Steganography

Alex Toumazis

December 3, 2009

1 Introduction

This paper is based on "Attacks on Steganographic Systems" by Andreas Westfeld and Andreas Pfitzmann. This paper surveys and described several attack on common production steganographic utilities. Although the techniques and systems discussed are somewhat dated, they provide a good general introduction to the principles and weaknesses of steganography.

1.1 A Brief Introduction to Steganography

Steganography is the embedding of messages within an innocuous *cover work* in a way which can not be detected[1] by anyone without access to the appropriate steganographic key. Wikipedia calls steganography, incorrectly, a form of "security through obscurity". This is not true as a correctly designed, key-based system will resist attackers that know the details of the algorithm but not the key. *Steganalysis* is the study of attacking such systems, analagous to cryptanalysis of cryptographic systems.

A *threat model* consists of a attack scenarios we use to evaluate steganographic techniques. In this paper, the threat model is that of a *passive warden* — someone who can see and analyze the data but cannot alter it in an attempt to destroy the hidden message. Therefore, the techniques aim to *hide* the existence of a message, without worrying too much about *robustness*.

2 Method

2.1 Steganography Techniques

This paper discusses several simple steganographic algorithms used by widely available information-hiding utilities. These techniques have in common the goal of embedding data in perceptually indistinguishable parts of an image.

[1] Note that this is a much stronger property than encoding a message in a way which cannot be read; cryptography does this but does not hide the presence of the message

2.1.1 EzStego

EzStego embeds data in GIF images, by altering the colors of pixels. To do this imperceptibly, it sorts the image's palette to minimize the perceptual distance between consecutive colors. This is achieved by finding the shortest-path between the palette colors within the RGB color space cube. Then, the message is embedded by altering pixels, in order, so that the least significant bits of the pixels' indexes into the sorted palette are the message bits.

2.1.2 Steganos

Steganos embeds data in the least significant bits of image files. It replaces this data completely in the cover image, regardless of message length. This is identical to the naive least significant bit steganography described in section 3.1 below.

2.1.3 JSTEG

JSTEG embeds each message bit into more than one bit of the cover image, to reduce the detectible effects of the embedding. This is achieved through transforming the image by dividing it into blocks and applying the discrete cosine transform, and then altering coefficients and transforming back. This results to each message bit affecting an entire block in the cover image, making it harder to detect the changes.

3 Experiment

3.1 LSB

3.1.1 Implementation

I implemented least-significant bit steganography in Matlab by replacing the green channel of a cover image with the message bitstream XORed with a pseudorandom key. The message I used was a 1-bit image of equal size to the cover image, but this technique can be used for messages of any length up to 3NM in a N by M color image. By using messages smaller than the image and spreading the message out in a simple way, for example skipping a key-determined pseudorandom number of pixels, some of the described attacks can be defeated. The results of such embedding is visually indistinguishable, as shown in Figure 1. Although simple, this technique, with some variation, is widely used; Steganos uses it, and EzStego is basically the extension of this technique to the GIF palette.

3.1.2 Visual Attacks

As mentioned previously, least-significant bits are not in fact pseudorandom. Therefore, changing these bits in an image to pseudorandom noise should be detectible. This can be shown by simply amplifying and examining the least-significant bits of the image, as shown in Figure 2. If embedding had been performed in all three color channels, the effect would be even more pronounced.

```
I = cover image
M = message
K = pseudorandom keystream
I = I − mod(I,2);
T=xor(M,K);
I=I+T
```

Figure 1: Pseudocode demonstrating simple LSB embedding

(a) Unmodified Image (b) Image with embedded data

Figure 2: Visual effect of least significant bit embedding

3.1.3 Chi-square attack

This attack can be automated using statistical techniques. As the embedded data is pseudorandom and the natural image data is not, a test for randomness should let us differentiate between a natural image and one with pseudorandom data embedded in it. The chi-squared statistic is a measure of fit between two discrete distributions, and is:

$$\chi^2 = \sum_{i=1}^{k} \frac{(n_i - n_i^*)^2}{n_i^*} \tag{1}$$

To apply it here, we measure the frequency of pixels of each value, then derive an expected value for each using the assumption that the least-significant bits are random; with this assumption, the expected value for f_{2k} is:

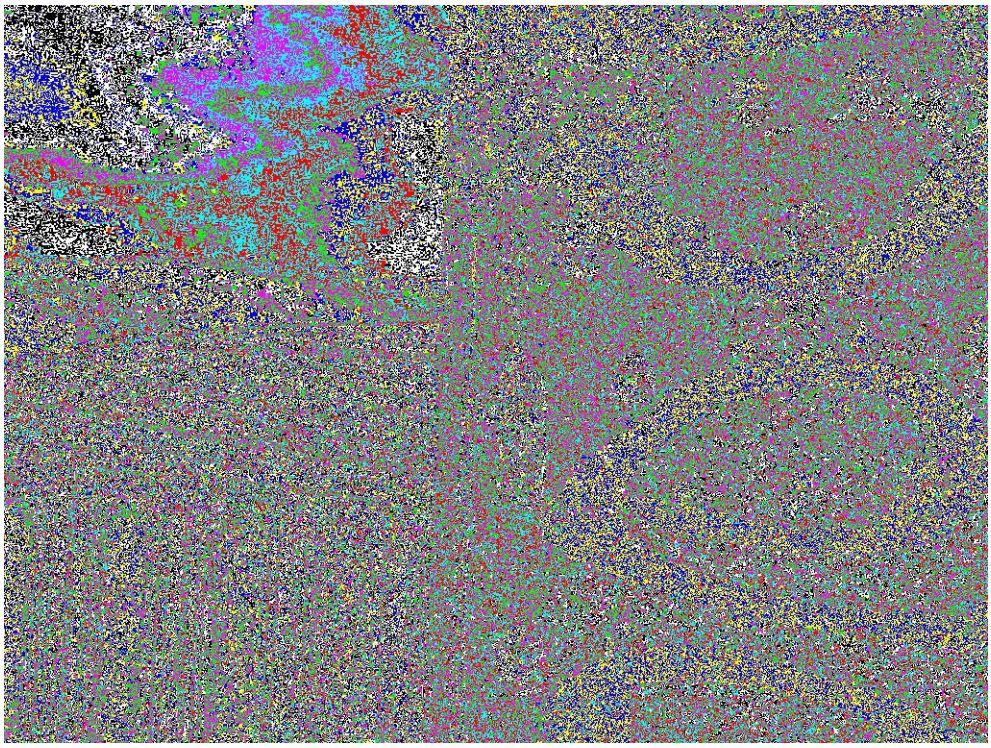
$$f_{2i}^* = \frac{f_k + f_{k+1}}{2} \tag{2}$$

We then a distribution with $i/2$ bins, each of which contains the frequency of the even bit values. The expected frequencies for each bin are calculated as shown above, and the χ^2 value is then calculated by:

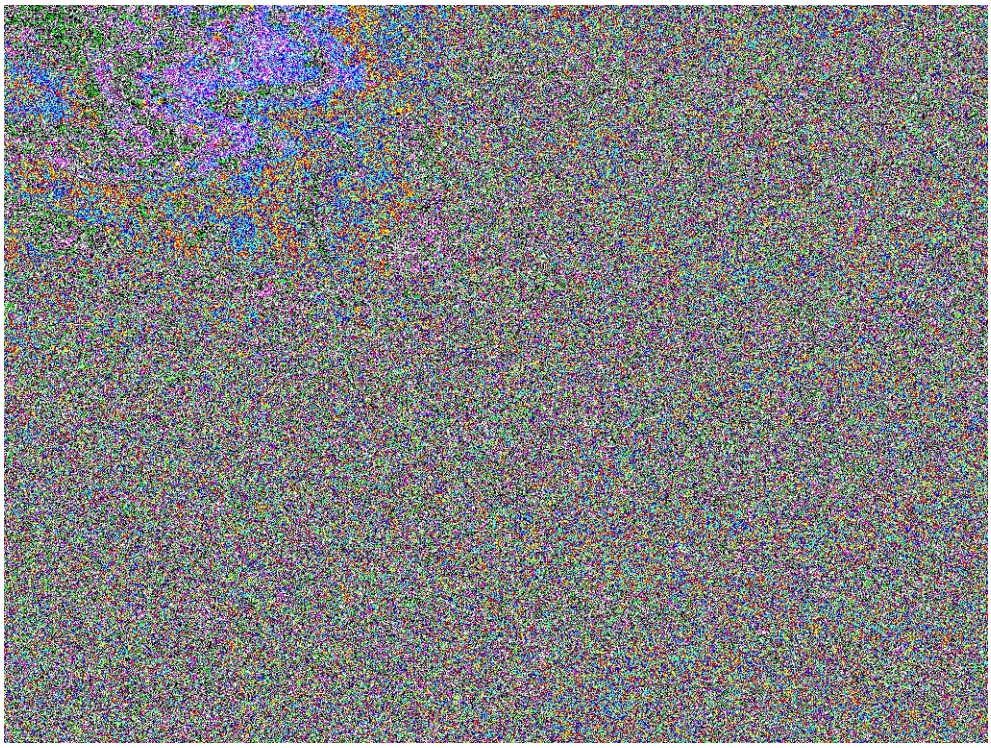
$$\chi^2 = \sum_{i=1}^{k/2} \frac{(f_{2i} - f_{2i}^*)^2}{f_{2i}^*} \tag{3}$$

To use this value to give a probability of embedding, we integrate to find the area under the probability density function:

$$p = 1 - \frac{1}{2^{\frac{k-2}{4}}\Gamma(\frac{k-2}{4})} \int_0^{\chi^2} e^{-\frac{x}{2}} x^{\frac{k-2}{4}-1} dx \tag{4}$$



(a) Unmodified Image



(b) Image with embedded data

Figure 3: Visual effect of least significant bit embedding: least-significant bits of each image

I implemented this as shown in Figure 4 and tested it on several images with varying lengths of embedded message. Figure 5 shows graphs of the probability of embedding against the cumulative number of rows being examined. This method clearly and unambiguously differentiates between modified and original images, and the case of partial embedding detects the point where the embedding stops.

3.2 JSTEG

3.2.1 Implementation

My JSTEG implementation consists of two nested parts. The outer part applies the DCT, quantizes the coefficients, and inverts the transform. In between quantization and the inversion, the inner stage embeds the message in the least significant bits of the coefficients. There are a couple of subtleties; the algorithm avoids embedding data in coefficients whose value is zero, as this usually is due to quantization, which means that values other than zero will raise suspicion, and recompression (with the same quantization table) will destroy the message bit. JSTEG is more fragile than the previous technique described for this reason, as a change leading to non-zero coefficients becoming zero will lead to the output bitstream becoming out of sync, while with simple LSB embedding changes can just lead to flipped bits in the output.

3.2.2 Visual Attacks

Images with messages embedded using JSTEG are not susceptible to the simple visual attacks demonstrated above. This is because the changes to the coefficients do not simply alter bits in the final image in a predictable manner, but instead affect entire blocks of the image in ways that cannot easily be perceived, even when examining the least-significant bits of the image. Figure 8 shows the least-significant bits of a cover image, the same image passed through the DCT/inverse DCT algorithm without having any of its coefficients altered (i.e. there is no embedded message), and the image with a message embedded. Although there is a visible difference between the latter two images, it is not apparent that one is more random than the other, and in general human observers cannot determine whether pseudorandom data has been inserted into the DCT coefficients.

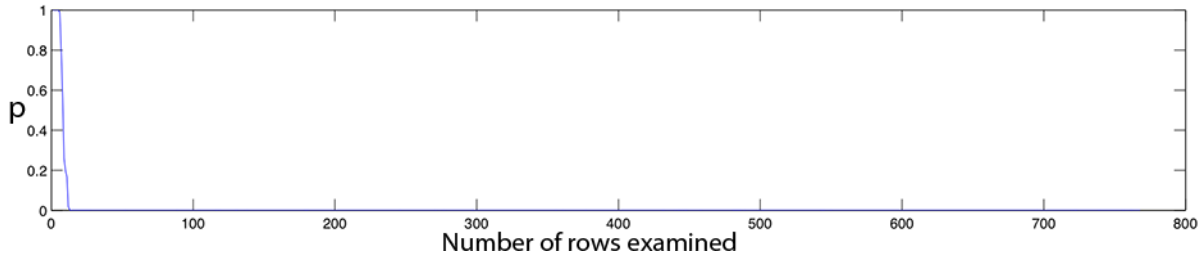
3.2.3 Chi-square attack

Although the least-significant bits of the image do not exhibit pseudorandom properties, the DCT coefficients into which message bits were hidden do. Therefore, if we know the transform performed — in this case the discrete cosine transform with an 8×8 block size — we can recreate the coefficients and analyze them to determine if they are drawn from a pseudorandom distribution. The embedding algorithm is reversed to extract the least-significant bit of each coefficient which is not zero or maximum, and then the chi square statistic and embedding probability are computed as in the simpler attack described above.

