

Stenography

Chapter 1

Introduction

Steganography is the art and science of hiding communication; a steganographic system thus embeds hidden content in unremarkable cover media so as not to arouse an eavesdropper's suspicion. In the past, people used hidden tattoos or invisible ink to convey steganographic content. Today, computer and network technologies provide easy to-use communication channels for steganography.

Essentially, the information-hiding process in a steganographic system starts by identifying a cover medium's redundant bits (those that can be modified without destroying that medium's integrity).¹ The embedding process creates a *stego medium* by replacing these redundant bits with data from the hidden message.

Modern steganography's goal is to keep its mere presence undetectable, but steganographic systems—because of their invasive nature—leave behind detectable traces in the cover medium. Even if secret content is not revealed, the existence of it is: modifying the cover medium changes its statistical properties, so eavesdroppers can detect the distortions in the resulting stego medium's statistical properties. The process of finding these distortions is called *statistical steganalysis*.

This article discusses existing steganographic systems and presents recent research in detecting them via statistical steganalysis. Other surveys focus on the general usage of information hiding and watermarking or else provide an overview of detection algorithms. Here, we present recent research and discuss the practical application of detection algorithms and the mechanisms for getting around them.

1.1 The basics of embedding:-

Three different aspects in information-hiding systems contend with each other: capacity, security, and robustness. 4. Capacity refers to the amount of information that can be hidden in the cover medium, security to an eavesdropper's inability to detect hidden information, and robustness to the amount of modification the stego medium can withstand before an adversary can destroy hidden information.

Information hiding generally relates to both watermarking and steganography. A watermarking system's primary goal is to achieve a high level of robustness—that is, it should be impossible to remove a watermark without degrading the data object's quality. Steganography, on the other hand, strives for high security and capacity, which often entails that the hidden information is fragile. Even trivial modifications to the stego Medium can destroy it.

A classical steganographic system's security relies on the encoding system's secrecy. An example of this type of system is a Roman general who shaved a slave's head and tattooed a message on it. After the hair grew back, the slave was sent to deliver the now-hidden message.⁵ although such a system might work for a time, once it is known, it is simple enough to shave the heads of all the people passing by to check for hidden messages—ultimately, such a steganographic system fails.

Modern steganography attempts to be detectable only if secret information is known—namely, a secret key. This is similar to Kerckhoffs' Principle in cryptography, which holds that a cryptographic system's security should rely solely on the key material.⁶ for steganography to remain undetected; the unmodified cover medium must be kept secret, because if it is exposed, a comparison between the cover and stego media immediately reveals the changes.

Information theory allows us to be even more specific on what it means for a system to be perfectly secure. Christian Cachin proposed an information-theoretic model for Steganography that considers the security of steganographic systems against passive eavesdroppers. In this model, you assume that the adversary has complete knowledge of the encoding system but does not know the secret key. His or her task is to devise a model for the probability distribution PC of all possible cover media and PS of all possible stego media. The adversary can then use *detection theory* to decide between hypothesis C (that a message contains no hidden information) and hypothesis S (that a message carries hidden content). A system is perfectly secure if no decision rule exists that can perform better than random guessing.

Essentially, steganographic communication senders and receivers agree on a steganographic system and a shared secret key that determines how a message is encoded in the cover medium. To send a hidden message, for example, Alice creates a new image with a digital camera. Alice supplies the steganographic system with her shared secret and her message. The steganographic system uses the shared secret to determine how the hidden message should be encoded in the redundant bits. The result is a stego image that Alice sends to Bob. When Bob receives the image, he uses the shared secret and the agreed on steganographic system to retrieve the hidden message. Figure 1 shows an overview of the encoding step; as mentioned earlier, statistical analysis can reveal the presence of hidden content.

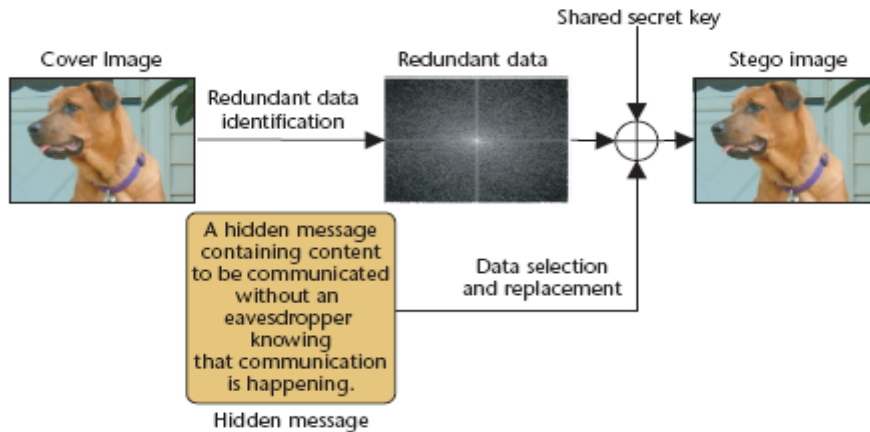


Figure 1. Modern steganographic communication. The encoding step of a steganographic system identifies redundant bits and then replaces a subset of them with data from a secret message.

1.2 What is Steganography?

While we are discussing it in terms of computer security, steganography is really nothing new, as it has been around since the times of ancient Rome. For example, in ancient Rome and Greece, text was traditionally written on wax that was poured on top of stone tablets. If the sender of the information wanted to obscure the message - for purposes of military intelligence, for instance - they would use steganography: the wax would be scraped off and the message would be inscribed or written directly on the tablet, wax would then be poured on top of the message, thereby obscuring not just its meaning but its very existence.

According to *Dictionary.com*, steganography (also known as "steg" or "stego") is "the art of writing in cipher, or in characters, which are not intelligible except to persons who have the key; cryptography. In computer terms, steganography has evolved into the practice of hiding a message within a larger one in such a way that others cannot discern the presence or contents of the hidden message. In contemporary terms, steganography has evolved into a digital strategy of hiding a file in some form of multimedia, such as an image, an audio file (like a .wav or mp3) or even a video file.

Hide and seek:-

Although steganography is applicable to all data objects that contain redundancy, in this article, we consider JPEG images only (although the techniques and methods for steganography and steganalysis that we present here apply to other data formats as well). People often transmit digital pictures over email and other Internet communication, and JPEG is one of the most common formats for images. Moreover, steganographic systems for the JPEG format seems more interesting because the systems operate in a transform space and are not affected by visual attacks. (Visual attacks mean that you can see Steganographic messages on the low bit planes of an image because they overwrite visual structures; this usually happens in BMP images.) Neil F. Johnson and Sushil Jajodia, for

example, showed that steganographic systems for palette-based images leave easily detected distortions.

Let's look at some representative steganographic systems and see how their encoding algorithms change an image in a detectable way. We'll compare the different systems and contrast their relative effectiveness.

Chapter 2

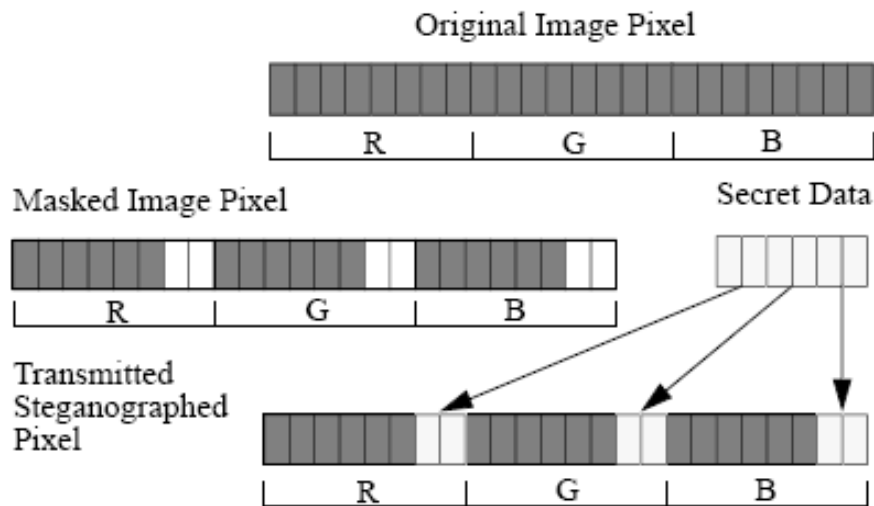
Techniques for to Hidden the data:-

Two techniques are available to those wishing to transmit secrets using unprotected communications media. One is cryptography, where the secret is scrambled and can be reconstituted only by the holder of a key. When cryptography is used, the fact that the secret was transmitted is observable by anyone. The second method is steganography.

