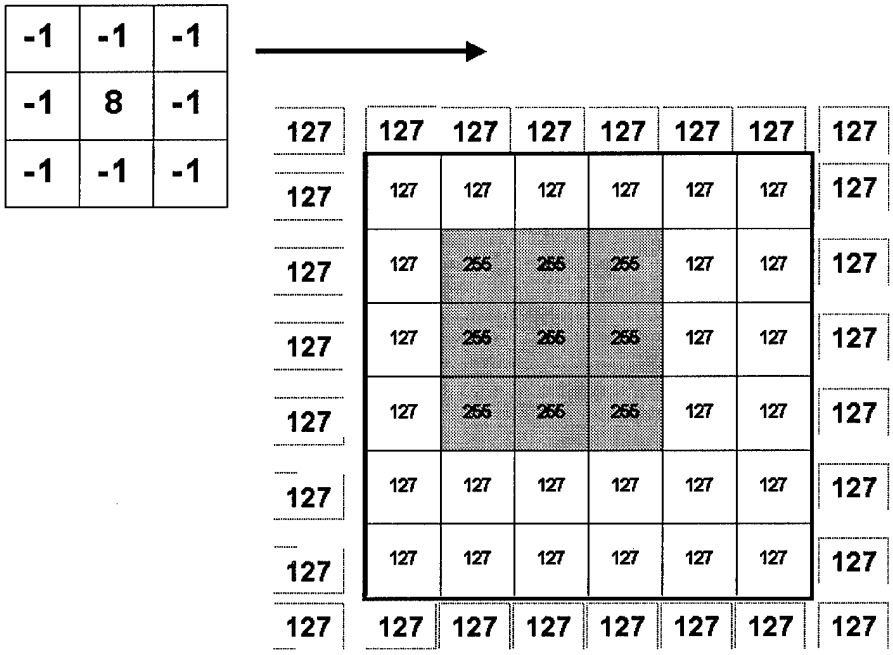


Figure 63, Padded Image

The remaining step in the process is to apply a threshold to the new calculated image values. In order to construct a binary image, the filtered image is converted by applying the following rule, where R is the response of the filter and T is the threshold:

If $|R| > T$, then pixel value = 255 (black),

else pixel value = 0 (white).

The three images in Figure 65 show the affects of three arbitrary thresholds: 250, 300, and 400. Notice how lower thresholds correspond to greater sensitivity.

-128	-256	-384	-256	-128	0
-256	640	384	640	-256	0
-384	384	0	384	-384	0
-256	640	384	640	-256	0
-128	-256	-384	-256	-128	0
0	0	0	0	0	0

Figure 64, Filtered Image Values

	|R| > 250				
0	1	1	1	0	0
1	1	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	0
0	1	1	1	0	0
0	0	0	0	0	0
		|R|>300			
0	0	1	0	0	0
0	1	1	1	0	0
1	1	0	1	1	0
0	1	1	1	0	0
0	0	1	0	0	0
0	0	0	0	0	0
		|R| >400			
0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	1	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

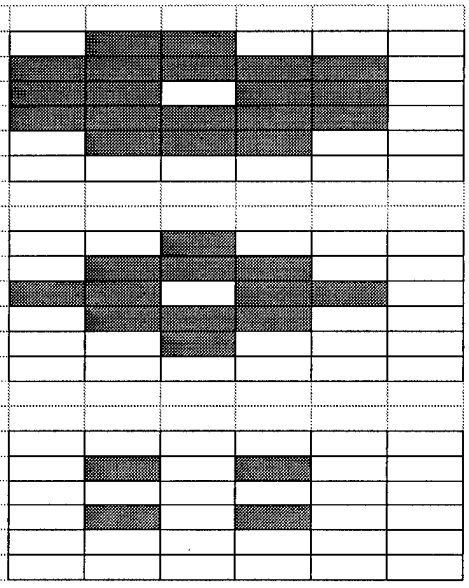


Figure 65, Threshold Binary Images

Bibliography

[1] Anderson, Ross, and Fabien Petitcolas. On the Limits of Steganography. University of Cambridge, Computer Laboratory: Cambridge, UK. September 1997. Published in *IEEE Journal on Special Areas in Communications v 16 no 4* (May 98) pp 463—473. Available at URL: http://www.cl.cam.ac.uk/~fapp2/papers/jsac98-limsteg/

[2] Aura, Tuomas. *Invisible Communication*, Proceedings of Information Hiding Workshop 1996. Helsinki Univ. of Technology, Finland, November 1995. Available at URL: ftp://saturn.hut.fi/pub/aaura/aura-ihws96.ps

[3] Brassil, J., S. Low, N.F. Maxemchuck, L. O'Gorman. *Hiding Information in Document Images*. AT&T Bell Laboratories, Murray Hill, NJ. Available at URL: ftp://ftp.research.att.com/dist/brassil/1994/infocom94a.ps.Z