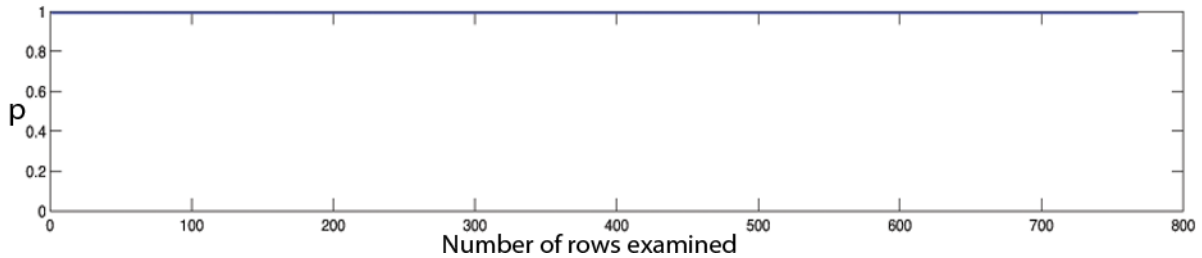
Figure 9 shows the resulting graphs; (a) shows the original image, (b) shows the probability graph resulting from applying the algorithm without a message, i.e. performing the transform and then inverting it, to show that this doe not cause false positives, and (c) shows the graph of an image with an embedded message.

```
G = imread('image.tif');
Bins(256)=0;
tsize=size(G);
Chis(tsize(1))=0;
P(tsize(1))=0;
for i=1:tsize(1),
%count pixel frequencies
    for j=1:tsize(2),
    Bins(G(i,j)+1)=Bins(G(i,j)+1)+1;
    end
    %calculate chi squared
    chi2=0;
    for k=1:127
    nexp=(Bins(2*k-1)+Bins(2*k))/2;
    if(nexp>5)
    chi2=chi2+((Bins(2*k)-nexp)^2)/nexp;
    end
    end
    Chis(i)=chi2;
end
%calculate P
xu=chi2;
r=127;
rOver2 = 0.5*r;
gammaval = gamma(rOver2);
F = @(x) x.^(rOver2-1).*exp(-0.5*x)/gamma(rOver2)/2^rOver2;
if xu == inf
    prob = 1
else
    prob = quad(F,0,xu);
end
P(i)=1-prob;
end
%plot graph
plot(P);
```

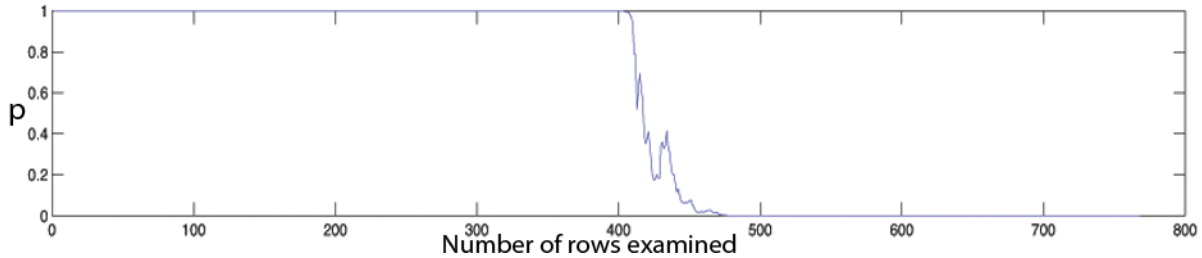
Figure 4: Matlab code for computing probability of LSB consisting of pseudorandom data



