An Introduction to More Advanced Steganography

Abstract
Steganography has advanced tremendously in the last few years and simple concepts have even been presented on mainstream TV. However, more sophisticated techniques are less well-known and may be overlooked by forensic analysts and even Steganalysis software.

This presentation will showcase several more advanced (and some unpublished) steganographic techniques, some with a very high data hiding capacities. One technique successfully hides 15% to 20% of data in a jpeg and YOU can't tell! That means your 8 MB jpeg image may contain 1.6 MB of covert data! An audio CD contains about 700 MB of data – even a modest 1% capacity allows for 7 MB of data.

This white paper presents several steganographic techniques illustrated by actual software so YOU can decide the effectiveness for yourself. Can you see or hear it? Will it be flagged by Steganalysis programs? We shall see … or not!

Steganography Overview
Information hiding is the science of concealing the existence of data even when it is being sought. Cryptography may very well conceal the meaning of the data, but in some cases, this is inadequate. Often times breaking “unbreakable” cryptography is as simple as a gun to the head or a briefcase full of money … or both!

Steganography is a sub-discipline of the broader science of information hiding and employs numerous technologies to achieve its goals: digital signal processing, cryptography, information theory, data compression, math, and human audio/visual perception, just to name a few.

Steganography has two primary goals: 1) Security – is the hidden data perceptible by either a person or a computer, and 2) Capacity – how much data can be hidden in a given cover file. These two goals are often in competition. The more data you hide, the more likely it is to be found, i.e. it has less security and vice versa. A third goal, robustness, is what separates steganography from watermarking (a 2nd sub-discipline of information hiding).

Robustness is the resilience of your hidden data to image/audio manipulation such as contrast, brightness, cropping, stretching, analog-to-digital-to-analog conversion, etc. There is a large commercial interest in watermarking for digital rights management. Since there is also a trade-off between robustness and capacity, steganographic
programs often do not attempt to be robust, and the techniques presented here are no exception.

There are three levels of failure for steganography: 1) detection, 2) extraction, and 3) destruction. When hidden data is detected, generally, game over. However, if the data cannot be extracted, your objective may still be met. Extraction can be made more difficult by encrypting and/or scrambling the message data.

Preventing destruction refers to maintaining the integrity of the hidden data without significant damage to the cover file. Certainly, one could always delete or overwrite the file in question, but preventing an opponent from destroying your data while keeping the value in the digital work is a challenge. For steganography, once the algorithm is known, you can use the same algorithm to insert randomized data into the same bits that carry the message. Message destroyed, image no worse off.

Finally, for the purpose of discussion, we can rate the perceptibility in 3 easy levels: 1) Indistinguishable, 2) can see/hear distortion when looking/listening closely for it, 3) blatantly obvious to a casual observer.

**Least Significant Bit (LSB)**

Least Significant Bit techniques are the simplest and most widely known of the steganographic techniques. I present them here for those readers who are not already familiar with steganography, but if you already understand LSB, then you'll only miss some cool examples and brilliant writing by skipping ahead.

**Image Background**

Images are composed of picture elements, i.e. pixels. There are three major classes of images: 1) black and white – each pixel is composed of a single bit and is either a zero or a one, representing either white or black; 2) Greyscale – each pixel is composed of 8 bits (in rarer cases, 16 bits) which defines the shade of grey of the pixel, from zero (black) to 255 (white); 3) Full color – also called 24-bit color as there are 3 primary colors (red, green, blue), each of which is defined by 8 bits. There are over 16 million possible colors.

There exist many other representations, but these three, by far, are the most common. For the steganography techniques presented here, we will either use grayscale or 24-bit color.

Considering 8-bit grayscale, each pixel has $2^8 = 256$ possible levels of grey, ranging from black to white. Each bit does not contribute the same amount of information. The Most Significant Bit (MSB) contributes $\frac{1}{2}$ the information, while the Least Significant Bit (LSB) contributes $\frac{1}{256}$th of the information. So, changing that LSB only affects $\frac{1}{256}$th of the intensity and humans simply cannot perceive a difference.
In fact, it is difficult to perceive a difference in $1/16^{th}$ of an intensity change, so we can alter the 4 LSBs with little or no perceptible difference. Below is a mandrill with a lion hidden in it.

![Figure 1](image1.png)

Figure 1 – Left, original image, right, 3 bits hidden

![Figure 2](image2.png)

Figure 2 – Left, 4 bits hidden, right, 5 bits hidden

The lion with the mandrill hidden is not quite as effective at 5 bits as the mandrill’s eyes are easily visible.
Even though the embedded images are not perceptible to humans at 3 bits, and difficult at best, at 4 bits, they are easily detected via a histogram. Next is a histogram of the original mandrill image and the mandrill with 3 bits hidden. Histograms for natural images tend to be smooth, but not so much with hidden data.
For color images, the results are largely the same, except each color plane is hidden within each corresponding color plane. The level of perceptibility and detectability is comparable.

As an additional point, this type of LSB hiding is less common, as we are limited to hiding one image, of the exact same size and type, into another image. It is good for illustrative purposes, but an actual LSB implementation would read the message file as a sequence of bits, and replace the cover image’s bits with those.
Bit Plane Complexity Segmentation (BPCS)

BPCS is another substitution type method, but rather than replacing specific bits, BPCS scans for complex areas of an image, and replaces those with the message data. The idea is that a human cannot distinguish between one complex patch and another complex patch.

![Figure 6 – Randomized data patches](image)

Certainly, looking at these images side-by-side and comparing you can see differences. But, if you were to look at one as a small piece of a larger image, and it was later replaced by the other, you would likely not notice a difference. These images are large, 512 x 512, but the BPCS algorithm uses 8 x 8 patches, making perceptible detection even less likely.

BPCS segments an image into bit planes, and in each plane, the value is either zero or one. Then BPCS scans an 8x8 patch and determines the “complexity.” How much change is there? For instance, a pure black or pure white patch has zero complexity, i.e. no change. A checkerboard pattern of alternating black and white, has the maximum complexity – there are 112 changes when scanned by row, then by column. A simply complexity measure is to divide the number of changes in the image sample, by the maximum, and get a value from 0 to 1.

Experimentally, a good threshold was determined to be 0.3. (It MUST be less than 0.5) So, if there are at least 34 changes (34/112 = 0.305), then the image sample is complex and we can hide our data there. If the threshold is not met, BPCS continues to the next 8x8 matrix and leaves that patch unchanged.

Next, the 64 bits are replaced by the message data. Now the problem is this: What if the message data is not complex? During extraction, the program will skip this bit plane. The solution is to “conjugate” the data by exclusive or’ing it with a checkerboard pattern. The conjugate complexity is always one minus the complexity of the non-conjugate data. This is why the threshold MUST be less than 0.5, otherwise the conjugation solution would not work (if the threshold is 0.7, and the message data’s complexity is 0.6, you cannot conjugate it to meet the threshold.
Now, you must indicate which data is conjugated. The solution in the original paper was to use one bit in the 8x8 matrix to indicate if it is conjugated. Other solutions have been proposed, but this one is simple and effective.

![Conjugation Example](image)

**Figure 7 – Conjugation Example, P is non-complex data, Wc is checkerboard pattern, and P* is the result of conjugation**

The following images illustrate BPCS in action. The histograms show that this technique can be easily detected statistically.

![BPCS in Action](image)

**Figure 8 – Original image, image with hidden data and a threshold of 0.3**
Better complexity measures have been developed since the original inception. These reduce capacity, but prevent highly patterned patches from being considered complex. For instance, a checkerboard pattern is complex, but if modified, humans will perceive the change in pattern.
Figure 11 – Histograms of images with a threshold of 0.2 and 0.1 respectively
High Capacity Hiding in JPEG Images (JPEG)

Jpeg files require a completely different hiding approach than altering bits in the cover file, as these bits will be distorted by the lossy compression process. Before discussing hiding, a brief overview of the compression process is required. Jpeg is designed to work best with 24-bit natural color images, but can also work with grayscale images too.

Jpeg examines an image in 8x8 blocks of pixels, does a color plane conversion from RGB (red, green, blue) to YCrCb (luminance and chrominance), applies a discrete cosine transform, quantizes the results (primary source of loss is right here), and entropy encodes the rest.