[4] -- Electronic Marking and Identification Techniques to Discourage Document Copying. Published in IEEE Infocom '94. AT&T Bell Laboratories, Murray Hill, NJ. Available at URL: ftp://ftp.research.att.com/dist/brassil/infocom94.ps

[5] -- Document Marking and Identification Using Both Line and Word Shifting. Published in IEEE Infocom '95, Boston, MA. April 1995. Available at URL: www.epm.ornl.gov/~dunigan/stega.html and ftp://ftp.research.att.com/dist/brassil/1995/ciss95.ps.Z

[6] Gonzalez, Rafael C. and Richard E. Woods. Digital Image Processing. Addison-Wesley: Reading, MA. 1993.

[7] Johnson, Neil F. and Sushil Jajodia, *Exploring Steganography: Seeing the unseen*, IEEE Computer, Vol. 31, No. 2, February 1998, pages 26--34.

[8] -- *Steganalysis of Images Created Using Current Steganography Software*, Preproceedings of the Second Information Hiding Workshop, Portland OR, 1998.

[9] Kahn, David. The Codebreakers: The Story of Secret Writing. Macmillan Company: NY. 1967.

[10] Kay, David C. and John R. Levine. Graphic File Formats. TAB Books, Blue Ridge Summit, PA. 1992.

[11] Murray, James D. and William vanRyper. Encyclopedia of Graphic File Formats. O'Reilly & Associates, Inc.: Sebastopol, CA. 1994.

[12] Nelson, Mark and Jean-Loup Gailly. The Data Compression Book 2nd Ed. M&T Books, New York, NY. 1996.

[13] Wayner, Peter. *Mimic Functions*. Department of Computer Science, Cornell University: Ithaca, NY. Published in *Cryptologia v XVI no 3,* pp 193 – 214, July 1992.

Vita

Captain Christopher J. Fogle was born on 17 May 1964 in Dover, Delaware. He graduated from Radford High School, Honolulu, Hawaii, in 1982 and entered the United States Coast Guard Academy in New London, Connecticut. He enlisted in the United States Air Force in July 1985 and continued his undergraduate studies with the University of Maryland, European Division. He graduated *Magna Cum Laude* with a Bachelor of Science degree in Computer Science in May 1992. He was commissioned on 21 January 1994 upon graduation from Officer Training School.

His assignments include duty as a military working dog handler at Carswell AFB, Texas; Kwang Ju AB, Republic of Korea; Bitburg AB, Germany; and Laughlin AFB, Texas. As a commissioned officer, he was assigned to Howard AFB, Panama, as the Chief, Base Network Control Center, and Support Flight Commander. In August 1997, he entered the School of Engineering, Air Force Institute of Technology. His follow-on assignment is to the Air Force Information Warfare Center, Kelly AFB, Texas.

Permanent Address: 1700 N. Ainger Rd.
Charlotte, MI 48813

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1999	Master's Thesis

4. TITLE AND SUBTITLE

STRATEGIES FOR STEGANALYSIS OF BITMAP GRAPHICS FILES

5. FUNDING NUMBERS

6. AUTHOR(S)

Christopher J. Fogle, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology (AFIT)
2950 P Street
Wright-Patterson AFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/99M-05

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Henry B. Potoczny, PhD
Professor of Computer Science and Computer Engineering
Air Force Institute of Technology (AFIT)
2950 P Street
Wright-Patterson AFB, OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Advisor: Dr. Henry B. Potoczny, PhD
(937) 255-6565 ext 4282 (DSN 785-6565 ext 4282)
potoczny@afit.af.mil

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Steganography uses covert channels to hide the presence of communications, and the use of graphics image files to conceal criminal communications is increasing. Steganalysis includes methods to detect the presence of embedded information; however, manually investigating thousands of files is painstaking work. This thesis explores steganalysis strategies that reduce the search space an investigator must confront. Emphasis is placed on methods that are easily incorporated in an automated detection tool.

This thesis develops two strategies that reliably detect embedded information when original file characteristics are unknown - a problem known as blind steganalysis. One strategy uses a point-noise detection filter to determine if pixels in an image are significantly different from its neighbors. The percentage of "different" pixels is used to select images that possibly contain embedded information. The second strategy examines a file's byte-frequency distribution. Files that have byte-value distributions outside the norm are selected as suspect images.

The results indicate these strategies provide reliable detection mechanisms for 75% of the search space. The point-noise threshold test achieves 98% success rates in cases where the embedding process results in a visibly distorted image. The byte-frequency analysis test achieves a 100% success rate for a subset of images containing no distortions.

14. SUBJECT TERMS	15. NUMBER OF PAGES
steganography; steganalysis; covert channels; cryptography; secure communications; information security	179
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL