Simple Additive LSB Steganography in Losslessly Encoded Images

Arik Z. Lakritz, Peter Macko, and David K. Wittenberg

September 26, 2007

Abstract

Kurak and McHugh [7] described LSB encoding, a steganographic technique to hide data in the low order bits of an image. Moskowitz, Longdon, and Chang [10] point out that most pictures have areas where the low order bits are clearly non-random, so if those bits are replaced with (apparently) random bits, the replacement is easily detected. Other attacks on LSB encoding are based on the observation that even low order bits which appear random when viewed alone correlate with the higher order bits in the region. Several groups [8, 9, 2, 4] have showed how to detect steganography using this fact. We introduce XLSB, which solves both problems without complicating the decoder, the first by avoiding the areas of an image in which the low order bits are non-random, and the second by adding small values to the data instead of replacing the low order bits. We give test results showing that several steganalysis techniques do not detect XLSB, even at very high data rates ($\sim$ 4 bpp). This is several hundred times the data rate at which the steganalysis techniques detect ordinary LSB encoding.

1 Introduction

Steganography predates cryptography as a method of hiding information. The first known uses were described by Herodotus around 480 BCE. In one a message was written on the wood of a writing tablet, rather than scratched in the wax which covered the wood, as was usually done. In another, a message was tattooed on a slave's head, and after his hair grew back, he was sent with the message hidden under his hair [13]. David Kahn wrote a classic short introduction to the history of steganography [6].

For most of its history, steganography consisted of thinking of new hiding methods and hoping that no one else thought to look in those places. Moskowitz, Longdon, and Chang [10] changed the basic model of steganography to be closer to the standard model of cryptography. Instead of trying to pick a method of communicating which others are unlikely to look for (so called "security through obscurity"), the steganographer now assumes that the steganalyst knows what steganographic system is in use, and tries to make it impossible to distinguish

objects with hidden data from unmodified objects without knowledge of the steganographic key. By specifying the technique and keeping the key hidden, one can compare steganographic techniques mathematically.

A classic steganographic technique is to replace the lower order bits in a digital image (which were thought to be essentially random noise) with the message one wishes to hide. By compression, encryption, or both, the message is made to look random so that obvious non-randomness does not announce the existence of a steganographic message. See Kurak and McHugh [7] for one discussion. If one assumes that the low order bits are essentially random, this should be undetectable. Unfortunately, as Moskowitz, Longdon, and Chang [10] show, this assumption turns out to be false. In almost every photograph we looked at, large sections (typically a quarter to a third of the area of the image) had low order bits which did not pass simple tests [11] for randomness. When the low order bits of the picture were viewed alone, these areas stood out visually.

More recent work has shown that even in areas of an image in which the low order bits appear random, they correlate with nearby high order bits. There are several steganalysis tools based on this observation [8, 9, 4, 2].

We introduce XLSB, a steganographic system which is not detected by either class of techniques.

2 Computational Model

We consider a world in which an encoder named Alice with reasonable computational power wishes to send hidden messages to a decoder named Bob whose computational power is somewhat limited, and who may not be able to obtain sophisticated software for decoding. Eve is an eavesdropper who tries to determine which images contain hidden data. Eve has large, but not unlimited, computational resources. This corresponds to sending information to political dissidents in an authoritarian country. As much as possible, we put all the difficult computations at the encoder's end, so that the decoder's program can be quite simple to write, and will execute reasonably quickly even on slow hardware.

We use the term *cover* for the original image in which we wish to hide data, *message* for the data we are trying to hide, and *stego-object* for the image with the message embedded in it.

It is important to realize that a major difference between cryptography and steganography is the comparative computing power of the encoder and the attacker. In both cases, the attacker (Eve) is assumed to have much greater total resources than Alice or Bob, but in steganography Alice can force Eve to spread her resources over many more possible messages. In cryptography, Alice encrypts a great many messages, from which Eve chooses a few to attack. Since the encryption must be done on almost all Alice's messages, and Eve can choose which few to concentrate on, Eve can bring to bear much more computing power per message than Alice.

In contrast, a steganalyst must look at large numbers of possible stego-

objects, many of which are simply covers before finding a real stego-object. Alice can force Eve to study more possible stego-objects just by sending more unmodified images, or equivalently by hiding a message in a few images in a large web gallery. Even if Eve only suspects that the gallery contains stego-objects, she must study a large fraction of the images. Thus Alice can afford more computation per message than Eve can, despite Eve's much greater overall processing power. It still must be easy for Bob to determine which images are covers and which are stego-objects.

3 Algorithm

This paper introduces XLSB, a steganographic system which incorporates two separate techniques (block selection and additive LSB encoding) which together constitute a relatively simple LSB steganographic system. Essentially all of the added complexity is in the encoder, leaving a relatively simple decoder. This is important as our model assumes an encoder with reasonable resources trying to get information to a resource-poor decoder.

We define the *hiding area* to be the part of each byte which contains embedded data. Note that in color images, each color component is represented by a byte. The hiding area is the lowest order bit or bits of each byte and is typically one or two bits per byte. We use H for the number of bits in each byte's hiding area. Note that bits outside the hiding area may be modified to help hide the steganography, but Bob need not look outside the hiding area.

3.1 Algorithm Overview

XLSB proceeds in three phases. The first phase, *block selection* (subsection 3.2), chooses blocks which are sufficiently irregular to make it difficult to detect steganography. Given the amount of data to be embedded, this phase checks if there are sufficiently many relatively irregular blocks to hold the amount of data and determines which blocks to use.

The second phase is to calculate exactly what data to embed. This consists of compressing or encrypting (or both) the data to make it appear random, and then breaking it into chunks of appropriate size and calculating a checksum for each block.

The third phase, *additive LSB encoding* (subsection 3.3), modifies the cover image to hold the necessary data.

Our testing used three color components image blocks of size 9×9 pixels, which have capacity $3 \times 9 \times 9 \times 2/8 \approx 60$ bytes for color images with a hiding area of two bits.

3.2 Block Selection

The word *block* is used in cryptography to mean a string of data (either before or after encryption) and in discussing images to mean a small rectangular area

of the image. In this paper, we will use blocks in the image sense, and use *chunks* to mean a string of data.

Block selection is the process of choosing which blocks to embed data in. We choose blocks which look as much as possible like a typical image. While we have an intuitive idea of what a typical image looks like, it is very hard to formalize that idea, and we do not try to do so. Obviously typical does not mean random, as it is not clear what population to choose over, and a simple calculation of an arithmetic mean returns a uniform grey. Real images have large areas which are very regular, and we avoid those areas. It is slightly paradoxical to note that an image which appears "typical" over its entire area is rare indeed.

Block selection in XLSB employs two types of tests, choosing blocks which are both relatively "random" to avoid the attack of Moskowitz, Longdon, and Chang [10] and reasonably "rough" to avoid the Gaussian smoothing attacks of Avcıbaş, Memon and Sankur [5]. The Gaussian smoothing attack is discussed further in Section 5.1. Because Alice has a computational advantage over Eve (on a per message basis), we can afford to add other tests as they are found useful. There is always an arms race in cryptography and steganography. As new methods of recognizing overly regular areas are discovered, they will have to be added to XLSB.

We divide the image into blocks of $N \times M$ pixels (usually 9×9). For each pixel in the block, we run the various tests on a bit-string composed of the bits from the hiding areas of the pixels from an $N \times M$ square (reading the $3 \times 9 \times 9 \times H$ bits as a string in row major order) surrounding the pixel. Some of these pixels are outside the block under consideration.

We measure a randomness and a roughness value for each block, then linearly combine them to get a suitability score. We then determine the number of blocks n which is required to hold the message. XLSB then uses the n blocks with the best suitability score. The algorithm has a security parameter which specifies how "random" a block must be for it to be used. If there are not enough blocks with suitability scores above the chosen threshold, it returns a "picture too small" error. For each block chosen, the encoder embeds part of the message together with a checksum in it (called, jointly, a *chunk*.) Bob can determine which blocks contain data by testing the checksums of all of them.

3.2.1 Randomness Testing

In each block we measure the randomness of the hiding area using part of the NIST suite of randomness checkers [11]. Our current implementation uses the first two of the 16 tests in the suite: frequency test and runs test. We have modified the frequency test to return a floating point value which grows as the apparent randomness decreases instead of returning 0 or 1, depending on some threshold. We currently combine the results of the tests linearly with weights of 0.65 for the frequency test and 0.35 for the runs test.

3.2.2 Smoothness Testing

We measure the smoothness of each block using the Gaussian smoothing and compare techniques of Avcıbaş, Memon and Sankur [5]. XLSB does this by computing the difference between the value of a pixel in the image and the local Gaussian mean around that pixel as determined by the Gaussian smoothing filter. We choose areas where $|c - \hat{c}|$ (where c is the color intensity vector of the tested pixel and $\hat{c}$ is the corresponding color intensity vector in the smoothed image) is high in order to avoid using smooth blocks for which steganalysis based on Gaussian smoothing is likely to succeed.

The measurements by Avcıbaş, Memon and Sankur [5] have led us to observe that large smooth flat areas of single or gradient color tend to appear regular-looking with little noise in them. Embedding secret information in these areas randomizes the noise and makes it irregular, since our method does not distribute data evenly throughout the picture. Gaussian smoothing detects these changes in a manner similar to the attack of Moskowitz et al. Therefore, such areas in the picture should be avoided when embedding data. On the other hand, areas with many edges and a large variety of colors usually contain much random noise, and therefore should be preferred.

3.3 Additive LSB Encoding

The more difficult problem is that early LSB steganography schemes treated the low order bits as an independent random variable. While those bits do indeed appear random (when viewed alone), they correlate with the values of nearby pixels and with the high order bits in the same pixel. This makes them susceptible to Sample Pair analysis [2], Color Cube analysis [8], and related attacks.

We introduce *additive LSB encoding* which avoids this problem by potentially modifying the entire byte, rather than just the low order bits. Instead of replacing the LSBs with the data we wish to embed, we add to each byte the smallest number which results in the correct value for the LSBs we wish to embed. This addition may carry into the high order bits. To avoid increasing the average intensity values, we subtract a small value (2 when H is 2) from each byte before increasing the value. The total effect is essentially to add a small pseudo-random value to each byte, which in effect is simply adding noise to the entire signal. This makes it much more difficult to detect our changes. Because XLSB adds to the entire byte, instead of just changing the LSBs, XLSB makes much smaller changes in the second order statistics than earlier algorithms. In particular, XLSB makes very little change in the relationship between the low order bits and the higher order bits.

If the addition causes overflow (or underflow), we replace the value with the largest legal (or smallest legal) value. This might cause additive LSB encoding to disrupt the distribution of values near zero and near the maximum byte value (usually 255), but this overflow or underflow is so rare as to not be an issue. In a majority of the images we examined, the blocks which were selected had

no pixels with values of 0, 1, 2, 253, 254, or 255, either before or after XLSB steganography. Thus the possible changes in distribution of values near the maximum value are unlikely to be visible, and can be avoided entirely by not using blocks in which overflow or underflow occur.

The critical advantage of additive LSB encoding over normal LSB encoding is that it does not create unnecessarily large differences between consecutive pairs of pixels, which is what many steganalysis systems detect with their second order statistics. Consider hiding 1100 in two 4-bit pixels with values 0100 and 0011, which differ by one. After performing traditional LSB, we obtain 0111 and 0000; these two values now differ by seven. On the other hand, XLSB encoding would produce 0011 and 0100, which still differ by only one.

For testing, we used a hiding area of two bits of data per 8-bit byte of the image. PNG uses three bytes per pixel (red, green, blue), so we embed up to 6 bpp (bits per pixel). If one were more worried about detection, one could embed one bit per byte (3 bpp).

3.4 Identifying Message Blocks

Bob needs some way to determine which blocks have data in them. The method of this determination is the steganographic key. If the data is encrypted, Bob can check to see that it decrypts sensibly, and if so, he knows that the block is in use. This requires that each image block be large enough to hold a cryptographic block. Our current implementation uses blocks which are too small to allow this, but the blocks could easily be enlarged. If the data is not encrypted, then one needs to use a hash function to identify the blocks in use.

Our current implementation uses MD5 to generate a 16 byte checksum, which is sent as the last 16 bytes of each chunk. Since each block can hold 60 embedded bytes, this leaves enough room for $60 - 16 = 44$ bytes of data in each block. We could either use a salt to MD5 as the steganographic key, or the key could be an agreed upon string used to pad the short blocks we use to the 512 bits in a MD5 block.

It is known that the short size of an MD5 hash allows one to search for different messages with the same hash using a "birthday" attack after which an attacker could replace one data chunk with another. This is not a concern here, as we assume the attacker does not know what salt is used for the MD5 hash. Knowing the salt is equivalent to knowing the steganographic key. We assume Eve does not know the key.

Bob does not need to do statistical tests, so his code is relatively simple and requires few computational resources.

4 Image Formats

There are many image formats in use. XLSB should work equally well with any lossless image format, or indeed (with the obvious minor modifications) for audio data. The only part of the algorithm which depends on the format is how

image blocks are defined and chosen in each format. We have concentrated on PNG formats [12]. We expect XLSB to work on TIFF [1] and BMP formats but have not studied them. If a format is sufficiently compressed, embedding data will visibly change the image, so those formats are not appropriate for steganography, or can only be used with very low data rates. Steganography depends on the cover having some noise, and in a highly compressed format, there is not enough noise for steganography. In general, the lossier the format, the less suitable it is for steganography. Fridrich, Goljan, and Du [3] explain why images which have been previously converted from JPEG to any other format are poor choices for steganography.

5 Steganalysis

There is a significant literature on steganalysis, and it is important to consider how XLSB copes with known means of detecting steganography. We applied several published steganalytic attacks to XLSB, and found that XLSB could encode large amounts of data without being detected by any of them. In some cases, there was a statistically detectable difference between a population of cover images and the population of stego-objects XLSB creates from them, but the difference is only detectable over populations of images, as there is tremendous overlap of the samples. By choosing safe images to use for XLSB steganography, even the population differences can be hidden. We tested on a set of 39 images. We used all the images in the data set, and did not select them for steganographic reasons.

We show data only from images of 640×480 pixels in order to show the algorithm on images of a size which is commonly used on web pages.

Note that the steganographer can select images in which the steganography is least obvious. Because the steganalyst has no knowledge of what images the steganographer chooses from, the steganographer need use only a small fraction of the images he considers. In this paper we ignore that ability. Using it would, of course, improve our results.

5.1 Non-Random or Smooth areas in images

For now, XLSB defends against the attack of Moskowitz et al. [10] and the attack of Avcıbaş, Memon and Sankur [5] by using the same tests they use, except that in XLSB they are used prospectively. That is, XLSB takes a test which a cryptanalyst might use to detect steganography, and if steganography is likely to be detected, omits that block from the blocks used to embed data.

Avcıbaş, Memon and Sankur [5] proposed to use ANOVA Image Quality Metrics to detect steganography. They did this by using a Gaussian smoothing convolution filter, which gives a local mean value for the area around each pixel. They use that mean (which is also the maximum likelihood estimate of pixel value in the original image under the Gaussian assumption) as the pixel value

in the smoothed image. If the value of a pixel varies greatly from the value in the smoothed image, that is taken as evidence of steganography.