Figure 12 – Jpeg compression process
In order to preserve the message data we insert, we must hide after the lossy part of the compression. Since this is after quantization, we choose to hide in the resulting quantized DCT components. This technique boasts a solid capacity of 15 to 20% for a high quality jpeg image. Interestingly, at lower qualities, the alterations are easily noticed as is illustrated by the sample images.

This technique is essentially an adaptive LSB method, for the DCT coefficients. The \( \log_2 \) of the value of the magnitude of the DCT coefficient is compared to the \( \log_2 \) of an alpha factor times the corresponding value in the quantization table. The lesser of these two values is the number of bits that can be hidden.

That number of bits in the DCT is replaced by an equal number of bits from the message. There are a couple of additional considerations. The DC component of the DCT results is altered less, as it is more significant and therefore we don’t want to change it as much. We also employ a block classification routine to increase capacity. By finding blocks that are less uniform (i.e. more complex), we can adapt the number of bits to hide – a busy picture is still a better cover file than a uniform one. However, the results are not nearly so pronounced as in the LSB or BPCS techniques.

Both of the next two images have roughly 22% of their data replaced with message data. Can YOU tell there is anything amiss? Large images are presented so you can take a really close look.
Figure 13 – Mandrill with 22% hidden data, 95% quality
The quality of the jpeg image is an important consideration with this technique. It works well roughly down to 60 – 65% quality, lower than that, there is noticeable distortion. The next image has only 18% of hidden data at 50% quality.
In order to detect this type of steganography, you must examine the DCT coefficients themselves, a simply histogram of the file is not effective, as seen next. There is nothing distinct about the histograms with or without the hidden data.
Figure 16 – Boxer with 18% hidden data, 50% quality
Hiding in the Silence of Sound (HISS)

Hiding in the Silence of Sound is a misnomer as this technique actually hides in low amplitude portions of an uncompressed wave file. The HISS technique works with both 8-bit and 16-bit samples at any sampling rate for either one or two channels. This includes the 16-bit, stereo, 44.1KHz audio CD format.

HISS scans a wave file, finds the length of audio samples within a user-specified threshold, and alters that length based on the message data. It is adaptive in the sense that longer periods of low amplitude samples allow for a greater number of bits to be embedded. The low order bits of the new length of samples within the threshold are the message.

There are 4 input parameters: Amplitude Threshold (2% to 24%), Minimum Length (8 – 64 samples), Max bits to hide (1 – 9), and Capacity Factor (1 – 5). These parameters affect the capacity and security of the resulting stego-wave file. The limits have been determined empirically and allow for a capacity with some level of perceptibility.

The amplitude threshold, input as a percentage, is the amplitude, below which we consider “low”. Since some audio samples may not peak at the maximum, and the negative peak may not equal the positive peak, we choose a percentage as opposed to a raw sample value level. This way, for quieter music, we will not inadvertently use a threshold higher than what actually exists in the audio sample! We scan the wave file, find the positive/negative peaks, and scale the threshold accordingly.

The minimum length is the minimum number of samples that be within the amplitude threshold in order to be used to hide data. The lowest value is 8. Given that the capacity factor (discussed shortly) is at least one, HISS can hide 2 bits.

The maximum number of bits to hide limits how much time will be affected in extremely long samples of low amplitude audio. Nine bits represents and change of +/- 512 samples which in real time is 11.6 milliseconds at 44,100 Hz. In practice there are not many low-amplitude blocks of this length. We must have at least 1024 samples within the threshold which is about 23 milliseconds.

The capacity factor is a user-specified parameter that affects how many bits are hidden in each span of low amplitude. For 8 samples, the log base 2 is three. Hiding 3 bits in a sample length of eight could increase the new length to 15, thus nearly doubling the time. On the contrary, it could also reduce a length of 15 to 8, nearly cutting the time in half. When the time is altered that much, it becomes easily noticeable. So, the capacity factor is subtracted from the logarithm result to determine the actual number of bits to hide. With 8 samples, we can hide 2 bits, resulting in a maximum change from 8 to 11 or 11 to 8 samples – roughly about 25%.