Here the secret is encoded in another message in a manner such that, to the casual observer, it is unseen. Thus, the fact that the secret is being transmitted is also a secret. Widespread use of digitized information in automated information systems has resulted in a renaissance for steganography. Information which provides the ideal vehicle for steganography is that which is stored with accuracy far greater than necessary for the data's use and display. Image, Postscript, and audio files are among those that fall into this category, while text, database, and executable code files do not.

It has been demonstrated that a significant amount of information can be concealed in bitmapped image files with little or no visible degradation of the image. This process, called steganography, is accomplished by replacing the least significant bits in the pixel bytes with the data to be hidden. Since the least significant pixel bits contribute very little to the overall appearance of the pixel, replacing these bits often has no perceptible effect on the image.

To illustrate, consider a 24 bit pixel which uses 8 bits for each of the red, green, and blue color channels. The pixel is capable of representing 2^{24} or 16,777,216 color values. If we use the lower 2 bits of each color channel to hide data (Figure), the maximum change in any pixel would be 2^6 or 64 color values; a minute fraction of the whole color space. This small change is invisible to the human eye. To continue the example, an image of 735 by 485 pixels could hold $735 \times 485 \times 6 \text{ bits/pixel} \times 1 \text{ byte}/8 \text{ bits} = 267,356$ bytes of data.



Kurak and McHugh [4.] show that it is even possible to embed one image inside another. Further, they assert that visual inspection of an image prior to its being downgraded is insufficient to prevent unauthorized flow of data from one security level to a lower one.

A number of different formats are widely used to store imagery including BMP, TIFF, GIF, etc. Several of these image file formats “palletize” images by taking advantage of the fact that the color veracity of the image is not significantly degraded to the human observer by drastically reducing the total number of colors available. Instead of over 16 million possible colors, the color range is reduced and stored in a table. Each pixel, instead of containing a precise 24-bit color, stores an 8-bit index into the color table. This reduces the size of the bitmap by 2/3.

When the image is processed for display by a viewer such as “xv”, the indices stored at the location of each pixel are used to obtain the colors to be displayed from the color table. It has been demonstrated that steganography is ineffective when images are stored using this compression algorithm. Difficulty in designing a general-purpose steganographic algorithm for palletized images results from the following factors:

A change to a “pixel” results in a different index into the color table, which could result in a dramatically different color, changes in the color table can result in easily perceived changes to the image, and color maps vary from image to image with compression choices made as much for aesthetic reasons as for the efficiency of the compression.

Despite the relative ease of employing steganography to covertly transport data in an uncompressed 24-bit image, lossy compression algorithms based on techniques from digital signal processing, which are very commonly employed in image handling

systems, pose a severe threat to the embedded data. An excellent example of this is the ubiquitous Joint Photographic Experts Group (JPEG) compression algorithm which is the principle compression technique for transmission and storage of images used by government organizations. It does a quite thorough job of destroying data hidden in the least significant bits of pixels. The effects of JPEG on image pixels and coding techniques to counter its corruption of steganographically hidden data are the subjects of this paper.

2.1 JPEG Compression

JPEG has been developed to provide efficient, flexible compression tools. JPEG has four modes of operation designed to support a variety of continuous-tone image applications. Most applications utilize the Baseline sequential coder/decoder which is very effective and is sufficient for many applications.

JPEG works in several steps. First the image pixels are transformed into a luminance/chrominance color space [6.] and then the chrominance component is down sampled to reduce the volume of data. This down sampling is possible because the human eye is much more sensitive to luminance changes than to chrominance changes. Next, the pixel values are grouped into 8x8 blocks which are transformed using the discrete cosine transform (DCT). The DCT yields an 8x8 frequency map which contains coefficients representing the average value in the block and successively higher-frequency changes within the block.

Each block then has its values divided by a quantization coefficient and the result rounded to an integer. This quantization is where most of the loss caused by JPEG occurs. Many of the coefficients representing higher frequencies are reduced to zero. This is acceptable since the higher frequency data that is lost will produce very little visually detectable change in the image. The reduced coefficients are then encoded using Huffman coding to further reduce the size of the data. This step is lossless. The final step in JPEG applications is to add header data giving parameters to be used by the decoder.

2.2 Stego Encoding Experiments

As mentioned before, embedding data in the least significant bits of image pixels is a simple steganographic technique, but it cannot survive the deleterious effects of JPEG. To investigate the possibility of employing some kind of encoding to ensure survivability of embedded data it is necessary to identify what kind of loss/corruption JPEG causes in an image and where in the image it occurs.

At first glance, the solution may seem to be to look at the compression algorithm to try to predict mathematically where changes to the original pixels will occur. This is impractical since the DCT converts the pixel values to coefficient values representing 64 basis signal amplitudes. This has the effect of spatially “smearing” the pixel bits so that the location of any particular bit is spread over all the coefficient values. Because of the complex relationship between the original pixel values and the output of the DCT, it is not feasible to trace the bits through the compression algorithm and predict their location in the compressed data.

Due to the complexity of the JPEG algorithm an empirical approach to studying its effects is called for. To study the effects of JPEG, 24 bit Windows BMP format files were compressed, decompressed, with the resulting file saved under a new filename.



The BMP file format was chosen for its simplicity and widespread acceptance for image processing applications. For the experiments, two photographs, one of a seagull and one of a pair of glasses (Figure 2 and Figure 3), were chosen for their differing amount of detail and number of colors. JPEG is sensitive to these factors. Table 1 below shows the results of a byte by byte comparison of the original image files and the JPEG processed versions, normalized to 100,000 bytes for each image. Here we see that the seagull picture has fewer than half as many errors in the most significant bits (MSB) as the glasses picture. While the least significant bits (LSB) have an essentially equivalent number of errors.



	MSB 8	7	6	5	4	3	2	LSB 1
Glasses	744	4032	10247	21273	33644	42327	27196	48554
Seagull	257	991	2821	7514	15039	29941	41593	46640

Table 1:

Table 2 shows the Hamming distance (number of differing bits) between corresponding pixels in the original and JPEG processed files normalized to 100,000 pixels for each image. Again, the seagull picture has fewer errors.

	1-3	4-6	7-9	10-12	13-15	16-18	19-21	22-24
Glasses	15581	37135	30337	11976	2205	172	4	0
Seagull	24188	38710	17564	4631	409	43	1	0

Table 2:

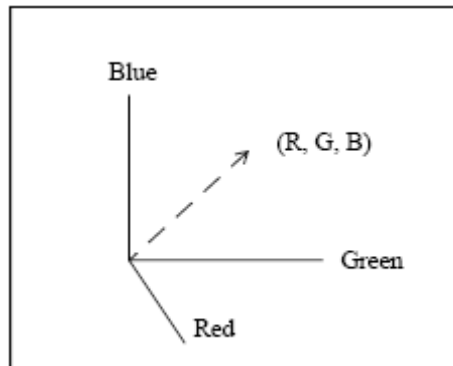
Given the information in Table 1, it is apparent that data embedded in any or all of the lower 5 bits would be corrupted beyond recognition. Attempts to embed data in these bits and recover it after JPEG processing showed that the recovered data was completely garbled by JPEG.

Since a straightforward substitution of pixel bits with data bits proved useless, a simple coding scheme to embed one data bit per pixel byte was tried. A bit was embedded in the lower 5 bits of each byte by replacing the bits with 01000 to code a 0 and 11000 to code a 1. On decoding, any value from 00000 to 01111 would be decoded as a 0 and 10000 to 11111 as a 1. The hypothesis was that perhaps JPEG would not change a byte value by more than 7 in an upward direction and 8 in a downward direction or, if it did, it would make drastic changes only occasionally and some kind of redundancy coding could be used to correct errors. This approach failed. JPEG is indiscriminate about the amount of change it makes to byte values and produced enough errors that the hidden data was unrecognizable.