Moskowitz *et al.* look for areas of low order bits which they expect to appear non-random based on the contents of the image they are looking at. They however do not specify which features of an image tend to correspond to non-random low order bits, but from our experience, a stegoanalyst can gain such intuition by studying a rather small number of unmodified images. The non-random areas generally tend to correspond to very bright, dark, or single-color areas of an image. The safest way around this is to embed messages only in blocks that look sufficiently random, preserving the non-random areas.

5.2 Statistical Tests

There are many different measurements which one can make to describe images. The point of steganalysis is to use some of those measurements to determine if an image has been tampered with. Here we discuss some of them.

Lee *et al.* point out that one can use two part statistical tests to detect steganography in color [8] (Color Cube Analysis) and grey scale [9] images. They use the ideas from Dumitrescu, Wu, and Wang's Sample Pair Analysis [2] which measures the length of hidden texts very accurately. Dumitrescu *et al.* built on the earlier RS-steganalysis work of Fridrich, Goljan, and Du [4].

5.2.1 Number of Colors

We define LSB2 as a measure of the number of colors in an image. Consider the high order 6 bits of each color. For each 18 bit value which appears, we count the number of different values of the 6 low order bits (2 bits of each color). The maximum value for LSB2 is 64. LSB2 is the average of these counts for values of high order bits which appear in an image.

For our cover images, LSB2 ranged from 8.49 to 11.49. After embedding 6 bpp over about three quarters of the image, for a density of about 4 bpp (15% of the total bits in the image are embedded bits), LSB2 ranged from 8.52 to 11.62. While for each picture, LSB2 increased by between .02 and .4 when we embedded data, it is certainly not enough of a change to draw a threshold for determining if data is hidden. (Note that the smallest LSB2 value before embedding does not correspond to the smallest LSB2 value after embedding.)

5.2.2 Color Cube Analysis

Color cubes are three dimensional sets of points made up of the red, green, and blue intensity values for pixels in pictures to be analyzed. The steganalytic technique is based on the observation that with a high probability, comparable color cubes have the same noisiness levels.

Lee *et al.* [8] described a statistical test that can be done on the set of colors used in the picture. They represent the color space as $\mathbb{Z}_{256} \times \mathbb{Z}_{256} \times \mathbb{Z}_{256}$ (where $\mathbb{Z}_{256}$ is the additive group of integers modulo 256). This color space has

point (r, g, b) if and only if a color corresponding to that point is used in the image. The color cube is defined as a cube with vertices (r, g, b), $(r + s\delta, g, b)$, $(r+s\delta, g+s\delta, b)$, $(r, g+s\delta, b)$, $(r, g, b+s\delta)$, $(r+s\delta, g, b+s\delta)$, $(r+s\delta, g+s\delta, b+s\delta)$, and $(r, g + s\delta, b + s\delta)$, where r, g, b are even, δ is odd, and $s \in \{-1, 1\}$. Right cube is defined as a cube with $s = 1$, left cube has $s = -1$, and the order of the cube is the number of vertices that are points in the color space.

The color cube analysis method assumes that in an unmodified cover, for each delta and each order, the numbers of left and right sub-cubes is about equal, and proceeds to check for statistically significant deviations from this symmetry.

Traditional LSB embedding adds noisiness to the color cubes by adding points within the cubes, which can be detected when the noise is not symmetric with the comparable cubes.

XLSB embedding 6 bpp of data only changes the results of color cube analysis [8] slightly, sometimes increasing and sometimes decreasing the measured amount of data embedded.

5.2.3 Sample Pair Analysis

Sample Pair Analysis [2] is based on the idea that in a digitization of a continuous signal (like audio or photographic data), it is likely that consecutive points will have values which are close together. Sample pair analysis detects steganography by statistically measuring the difference between adjacent points. Because XLSB does not increase that difference nearly as much as standard LSB embedding does, sample pair analysis is not sensitive enough to XLSB to have a threshold over which one should suspect steganography. The distribution of measurements from covers has enormous overlap with the distribution of measurements for stego-objects.

Using Sample Pair Analysis [2], we find that embedding 6 bpp sometimes increases and sometimes decreases the measured bit rate. Untouched pictures had measured bit rates of 1.25 to 3.49, while pictures with embedded data had measured bit rates of 0.49 to 3.38. In most cases the measured bit rate went down. Again, there is so much overlap between the distribution of untouched pictures and the distribution of pictures with embedded data that it is not possible to use a threshold function to separate them.

With normal LSB encoding, Sample Pair analysis detects steganography at .03 bpp, while XLSB is not detected at 200 times that level.

5.2.4 RS Steganalysis

Fridrich *et al.* [4] introduced *RS steganalysis* which uses the distribution of regular and singular groups to detect steganography in uncompressed images with an exceedingly high success rate. They claim that they can reliably detect secret messages with density as low as 0.03 bpp. Note that they assume that the hidden message is randomly distributed throughout the image's low order bits,

which makes decoding (and detection) more difficult. In XLSB the hidden message is placed deterministically, but can still hide 6 bpp without being detected by RS Steganalysis.

!!! how many bits are we embedding?? It's not 6 bpp, because we don't use every pixel. Is 4 a good number? We may need some testing here.

When we embed 6 bpp (2 for each color), we slightly decrease the estimated percentage of flipped pixels RS Steganography returns, but the change is detectable only because the distributions are slightly different. They still have an enormous overlap. The original images have values (RS measured embedding densities R + G + B) ranging from 1.28% to 2.04%, while the images with data embedded range from 0.75% to 1.95%. We do not know why XLSB decreases the apparent density of steganographic bits below the values measured on unmodified images.

6 Results

Table 1 shows results of 3 steganalysis techniques on images in three forms: First the cover image, then the image with a jpeg embedded in it, and finally the image with text embedded in it. Notice that while each of the measurements changes with the data embedding, none of them consistently change in the same direction. We do not understand why XLSB embedding sometimes reduces the measured embedding rate compared to the cover image. Further note the large overlap in the ranges of each measurement with and without embedding. This overlap in the ranges makes it all but impossible to use these analysis techniques to detect XLSB. All of the images are available on-line at the URL at the end of the paper. We discuss them briefly here.

Picture 1075 is a photograph of a banana on a light colored desk. All the colors are in the yellow range, and there is relatively little contrast. We do not know why the analysis tools flag that photograph, nor why embedding data in it reduces the measured embedding rate so sharply. Picture 1086 is a photograph of a large rectangular fluorescent light fixture. Like 1075, it has rather little contrast, and most of the image is pale yellows.

7 Images Used

Most of the images we tested were photographs taken by Jason C. Wu with a Panasonic DMC-FZ30 8 Megapixel camera in TIFF mode. Other images were taken with a Olympus Camedia C4000 Zoom in TIFF mode. We converted the TIFF images to PNG format with ImageMagick `convert`. For smaller images, we used ImageMagick to reduce the size to 640×480.

All of the images are available on the web site:
`http://www.cs.brandeis.edu/~dkw/xlsb`

File Name	LSB2	Color Cube		Sample Pair	RS Group
		χ^2	P-value		
pa150034.png (portrait)	12.84	2.61	0.08	0.08	2.01
pa150034.png with jpeg	12.87	4.20	0.24	0.20	1.08
pa150034.png with text	12.84	3.73	0.19	0.091	1.16
1073.png (buildings)	16.37	2.24	0.05	5.22	2.37
1073.png with jpeg	16.47	3.21	0.14	5.56	2.63
1073.png with text	16.45	2.72	0.09	4.37	1.90
1086.png (flourescent)	20.45	4.82	0.32	7.45	4.02
1086.png with jpeg	20.77	4.51	0.28	7.33	3.96
1086.png with text	20.49	5.48	0.40	7.69	4.22
1075.png (bannanas)	11.15	39.93	~ 1	15.56	8.31
1075.png with jpeg	11.30	32.16	~ 1	6.57	3.62
1075.png with text	11.25	36.63	~ 1	8.28	3.02
P1010053.png (bird)	8.28	3.76	0.19	2.74	2.17
P1010053.png with jpeg	8.37	1.66	0.02	1.88	1.76
P1010053.png with text	8.36	2.27	0.06	1.24	1.46