Supposing that the number of samples within the threshold is N=104. The floor of \( \log_2 104 \) is 6. From that we subtract the capacity factor (minimum is one) and get 5. This
means we can embed 5 bits of message data. We read the 5 bits of message data (suppose they equal 20 in value, 0x14). Next we mask off the low order five bits from the sample length (0x0068 bitwise AND 0xFFE0) resulting in the value of 96 (0x0060). The message bits (20) are added to 96 to get the new length, 116. Now we need 116 samples within the threshold, as opposed to the original 104, so we insert 12 samples.

They are inserted uniformly throughout the 102 samples increasing the length to 116 so that upon extraction, we find 116 samples within the threshold. The integer \( \log_2 116 \) (0x74) is 6. Subtract the same capacity factor of 1 to get 5. Note the low order 5 bits of the hexadecimal value for 116 is 0x14.

In the sample wave diagram on the next page, there are 30 samples within the threshold. \( \log_2 30 \) is 4. Subtract 1 is 3. So we can embed 3 bits of the message data. If the capacity factor were 2, then we would embed only 2 bits, and if it were 3, we would embed only 1 bit.

Making the threshold lower decreases the number of samples within it, reducing capacity, while increasing the threshold increases the number of the samples.

![Audio sample, a portion shown within a threshold](image-url)

Figure 17 – Audio sample, a portion shown within a threshold
Removing samples to decrease the overall length within the threshold is a simple operation. Using the overall length of the samples, delete the required number of samples dispersed throughout. Start with the first sample and delete with an interval of the original number of samples divided by the number of samples to remove. We must be careful not to delete the only maximum or minimum peak value as that would distort extraction.

Inserting the samples is more difficult since we must determine amplitude. The exact amplitude value is not critical as long as it does not exceed the threshold. Determine the insertion point and get the samples on either side, take the average amplitude of the two samples to determine the new amplitude. This will always guarantee that the max/min peak values will not be exceeded and will not introduce any high frequency harmonics.

Stereo signals require another consideration: What if one channel is within the threshold while the other is not? If one channel were to lengthen, the two would be out of synchronization, which becomes easily noticeable. There are two approaches that have been implemented for this scenario: 1) overlap - when both channels are within the threshold, and 2) channel averaging – when the average of the two channels is within the threshold.

Both approaches are successful, but as you might expect, the overlap technique has a vastly reduced capacity (though security is increased) when compared to the averaging technique.

Extraction requires that the same threshold and capacity factor are specified. The stego-wave file is scanned, the threshold values determined, and then HISS scans the file for sequences of samples within the threshold. The log base two of the length is taken to determine the maximum number of bits embedded in the sequence, the capacity factor is subtracted, and the resulting number of low order bits are added to the message file. When the entire wave file is scanned, or the length of the message is reached, extraction is complete. LATM stores the length in the first 4 bytes of the data embedded, so there is a small overhead.

HISS is especially sensitive to the cover file. Classical music, for example, which has precise single tones, is a terrible cover medium. Even slight alterations are easily noticed by a casual observer. Jazz is a little better, but not much. Typical pop and country music are decent cover files. The best cover files for this technique are hard rock and heavy metal, as my parents call it noise anyway.

HISS is not easily detectible. For one, the percentage able to be hidden with little or no noticeable distortion is small one the order of 1%, so very little data is altered. Second, there are no values of samples altered, but rather fake ones are inserted or legitimate ones are removed.
A histogram comparison shows nothing unusual. The histogram on the left is a clean sample, while on the right is a heavily modified sample, modified to the point of the distortion being noticeable to even the casual observer.

Figure 18 –Histograms of clean and heavily modified audio

Conclusion
In 2005 I mined over 2200 steganography papers alone, and that number was a small fraction of the number of papers on watermarking. Now, six years later, I am confident there are 5000+ papers.

Several steganographic techniques have been presented in this paper, designed mainly to raise your curiosity and intrigue. They can successfully hide/extract arbitrary data and remain visually undetectable. The recent revelation that Russian spies used steganography to communicate only highlights the need for continues research. These programs are a stepping stone to truly sophisticated and nearly undetectable steganography.
Bibliography


[4] Tao Zhang, Zhaohui Li, Peipei Shi, Statistical Analysis Against Improved BPCS Steganography, Nankai University, China

[5] Silvia Torres, Mariko Nakano, Ruben Vasquez, Robust Steganography using Bit Plane Complexity Segmentation, Escuela Superior de Ingenieria Mecanica, Mexico