The negative results of the first few attempts to embed data indicated that a more subtle approach to encoding was necessary. It was noticed that, in a JPEG processed image, the pixels which were changed from their original appearance were similar in color to the original. This indicates that the changes made by JPEG, to some extent, maintain the general color of the pixels. To attempt to take advantage of this, a new coding scheme was devised based on viewing the pixel as a point in space (Figure 4) with the three color channel values as the coordinates.

The coding scheme begins by computing the distance from the pixel to the origin (0,0,0). Then the distance is divided by a number and the remainder ($r = \text{distance} \bmod n$) is found. The pixel value is adjusted such that its remainder is changed to a number corresponding to the bit value being encoded. Qualitatively, this means that the length of the vector representing the pixel's position in three-dimensional RGB color space is modified to encode information. Because the vector's direction is unmodified, the relative sizes of the color channel values are preserved.

Suppose we choose an arbitrary modulus of 42. When the bit is decoded, the distance to origin will be computed and any value from 21 to 41 will be decoded as a 1 and any value from 0 to 20 will be decoded as a 0. So we want to move the pixel to a middle value in one of these ranges to allow for error introduced by JPEG. In this case, the vector representing the pixel would have its length modified so that the remainder is 10 to code a 0 or a 31 to code a 1. It was hoped that JPEG would not change the pixel's distance from the origin by more than 10 in either direction thus allowing the hidden information to be correctly decoded.



For example, given a pixel (128, 65, 210) the distance to the origin would be computed: $d = \sqrt{(128^2 + 65^2 + 210^2)} = 254.28$. The value of d is rounded to the nearest integer. Next we find, which is 2. If we are coding a 0 in this pixel, the amplitude of the color vector will be increased by 8 units to an ideal remainder of 10 ($d = 262$) and moved down 13 ($d = 241$) units to code a 1. Note that the maximum displacement any pixel would suffer would be 21. Simple vector arithmetic permits the modified values of the red, green, and blue components to be computed. The results of using this encoding are described in the next section.

Another similar technique is to apply coding to the luminance value of each pixel in the same way as was done to the distance from origin. The luminance, y , of a pixel is computed as $y = 0.3R + 0.6G + 0.1B$ [6.]. Where R, G, and B are the red, green, and blue color values respectively. This technique appears to hold some promise since the number of large changes in the luminance values caused by JPEG is not a high as with the distance from origin. One drawback of this technique is that the range of luminance value is from 0 to 255 whereas the range of the distance from origin is 0 to 441.67.

Chapter 3

Steganography detection on the Internet

How can we use these steganalytic methods in a real world setting—for example, to assess claims that steganographic content is regularly posted to the Internet? To find out if such claims are true, we created a Steganography detection framework that gets JPEG images off the Internet and uses steganalysis to identify subsets of the images likely to contain steganographic content.

3.1 Steganographic systems in use:-

To test our framework on the Internet, we started by searching the Web and Usenet for three popular steganographic systems that can hide information in JPEG images: JSteg (and JSteg-Shell), JPHide, and OutGuess. All these systems use some form of least-significant bit embedding and are detectable with statistical analysis. JSteg-Shell is a Windows user interface to JSteg first developed by John Korejwa. It supports content encryption and compression before JSteg embeds the data.

JSteg-Shell uses the RC4 stream cipher for encryption (but the RC4 key space is restricted to 40 bits). JPHide is a steganographic system Allan Latham first developed that uses Blowfish as a PRNG.^{24,25} Version 0.5 (there's also a version 0.3) supports additional compression of the hidden message, so it uses slightly different headers to store embedding information. Before the content is embedded, the content is Blowfish encrypted with a user-supplied pass phrase.

3.2 Finding images

To exercise our ability to test for steganographic content automatically, we needed images that might contain hidden messages. We picked images from eBay auctions (due to various news reports) and discussion groups in the Usenet archive for analysis. To get images from eBay auctions, a Web crawler that could find JPEG images was the obvious choice. Unfortunately, there were no open-source, image-capable Web crawlers available when we started our research. To get around this problem, we developed Crawl, a simple, efficient Web crawler that makes a local copy of any JPEG images it encounters on a Web page. Crawl performs a depth-first search and has two key features:

- Images and Web pages can be matched against regular expressions; a match can be used to include or exclude Web pages in the search.
- Minimum and maximum image size can be specified, which lets us exclude images that are too small to contain hidden messages. We restricted our search to images larger than 20 Kbytes but smaller than 400.

We downloaded more than two million images linked to eBay auctions. To automate detection, Crawl uses *stdout* to report successfully retrieved images to Stegdetect. After processing the two million images with Stegdetect, we found that over 1 percent of all images seemed to contain hidden content. JPHide was detected most often. We augmented our study by analyzing an additional one million images from a Usenet archive. Most of these are likely to be false-positives. Stefan Axelsson applied the *base-rate fallacy* to intrusion detection systems and showed that a high percentage of false positives had a significant effect on such a system's efficiency.²⁷ The situation is very similar for Stegdetect.

We can calculate the true-positive rate—the probability that an image detected by Stegdetect really has steganographic content—as follows:-

$$P(S|D) = \frac{P(S) \cdot P(D|S)}{P(D)}$$

$$= \frac{P(S) \cdot P(D|S)}{P(S) \cdot P(D|S) + P(\neg S) \cdot P(D|\neg S)}$$

where $P(S)$ is the probability of steganographic content in images, and $P(\neg S)$ is its complement. $P(D|S)$ is the probability that we'll detect an image that has steganographic content, and $P(D|\neg S)$ is the false-positive rate. Conversely, $P(\neg D|S) = 1 - P(D|S)$ is the false-negative rate. To improve the true-positive rate, we must increase the numerator or decrease the denominator.

For a given detection system, increasing the detection rate is not possible without increasing the false-positive rate and vice versa. We assume that $P(S)$ —the probability that an image contains steganographic content—is extremely low compared to $P(\neg S)$, the probability that an image contains no hidden message. As a result, the false-positive rate $P(D|\neg S)$ is the dominating term in the equation; reducing it is thus the best way to increase the true-positive rate. Given these assumptions, the false-positive rate also dominates the computational costs to verifying hidden content. For a detection system to be practical, keeping the false-positive rate as low as possible is important.

3.3 Verifying hidden content:-

To verify that the detected images have hidden content, Stegbreak must launch a *dictionary attack* against the JPEG files. JSteg-Shell, JPHide, or Outguess all hide content based on a user-supplied password, so an attacker can try to guess the password by taking a large dictionary and trying to use every single word in it to retrieve the hidden message. In addition to message data, the three systems also embed header information, so attackers can verify a guessed password using header information such as message length. For a dictionary attack²⁸ to work, the steganographic system's user must select a weak password (one from a small subset of the full password space).

Chapter 4

Steganography: How to Send a Secret Message

This may seem to be an ordinary beginning to an ordinary article. It is not. There's a secret message hidden here, in this very paragraph. It's not in view, and its source is modern. But the art of hiding messages is an ancient one, known as steganography.

Steganography is the dark cousin of cryptography, the use of codes. While cryptography provides privacy, steganography is intended to provide secrecy. Privacy is what you need when you use your credit card on the Internet -- you don't want your number revealed to the public. For this, you use cryptography, and send a coded pile of gibberish that only the web site can decipher. Though your code may be unbreakable, any hacker can look and see you've sent a message. For true secrecy, you don't want anyone to know you're sending a message at all.

Early steganography was messy. Before phones, before mail, before horses, messages were sent on foot. If you wanted to hide a message, you had two choices: have the messenger memorize it, or hide it on the messenger. In fact, the Chinese wrote messages on silk and encased them in balls of wax. The wax ball, "la wan," could then be hidden *in* the messenger.

Herodotus, an entertaining but less than reliable Greek historian, reports a more ingenious method. Histaeus, ruler of Miletus, wanted to send a message to his friend Aristagorus, urging revolt against the Persians. Histaeus shaved the head of his most trusted slave, then tattooed a message on the slave's scalp. After the hair grew back, the slave was sent to Aristagorus with the message safely hidden.

Later in Herodotus' histories, the Spartans received word that Xerxes was preparing to invade Greece. Their informant, Demeratus, was a Greek in exile in Persia. Fearing discovery, Demeratus wrote his message on the wood backing of a wax tablet. He then hid the message underneath a fresh layer of wax. The apparently blank tablet sailed easily past sentries on the road.

A more subtle method, nearly as old, is to use invisible ink. Described as early as the first century AD, invisible inks were commonly used for serious communications until WWII. The simplest are organic compounds, such as lemon juice, milk, or urine, all of which turn dark when held over a flame. In 1641, Bishop John Wilkins suggested onion juice, alum, ammonia salts, and for glow-in-the dark writing the "distilled Juice of Glowworms." Modern invisible inks fluoresce under ultraviolet light and are used as anti-counterfeit devices. For example, "VOID" is printed on checks and other official documents in an ink that appears under the strong ultraviolet light used for photocopies.

During the American revolution, both sides made extensive use of chemical inks that required special developers to detect, though the British had discovered the American formula by 1777. Throughout World War II, the two sides raced to create new secret inks and to find developers for the ink of the enemy. In the end, though, the volume of communications rendered invisible ink impractical.

With the advent of photography, microfilm was created as a way to store a large amount of information in a very small space. In both world wars, the Germans used "microdots" to hide information, a technique which J. Edgar Hoover called "the enemy's masterpiece of espionage." A secret message was photographed, reduced to the size of a printed period, then pasted into an innocuous cover message, magazine, or newspaper. The Americans caught on only when tipped by a double agent: "Watch out for the dots -- lots and lots of little dots."

Modern updates to these ideas use computers to make the hidden message even less noticeable. For example, laser printers can adjust spacing of lines and characters by less than 1/300th of an inch. To hide a zero, leave a standard space, and to hide a one leave 1/300th of an inch more than usual. Varying the spacing over an entire document can hide a short binary message that is undetectable by the human eye. Even better, this sort of trick stands up well to repeat photocopying.

All of these approaches to steganography have one thing in common -- they hide the secret message in the physical object which is sent. The cover message is merely a distraction, and could be anything. Of the innumerable variations on this theme, none will work for electronic communications because only the pure information of the cover message is transmitted. Nevertheless, there is plenty of room to hide secret information in a not-so-secret message. It just takes ingenuity.

The monk Johannes Trithemius, considered one of the founders of modern cryptography, had ingenuity in spades. His three volume work *Steganographia*, written around 1500, describes an extensive system for concealing secret messages within innocuous texts. On its surface, the book seems to be a magical text, and the initial reaction in the 16th century was so strong that *Steganographia* was only circulated privately until publication in 1606. But less than five years ago, [Jim Reeds](#) of AT&T Labs deciphered mysterious codes in the third volume, showing that Trithemius' work is more a treatise on cryptology than demonology. Reeds' [fascinating account](#) of the code breaking process is quite readable.

One of Trithemius' schemes was to conceal messages in long invocations of the names of angels, with the secret message appearing as a pattern of letters within the words. For example, as every other letter in every other word:

padiel aporsy mesarpon **omeuas** peludyn malpreaxo

which reveals "prymus apex."

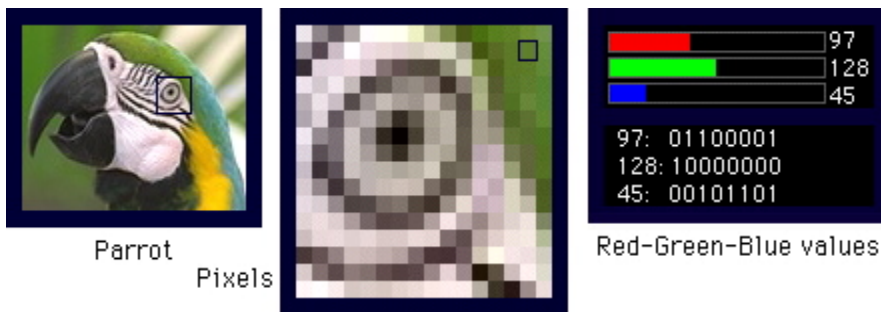
Another clever invention in *Steganographia* was the "Ave Maria" cipher. The book contains a series of tables, each of which has a list of words, one per letter. To code a

message, the message letters are replaced by the corresponding words. If the tables are used in order, one table per letter, then the coded message will appear to be an innocent prayer.

The modern version of Trithemius' scheme is undoubtedly [SpamMimic](#). This simple system hides a short text message in a letter that looks exactly like spam, which is as ubiquitous on the Internet today as innocent prayers were in the 16th century. SpamMimic uses a "grammar" to make the messages. For example, a simple sentence in English is constructed with a subject, verb, and object, in that order. Given lists of 26 subjects, 26 verbs, and 26 objects, we could construct a three word sentence that encodes a three letter message. If you carefully prescribe a set of rules, you can make a grammar that describes spam.

Unfortunately, for serious users, every scheme we've seen is unacceptable. All are well known, and once a technique is suspected the hidden messages are easy to discover. Worse, a ten page document whose line spacing spells out a secret message is completely incriminating, even if the message is in an unbreakable code. A good steganographic technique should provide secrecy even if everyone knows it's being used.

The key innovation in recent years was to choose an innocent looking cover that contains plenty of random information, called white noise. You can hear white noise as a nearly silent hiss of a blank tape playing. The secret message replaces the white noise, and if done properly it will appear to be as random as the noise was. The most popular methods use digitized photographs, so let's explore these techniques in some depth. Digitized photographs and video also harbor plenty of white noise. A digitized photograph is stored as an array of colored dots, called pixels. Each pixel typically has three numbers associated with it, one each for red, green, and blue intensities, and these values often range from 0-255. Each number is stored as eight bits (zeros and ones), with a one worth 128 in the most significant bit (on the left), then 64, 32, 16, 8, 4, 2, and a one in the least significant bit (on the right) worth just 1.



A difference of one or two in the intensities is imperceptible, and, in fact, a digitized picture can still look good if the least significant four bits of intensity are altered -- a change of up to 16 in the color's value. This gives plenty of space to hide a secret message. Text is usually stored with 8 bits per letter, so we could hide 1.5 letters in each pixel of the cover photo. A 640x480 pixel image, the size of a small computer

monitor, can hold over 400,000 characters. That's a whole novel hidden in one modest photo!

Hiding a secret photo in a cover picture is even easier. Line them up, pixel by pixel. Take the important four bits of each color value for each pixel in the secret photo (the left ones). Replace the unimportant four bits in the cover photo (the right ones). The cover photo won't change much, you won't lose much of the secret photo, but to an untrained eye you're sending a completely innocuous picture.

Unfortunately, anyone who cares to find your hidden image probably has a trained eye. The intensity values in the original cover image were white noise, i.e. random. The new values are strongly patterned, because they represent significant information of the secret image. This is the sort of change which is easily detectable by statistics. So the final trick to good steganography is making the message look random before hiding it.

One solution is simply to encode the message before hiding it. Using a good code, the coded message will appear just as random as the picture data it is replacing. Another approach is to spread the hidden information randomly over the photo. "Pseudo-random number" generators take a starting value, called a seed, and produce a string of numbers which appear random. For example, pick a number between 0 and 16 for a seed. Multiply your seed by 3, add 1, and take the remainder after division by 17. Repeat, repeat, repeat. Unless you picked 8, you'll find yourself somewhere in the sequence 1, 4, 13, 6, 2, 7, 5, 16, 15, 12, 3, 10, 14, 9, 11, 0, 1, 4, . . . which appears somewhat random. To spread a hidden message randomly over a cover picture, use the pseudo-random sequence of numbers as the pixel order. Descrambling the photo requires knowing the seed that started the pseudo-random number generator.

Here's a sample. The bear above is an adorable glow-in-the-dark skeleton costumed bear. The bear below is the same photo, now containing a hidden secret picture. To see the secret photo, get yourself a copy of [S](#)

[Tools](#) by Andy Brown and decrypt using the secret password "strange." Or, click [here](#).



With these new techniques, a hidden message is indistinguishable from white noise. Even if the message is suspected, there is no proof of its existence. To actually prove there was a message, and not just randomness, the code needs to be cracked or the random number seed guessed. This feature of modern steganography is called "plausible deniability."

All of this sounds fairly nefarious, and in fact the obvious uses of steganography are for things like espionage. But there are a number of peaceful applications. The simplest and oldest are used in map making, where cartographers sometimes add a tiny fictional

street to their maps, allowing them to prosecute copycats. A similar trick is to add fictional names to mailing lists as a check against unauthorized resellers.

Most of the newer applications use steganography like a watermark, to protect a copyright on information. Photo collections, sold on CD, often have hidden messages in the photos which allow detection of unauthorized use. The same technique applied to DVDs is even more effective, since the industry builds DVD recorders to detect and disallow copying of protected DVDs.

Even biological data, stored on DNA, may be a candidate for hidden messages, as biotech companies seek to prevent unauthorized use of their genetically engineered material. The technology is already in place for this: three New York researchers successfully hid a secret message in a DNA sequence and sent it across the country. Sound like science fiction? A secret message in DNA provided *Star Trek's* explanation for the dubious fact that all aliens seem to be humans in prosthetic makeup!

Maybe, as in *Star Trek*, there really is a message hidden somewhere for humans to find. In the real world, the place to look for such a message is space, and humans have been looking for quite some time. Marconi, the inventor of radio, speculated that strange signals heard by his company might be signals from another planet. To his credit, he was hearing these signals years before his competitors, but today they are known to be caused by lightning strikes.

In 1924, Mars passed relatively close to Earth, and the U.S. Army and Navy actually ordered their stations to quiet transmissions and listen for signals. They found nothing. In 1960, Dr. Frank Drake and a cadre of radio technicians used their 85 foot radio telescope for one of the first extensive studies of signals from space. They listened to Tau Ceti and Epsilon Erdani for 150 hours, and found nothing.

Today, the search for messages from space is underway on an unbelievable scale. The [SETI@home](#) project, based in Berkeley, has convinced millions of people to use their home computers in the search for signals. Their simple marketing trick was to package the calculations in a nifty screensaver, and now SETI@home is the largest computation in history. They've been looking for more than two years, with a telescope a thousand feet wide, but still they have found nothing.

Why have they found nothing? Maybe they haven't searched enough. But there is a dilemma here, the dilemma that empowers steganography. You never know if a message is hidden. You can search and search, but when you've found nothing you can only conclude: Maybe I didn't look hard enough, but maybe there is nothing to find.

Chapter 5

Project on Steganography Application:-

5.1 Requirements:-

- You are to create an application called Steganography.java. All your code will be in this file. This is what you will submit on email.
- Your project is to work with the standard (original) Picture.java class. You shouldn't need any changes to this class in order to make your project work. You will not be submitting a Picture.java file. Instead, I will use my copy to run your program.
- There is a file Secret.bmp on the class web page. Encoded in this file is a question. Use your program to decode the message. Answer the question (in 255 chars or less). Then submit back to me your response encoded in a different bmp picture. You are to copy this bmp file in your file on the shared drive (before 11:30am, May 1). Of course, make sure your own program can decode the response you put in this picture; that way you can be sure that my program can decode the response too.

5.2 Bitmap Files

- First you will need to read your picture as a jpg and then save it in 24-bit bmp format. You will need to use bmp files for this assignment because jpg's are "lossy" meaning that what you write to the file may be changed slightly so that the resulting image can be stored more efficiently. Thus jpg will not work for steganography because jpgs will change the secret message when storing the file to disk. Here are the commands to save your file. You can give it the same name except be sure to put a .bmp file extension on the end. (For example, I loaded "Matt.jpg" and then saved "Matt.bmp").

```
> Picture p = new Picture(FileChooser.pickAFile());  
> p = p.halve().halve();  
> p.saveBMP(FileChooser.pickSaveFile());
```

- There is also a loadBMP method. You can probably guess how this works.
- Note that I reduced my image to 1/4 original size because bmp files take a lot of memory. You will run in to less trouble if your image is smaller (say 100x100 or less).

5.3 Bit Manipulation

- You will need to be able to manipulate the bits stored in numbers. There are three basic bit manipulation operations: and, or, and shift. You will need all three.

- See the BitExample.java example to see how to use these different operations.

5.4 Interaction

- Prompt the user if they want to encode or decode a message.
- Use the FileChooser dialog to prompt the user for an input file.
- If encode, prompt the user for an input message. Encode the message into the picture (details below). Then use the FileChooser dialog to prompt the user for an output file. Save the new picture/message in this file (using bmp format).
- If decode, extract the message from the file. Print the message.

5.5 Encoding/Decoding Method

- You can extract the pixels of your target picture in one big array using the `textttgetPixels()` method.
- Use the first pixel (at spot 0) to hide the length of your message (number of characters). You will limit yourself to messages that are between 0 and 255 characters long.
- After that use every eleventh pixel to hide characters in your message. Start at pixel 11, then pixel 22, and so on until you hide all characters in your message.
- Every thing that you need to hide in a pixel is 8-bits long. The length (in the first pixel) is a byte. You can typecast all the unicode chars to bytes as well.
- Use the method below to hide each byte in an appropriate pixel.

5.6 Hiding Method

The problem with changing the red values in our encode/decode steps, is that these often cause quite visible changes in the resulting image. This is especially true if the pixels that are being changed are part of a large section of uniformly colored pixels – the "dots" stand out and are noticeable. As an option, we can change only the lower order bits of each pixel color (red, blue, and green). This will make subtle changes to each pixel's color and will not be as evident.

Remember that each pixel has three bytes: one byte for red, blue and green colors. Each byte has 8 bits to encode a number between 0 and 255. When we swap out the red color byte for a character, it is possible that we are changing the redness of that pixel by quite a bit. For example, we might have had a pixel with values of (225, 100, 100) which has lots of red, some green and some blue – this is basically a reddish pixel with a slight bit of pink color to it. Now suppose we are to store the character "a" in the red part of this pixel. An "a" is encoded as decimal number 97 so our new pixel becomes (97, 100, 100). Now

we have equal parts of all three colors to produce a dark grey pixel. This dark grey is noticeably different than the dark pink we had before; it will definitely stand out in the image especially if the other nearby pixels are all dark pink.

We want a way to encode our message without making such drastic changes to the colors in the original image. If we only change the lowest bits of each pixel, then the numeric values can only change by a small percentage. For example, suppose we only change the last three bits (lowest three bits) – these are the bits that determine the "ones place", the "twos place" and the "fours place". We can only alter the original pixel color value by ± 7 . Let us think of our original pixel as a bit:

(r7 r6 r5 r4 r3 r2 r1 r0, g7 g6 g5 g4 g3 g2 g1 g0, b7 b6 b5 b4 b3 b2 b1 b0).

And our character (byte) as some bits:

c7 c6 c5 c4 c3 c2 c1 c0.

Then we can place three of these character bits in the lowest red pixel, three more in the lowest green pixel, and the last two in the lowest blue pixel as follows:

(r7 r6 r5 r4 r3 c7 c6 c5, g7 g6 g5 g4 g3 c4 c3 c2, b7 b6 b5 b4 b3 b2 c1 c0).

If we had done this to the example of pixel (225, 100, 100) with character "a", we obtain:

original pixel = (11100001, 01100100, 01100100)
"a" = 01100001
new pixel = (11100011, 01100000, 01100101)
new pixel = (227, 96, 101)

Notice the new pixel of (227, 96, 101) is almost the same value as the old pixel of (225, 100, 100). There will be no noticeable color difference in the image! To retrieve the message, you simply extract the appropriate pixels from the RGB values to reconstruct the secret character.

To accomplish this, you will need to be handy with the "logical and" and "logical or" operators and also the "shifting" operator. Obtain a java reference book to research these operations. You might want to test them out on a small program first or on the Dr. Java command line.

Chapter 6

Semantics-Preserving Application-Layer Protocol Steganography

6.1 Introduction

Steganography, from the Greek “covered writing,” refers to the practice of hiding information within other information. Historically, notions of classical Steganography can be found even centuries before Christ. In recent years, Steganography has become digital: the favorite media for information hiding are images, music scores, formatted and written text, digital sounds, and videos. This evolution of steganographic techniques has received particular attention, as have the security and robustness of such methods [1, 3, 17, 19, 20].

Traditionally, most steganographic systems relied on the secrecy of the encoding system. At present, the security of a stegosystem depends on how well it conceals the existence of a hidden message and in the secrecy of a key, if used, for embedding the message. Protocol Steganography is the art of embedding information within messages and network control protocols used by common applications.

An important consideration in the embedding process is whether it is semantics-preserving, i.e., whether the resulting message still conforms to the protocol specification. That property guarantees that if the message is interpreted at any point during its transmission, it will produce meaningful results. In addition to that, semantic preservation in modified messages helps to make them indistinguishable from unmodified cover messages. Using protocol Steganography, we can embed information in overt channels, in contrast to the use of covert channels, which allow signaling mechanisms to occur where no explicit communication path exists. Advantages of protocol Steganography include achieving greater bandwidth in hidden communication as well as taking advantage of the most widely-used network protocols.

We define two levels of semantics preservation, both of which imply that the stego-message is a correct message within the protocol. *Weak semantics preservation* means that the stego-message, while legal, has a different meaning than the original cover message. *Strong semantics preservation* means that the stego-message has the same meaning as the original cover.

Networking protocols are divided into multiple layers, as shown in Figure 1. The physical layer is responsible for communicating with the actual network hardware (e.g., the Ethernet card), dealing with the format of the bits on the wire. Therefore, it is tied to the local network technology, such as Fast Ethernet or 802.11b wireless. The network layer handles routing, and it is the IP layer of the TCP/IP protocol suite. The network layer is invisible to user programs. The transport layer handles the quality-control issues of

reliability, flow control, and error correction. The TCP/IP protocol suite defines two widely-used transport protocols: UDP and TCP.

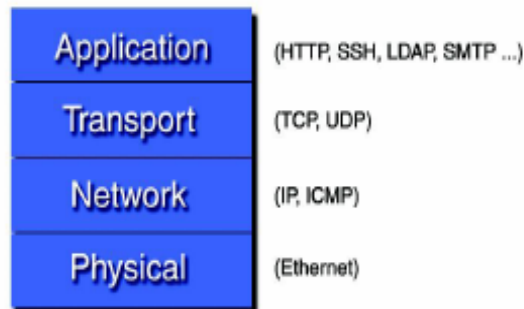


Figure 1: TCP/IP Protocol Suite Layers.

There are several application protocols in the TCP/IP suite, including SMTP (for email service), FTP (for file transfer), SSH (for secure login), LDAP (for distributed directory services), and HTTP (for web browsing, which alone accounts for approximately 70% of all Internet traffic).

A *secure* stego system can withstand an opponent that understands the system (or even has grounds for suspicion), meaning that the opponent cannot determine with a high degree of certainty the existence of the communication. A *robust* system can withstand an active attack, where the adversary makes legal (strong semantics-preserving) changes to the message.

The most obvious way of hiding information within messages is to place data in unused or reserved fields of protocol headers or trailers. However, that method of Steganography is easy to detect using simple intrusion detection systems, or is susceptible to traffic analysis, which makes it insecure and not robust. Even if analyzing the content of the hidden information becomes impossible, perhaps due to encryption, this approach is weak. Our techniques for protocol Steganography aim to achieve strong Steganography, wherein the system is both secure and robust.

Given those goals and the intention to provide means of private communication, our approach to protocol Steganography focuses mainly on transport layer protocols and application layer protocols, although other protocols at different layers of the TCP/IP protocol suite could also be considered.

In particular, this paper describes how protocol Steganography is feasible using the SSH protocol as proof-of-concept.

There are many potential applications for protocol Steganography, considering when information hiding is used for both positive and negative means. When using information hiding for positive means, protocol Steganography is appropriate to achieve private

communication and, in some cases, anonymity and plausible deniability, such as environments where censorship polices restrict web access.

More specifically, protocol Steganography seems to be appropriate for environments where unobtrusive communications are required. For example, in the military and intelligence agencies, even if the content of the communication is encrypted, a significant increase in communications between military units could signal an impending attack.

Hiding information inside regular Internet traffic, such as browsing results, will avoid the need for extra communication, thereby giving no indication to one's adversaries that something is about to happen. On the other hand, considering a framework where the agents that wish to communicate secretly are not necessarily the initiators of the communication, the ability to embed messages in a variety of TCP/IP protocols gives us a much higher likelihood of being able to transmit the secret message within a reasonable time bound.

When using information hiding with malignant purposes, the study of protocol Steganography can help improving the design of network protocols and firewalls. Protocols can be harder to misuse.

Firewalls can be harder to bypass. The remainder of this paper is organized as follows. Section 2 describes the model for secret communication considered in protocol Steganography and discusses its potential advantages. Section 3 presents a summary of the research to date and related work in relevant areas of Steganography. Section 4 explores the concept of protocol Steganography through the SSH protocol, describes a prototype implementation, and discusses consequences and important issues regarding security and robustness of the approach as well. Section 5 analyzes future research opportunities in the area. Finally, Section 6 lists some conclusions.

6.2 Framework for Secret Communication

Our model for protocol Steganography involves two agents that wish to communicate secretly through channels of Internet traffic in a hostile environment (see Figure 2). The agents *A* and *B*, named for Simmons's famous prisoners Alice and Bob, take advantage of a communication path already in place between themselves or two arbitrary communicating processes, the *sender* and *receiver*. We assume that Alice wishes to pass a message to Bob, and may in fact be operating in an environment over which their adversary has administrative control (such would be the case if Alice were an undercover investigator or intelligence operative).

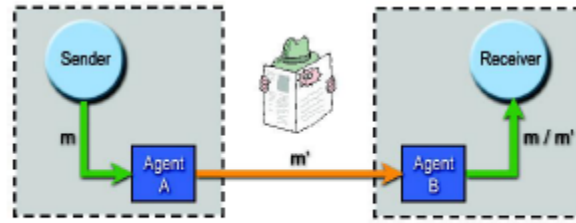


Figure 2: Framework for Secret Communication.

Consequently, two scenarios are possible depending on whether or not agent A and B are the same as the sender and the receiver, respectively. In the first case, agent A and agent B are trying to hide secret information in some of their own harmless messages, as in traditional Steganography models. They both run a modified version of the communicating software that allows them to convey the secret message.

In the second case, agent A and agent B are placed somewhere along an arbitrary communication path, modifying the message in transit to hide meaningful information. In short, both the internal agent and the external confederate might be either end points of the communication or middlemen, acting to embed and extract the hidden message as the data passes them in the communication stream. In fact, the receiving middleman has the option of removing the hidden message, thus restoring and forwarding the original message. The midpoints where agents A and B can alter the message might be within the protocol stack of the sending and receiving machines (which is still distinct from the sending process), or at routers along the communication path. These arbitrary boundaries are indicated by the dashed boxes in Figure 2.

Considering all combinations of internal agents and external confederates and all different points where the message can be altered yields six different roles for the agents, as shown in Figure 3. In this discussion, following the established information hiding terminology, agent A executes the *embedding* process and agent B the *extraction* process, represented in the picture as a circle and a diamond, respectively. As pointed out by Pfizmann, the embedding and extracting processes required the use of a *stego-key*, not shown in the picture. The cover (i.e. the original harmless message) is m , and the *stego-message* (i.e. the message with steganographic content) is m' .

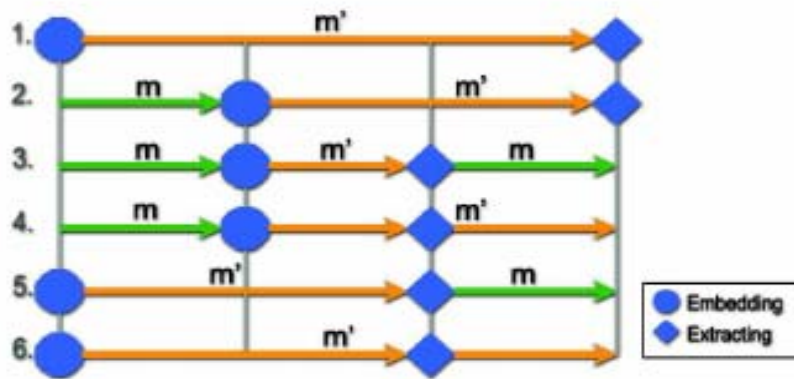


Figure 3: Message Paths.

The six possible sets of agent roles are as follows:

1. *Agent A* acts as sender and *agent B* as receiver the message along the entire path is m_0 .
2. *Agent A* is a middleman, embedding information to the message on its way, and *agent B* acts as *receiver* the message from the *sender* to *agent A*'s location is m , while from there to the endpoint is m' .
3. Both agents are middlemen, and *B* restores the message to its original form the message from the *sender*'s point to where *agent A*'s location is m , from *A*'s to *B*'s is m' , and from there to the endpoint is m again, since extraction of the hidden content and restoration of the original cover message occurred at *B*'s location.
4. Both agents are middlemen, but *agent B* does not restore the message the message from the *sender*'s point to the *agent A*'s location is m , and from *A*'s to the *receiver*'s point is m' , with the hidden information extracted at *B*'s location while the message was in transit.
5. *Agent A* is acting as *sender*, with *B* as a middleman extracting the embedded information and restoring the original message the message from the initial point to *agent B*'s location is m' , and from *B*'s location to the *receiver*'s point is m .
6. *Agent A* is acting as sender and *agent B* is a middleman extracting the hidden information without restoring the message as it travels to the *receiver* the message from the end to end is m' , but *B* gets the hidden content somewhere before the message reaches its destination.

Not every one of these scenarios might be realistic, but cases 1 and 3 certainly are. Therefore, they have been the focus of this study. All the options where the hidden content is extracted but the message is not restored seem very risky; in particular, case 4

wherein the message seen by the receiver is clearly different from that seen by the sender, neither of whom is the agents communicating secretly.

Having the agents acting as middlemen in the communication stream provides several advantages, because any packet that will flow past the locations where agents A and B are can be modified (as long as a semantics-preserving embedding function is available for the transport or application protocol in that packet). The idea is to lower our susceptibility to traffic analysis, as there is no longer a single source/sink for the stego-messages, and there is no specific protocol used. This also allows us to achieve a higher bit rate as well as privacy, anonymity, and plausible deniability, in some cases. An ideal situation would be that agent A is located on the last router inside the sender's domain (the egress router for that domain), and agent B is located on the first router outside the domain (the ingress router). This will have m0 "on the wire" for the minimum possible time, also lowering the probability of detection.

6.3 Adversary Models

Depending on the goals of steganalysis, adversaries can be *active* or *passive*. Passive adversaries observe the communication in order to detect stego-messages, find out the embedded information, if possible, and prove to third parties, when the case requires it, the existence of the hidden message. Active adversaries attempt to remove the embedded message without changing the stego-message significantly, i.e., they attempt to provide strong semantic preservation. In some cases, active adversaries do not need to verify the existence of the message before they attempt to block any secret communication, thus appropriately manipulating the bits of the messages that pass through them is enough (e.g., zeroing unused header fields).

Steganography systems consider both passive and active adversaries, while in watermarking and fingerprinting systems, generally, only active adversaries raise concern. However, most of the literature in stegosystems deals only with passive adversaries. For the purposes of this study, both passive and active adversaries are taken into account, because of hostility of the Internet environment, the constant improvement of routers and firewalls, and the goal of developing not only secure, but also robust, Steganography techniques.

6.4 Related Work

Handel and Sanford reported the existence of covert channels within network communication protocols. They described different methods of creating and exploiting hidden channels in the OSI network model (see Figure 4), based on the characteristics of each layer. In particular, regarding to the application layer, they suggested covert messaging systems through features of the applications running in the layer, such as programming macros in a word processor.

In contrast, the protocol Steganography approach studies hiding information within messages and network control protocols used by the applications, not inside images transmitted as attachments by an email application, for example. They also considered techniques of embedding information that require substituting existing modules of the source code that implements a particular layer, and some that do not. In a similar order of ideas, when agent A and agent B act as sender and receiver, respectively, some application modules will be replaced for embedding and extracting secret messages.



Figure 4: The OSI Idealized Network Model Layers.

Examples of implementation of covert channels in the TCP/IP protocol suite (see Figure 1) are presented by Rowland, Project Loki, Ka0ticSH, and more deeply and extensively by Dunigan. These researchers focused their attention in the network and transport layers of the OSI network model (shown in Figure 4). In spite of that, Dunigan did point out in his discussion of network Steganography that application-layer protocols, such as telnet, ftp, mail, and http, could possibly carry hidden information in their own set of headers and control information. However, he did not develop any technique targeting these protocols. More in detail,

Rowland implemented three methods of encoding information in the TCP/IP header: manipulating the IP identification field, with the initial sequence number field, and with the TCP acknowledge sequence number field “bounce.”

Dunigan analyzed the embedding of information, not only in those fields, but in some other fields of both the IP and the UDP headers as well as in the ICMP protocol header. He based his analysis, mainly, in the statistical distribution of the fields and the behavior of the protocol itself. Project Loki [13, 23] explored the concept of ICMP tunneling, exploiting covert channels inside of ICMP ECHO traffic. All these approaches, without minimizing their importance, suffer from two problems: low bandwidth and simplicity of detection or defeat with straightforward mechanisms.

One such mechanism is reported in Fisk. Their work defines two classes of information in network protocols: *structured* and *unstructured* carriers. Structured carriers present well defined, objective semantics, and can be checked for fidelity en route (e.g., TCP packets can be checked to ensure they are semantically correct according to the protocol). On the contrary, unstructured carriers, such as images, audio, or natural language, lack

objectively defined semantics and are mostly interpreted by humans rather than computers.

The defensive mechanism they developed aims to achieve security without spending time looking for hidden messages: using active wardens they defeat Steganography by making strong semantic-preserving alterations to packet headers (e.g. zeroing the padding bits in a TCP packet).

The most important considerations to their work related to protocol Steganography are the identification of the cover-messages in used as structured carries, and the feasibility of similar methods of steganalysis that target application-layer protocols.

Lastly, Bowyer described a theoretical example without implementation, wherein a remote access Trojan horse communicates secretly with its control using an http GET request. To send data upstream to a faux web server, a remote access Trojan horse could append data at the end of a GET request.

Downstream communication is possible by sending back steganographic images, or embedding data within the HTML (e.g., in HTML tags). Although this approach takes advantage of the semantics of regular http messages, as we intent to do, it is different from our approach because it has low bandwidth and can be blocked by restricting access to certain websites, or by scanning images for Steganography content.

6.5 A Case Study: SSH

The SSH protocol is defined by the Internet drafts [30, 31, 32, 33] of the Internet Engineering Task Force (IETF). It is a “protocol for secure login and other secure network services over an insecure network” [32]. The main goal of the protocol is to provide server authentication, confidentiality, and integrity with perfect forward secrecy. There are several, both commercial and open-source, implementations of SSH. The latest version of the protocol is SSH2 and, being version most widely and currently used, it is the one object of this study.

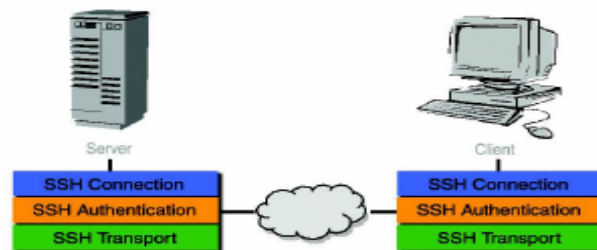


Figure 5: SSH2 Protocol Architecture.

The SSH2 protocol consists of three major components as illustrated in Figure 5:

Transport Layer Protocol

It provides server cryptographic authentication, confidentiality through strong encryption, and integrity plus, optionally, compression. Typically, it runs over a TCP/IP connection listening for connections on port 22.

User Authentication Protocol

It authenticates the client-side user to the server. It runs over the transport layer protocol.

Connection Protocol

It multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol. It provides interactive login sessions, remote execution of commands, forwarded TCP/IP connections, and forwarded X11 connections.

In particular, the Transport Layer protocol defines the *Binary Packet Protocol*, which establishes the format SSH packets follow (see Figure 6). According to the specification [33], each packet is composed of five fields:

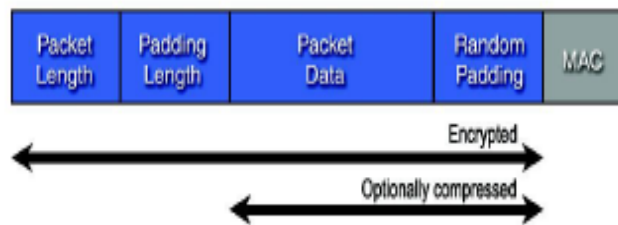


Figure 6: SSH2 Binary Packet Protocol.

1. Packet Length

Number of octets representing the length of the packet data, not including the MAC or the packet length itself.

2. Padding Length

Number of octets representing the length of the padding.

3. Packet Data

The payload, the actual content of the message. If compression has been negotiated, this field is compressed.

4. Random Padding

Arbitrary-length padding, such as the total length of packet length + padding length + packet data + padding is a multiple of the cipher block size or 8, whichever is larger.

5. MAC (message authentication code)

When message authentication is negotiated, it contains the MAC octets. Only initially, the value of the MAC algorithm is none (before authentication).

An SSH client and server start the communication negotiating an encrypted session, followed by client password authentication. Establishing the encrypted session includes exchanging keys and negotiating algorithms (key exchange algorithms, server host key algorithms, encryption algorithms, MAC algorithms and compression algorithms) as well as determining a preferred language. The password authentication process is similar to the one in any remote login application, with the advantage of being more secure due to encryption. The password is prone only to key logging.

The main reason for selecting the SSH protocol as a case of study is the randomness of the content of its packets, which is an excellent factor when trying to blend hidden content in what is considered a “normal” traffic. In addition to that, it is widely used but encrypted, fact that by itself can keep adversaries away from trying to analyze its content, and, as with many other protocols, and pointed out by Barrett and Silverman its design does not attempt to eliminate covert channels.

6.6 SSH Potential for Information Hiding

There are several potential places where information can be hidden without breaking the SSH protocol. Four of those ways of Steganography are described below.

Generating a MAC-like Message

As shown in Figure 6, the SSH2 specification defines the fields:

uint32	packet_length
octet	padding_length
octet[n1]	payload; n1 = packet_length - padding_length - 1
octet[n2]	random padding; n2 = padding_length
octet[m]	mac (message authentication code); m = mac_length

Where `octet [m]` contains the computed MAC. The MAC is normally computed with the previously negotiated MAC algorithm using the key, the sequence number of the packet, and the unencrypted (but compressed, if compression is required) packet data. The MAC algorithms defined by the protocol are `hmac-sha1`, `hmac-sha1-96`, `hmac-md5`, and `hmac-md5-96` whose digest lengths vary from 12 to 20 octets. Therefore, generating a MAC-like message will open the possibility to transmit from 12 to 20 octets of information.

Generating Random Padding-like Message

Basically, this idea is similar to the previous one, but stores the message in the random padding field.

Hiding information in as part of the Authentication Mechanism

The following is the defined format for the authentication request established by the SSH authentication protocol:

octet	SSH_MSG_USERAUTH_REQUEST
string	user name (in ISO-10646 UTF8 encoding)
string	service name (US-ASCII)
string	method name (US-ASCII)
...	method-specific data

The first four fields cannot be modified if we are to conform to the protocol, but there is the possibility of embedding some information in the method-specific data field and still retaining the required semantics.

The format of the response to the authentication request looks like this:

octet	SSH_MSG_USERAUTH_FAILURE
string	authentications that can continue
boolean	partial success

Where authentications that can continue is a comma-separated list of authentication method names.

When the server accepts authentication, the response is:

octet	SSH_MSG_USERAUTH_SUCCESS
-------	--------------------------

but only when the authentication is complete.

We defined a handshake between client and server about what method/type of Steganography is going to be used in the MAC-like message generation or random padding-like message generation. The idea is to take advantage of the parameter exchange done by the regular authentication mechanism.

The two agents, A and B, just need to agree on a covert meaning for the method-specific data sent as an option. Moreover, the protocol recommends the inclusion in the list of authentications that can continue only those methods that are actually useful; it also says that even if there is no point in clients sending requests for services not provided by the server, sending such a request is not an error, and the server should simply reject it. Thus, sending a bogus list of authentications that can continue is not an error.

Another advantage of using the authentication mechanism for hiding data is the fact that the plain text would be encrypted, so no matter what is sent in the string fields, it will not be subject o traffic analysis.

6.7 Adding additional encrypted content to the packet

The previous approaches are only effective when the agents are the same as the sender and receiver (see Figure 2). But the following idea explores having, agent A and agent B, located somewhere along the line of communication of two arbitrary entities that produce SSH traffic. Intercepting the traffic and inserting an encrypted-like portion at the beginning of the encrypted part of the packet is an option, as detailed in Figure 7. The inserted portion consists of two parts: the hidden message itself and a “magic” number that tells agent B there is a hidden message in that SSH packet.



Figure 7: Adding an encrypted portion with a hidden message to a regular SSH packet.

This option offers the advantage of having agents communicating secretly anywhere and using any SSH traffic, but it requires careful study of its susceptibility to traffic analysis. Traffic analysis might indicate that those modified SSH packets are longer than normal, which will indicate suspicion of being a stego-message and, ultimately, compromise the security of the method.

The SSH protocol standard states that any implementation must be able to handle packets with uncompressed payload length of 32,768 octets or less, being the maximum total packet size 35,000 (including length, padding length, packet data, padding, and MAC). Therefore, the length can vary widely. How much variance there actually is in SSH packet length in typical traffic is an open question.

Another question that needs to be answered is where along the communication stream the agents can be placed so an adversary analyzing the traffic cannot perceive the length difference (i.e., the adversary is not able to get both the original packet and the packet containing the stegomessage).

Another issue with this approach is that the “magic” number needs to be of certain minimum length in order to minimize the probability of having the magic number appear naturally in the data stream. We have chosen a four octet magic number for our initial implementation, but this introduces a one in 4,096M chance that we will incorrectly interpret a cover message as a stego-message.

6.8 Advantage of semantic preserving application-layer protocol Steganography

Because of its applicability to a wide range of protocols, we can embed messages in the vast majority of network traffic on the Internet.

The use of non-source stego (en route embeddings and extractions) increases the available bandwidth and complicates traffic analysis because of the ability to choose traffic from a variety of senders and receivers.

Semantics preservation dramatically increases the security of our Steganography.

Chapter 7

Conclusion

Steganography is a fascinating and effective method of hiding data that has been used throughout history. Methods that can be employed to uncover such devious tactics, but the first step are awareness that such methods even exist. There are many good reasons as well to use this type of data hiding, including watermarking or a more secure central storage method for such things as passwords, or key processes. Regardless, the technology is easy to use and difficult to detect. The more that you know about its features and functionality, the more ahead you will be in the game.

However a contrary perspective on encryption was presented by Freeh, a US politician, who commented to the Senate Judiciary committee in September 1998 that “we are very concerned, as this committee is, about the encryption situation, particularly as it relates to fighting crime and fighting terrorism.....we believe that an unrestricted proliferation of products without any kind of court access and law enforcement access, will harm us, and make the fight much more difficult” (cited in Hancock, 2001). He did not however mention Steganography, but viewed encryption as harmful due to its potential uses by terrorists.

To return to the commentary on the use of Steganography in the planning of September terrorist attacks, it would appear that there is no hard evidence of the use of Steganography by terrorists. Until such evidence is produced, one should be wary of an exaggerated response. Nonetheless, it is imperative that the mechanism of Steganography be properly assessed and evaluated before any regulatory authorities ban its use – there is often a knee-jerk reaction to quickly draft legislation, which on the whole causes more harm than good.

Clearly the use of Steganography is largely an American concern at present. It does however pose a potential international problem as the internet operates on a global level. The proliferation of freely available steganographic software and its detrimental applications do send warning signals in the uncertain and criminally-active climate in which we operate today. It becomes clear that it is not a security mechanism to be ignored or avoided – its use could result in serious ramifications and it desperately needs to be considered by all stakeholders from every angle. *Steganography: is it becoming a double-edged sword in computer security?*

Chapter 8

Reference

- [1] Steganography, by Neil F. Johnson, George Mason University,
<http://www.jjtc.com/stegdoc/sec202.html>

- [2] <http://dictionary.reference.com/search?q=steganography>

- [3] The Free On-line Dictionary of Computing, © 1993-2001 Denis Howe
<http://www.nightflight.com/foldoc/index.html>

- [4] Applied Cryptography, Bruce Schneier, John Wiley and Sons Inc., 1996

- [5] Steganography: Hidden Data, by Deborah Radcliff, June 10, 2002,
<http://www.computerworld.com/securitytopics/security/story/0,10801,71726,00.html>

- [6] <http://www.howstuffworks.com>

- [7] <http://www.answers.com>