Table 1: In each group of three images, the first is the cover image, the second a stego object consisting of the cover with a jpeg XLSB embedded, and the third a stego object consisting of the cover with a text message embedded. The same 35.5 Kbyte jpeg and 24.8 Kbyte text messages were used for each image. The "color cube" column (see Section 5.2.2) is the test of Lee *et al.* [8]. The P-value is the calculated probability that the image contains steganographic data. The Sample Pair (see Section 5.2.3) column is the test of Dumitrescu, Wu, and Wang [2], and RS group (see Section 5.2.4) is the RS steganalysis of Fridrich *et al.* [4]

8 Acknowledgments

We thank the Brandeis Computer Science Department Security Reading Group, in particular Jon Sagotsky for suggesting this problem. We thank Marty Cohn for his encouragement, Tim Hickey for the suggested application, and Jason C. Wu and Neal Lakritz for providing images.

References

[1] Adobe Developer's Association. TIFF revision 6.0, 1992. http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.

[2] Sorina Dumitrescu, Xiaolin Wu, and Zhe Wang. Detection of LSB steganography via sample pair analysis. In Fabien A. P. Petitcolas, editor, *Information Hiding*, volume 2578 of *Lecture Notes in Computer Science*, pages 355–372. Springer, 2002.

[3] Jessica Fridrich, Miroslav Goljan, and Rui Du. Steganalysis based on JPEG compatibility. In *Digital Watermarking and Data Hiding, SPIE Multimedia Systems and Applications IV*, pages 275–280, 2001.

[4] Jessica J. Fridrich, Miroslav Goljan, and Rui Du. Reliable detection of LSB steganography in color and grayscale images. In *Proc. of the ACM Workshop on Multimedia and Security*, pages 27–30, 2001.

[5] İsmail Avcıbaş, Nasir Memon, and Bülent Sankur. Steganalysis using image quality metrics. *IEEE Transactions on Image Processing*, 12(3):221–229, Feb 2003.

[6] David Kahn. The history of steganography. In Ross Anderson, editor, *Information Hiding: First International Workshop*, volume 1174 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 1996.

[7] C. Kurak and J. McHugh. A cautionary note on image downgrading. In *Computer Security Applications Conference*, pages 153–159, 1992.

[8] Kwangsoo Lee, Changho Jung, Sangjin Lee, and Jongin Lim. Color cube analysis for detection of LSB steganography in RGB color images. In *Information Security and Hiding Workshop in Intenational Conference on Computer Science and its Applicatons - ICCSA Part II*, volume 3481 of *Lecture Notes in Computer Science*, pages 537–546, 2005.

[9] Kwangsoo Lee, Changho Jung, Sangjin Lee, and Jongin Lim. New steganalysis methodology: LR cube analysis for the detection of LSB steganography. In Mauro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding: 7th International Conference*, volume 3727 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2005.

[10] Ira S. Moskowitz, Garth E. Longdon, and LiWu Chang. A new paradigm hidden in steganography. In *Paradigms Workshop*, 2000.

[11] National Institute of Standards and Technology. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications, 2001. http://csrc.nist.gov/rng/SP800-22b.pdf.

[12] Portable network graphics (PNG) specification (second edition), 2003. ISO/IEC 15948:2003 (E) http://www.w3.org/TR/2003/REC-PNG-20031110.

[13] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* Fourth Estate, London, 1999.