(a) Unmodified Image



(b) Image with message embedded

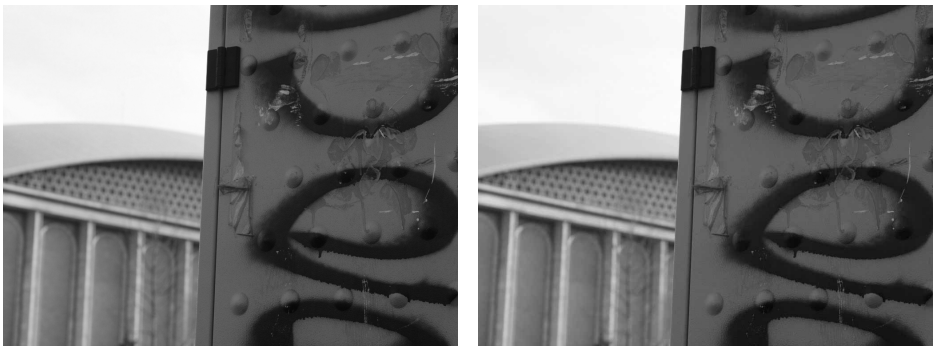


(c) Message embedded in first half of image

Figure 5: LSB: Cumulative probabilities of embedding, derived from χ^2

```
I = cover image
M = message
K = pseudorandom keystream
T = DCT(I, blocksize)
quantize(T, quantization_table)
progress=0
for each element i of T:
        if(i!=0 && i != max_i)
                setLSB(i, M(progress))
                progress++
        end
        if(progress>size(M)) break
end
I=inverseDCT(T, blocksize)
```

Figure 6: Pseudocode demonstrating JSTEG embedding



(a) Unmodified Image (b) Image with embedded data

Figure 7: Visual effect of JSTEG embedding

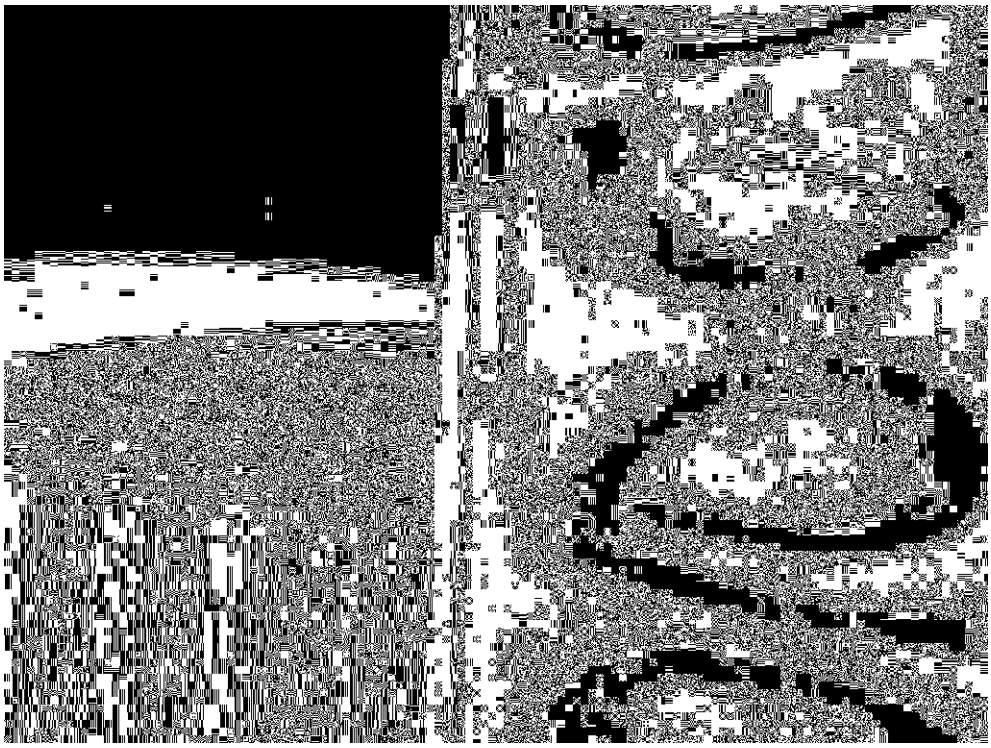
4 Further results

The results I've presented here mirror several of the results shown in the paper. It also dealt with EzStego, which acts on GIF images in a similar way to the least significant bit technique described above, with an additional step of sorting the palette by finding the shortest path through it in the RGB cube. The LSBs then used for message encoding are the LSBs of the index into this sorted palette. By recreating this palette and the indexes, both the visual and statistical attacks described in Section 3.1 can be applied.

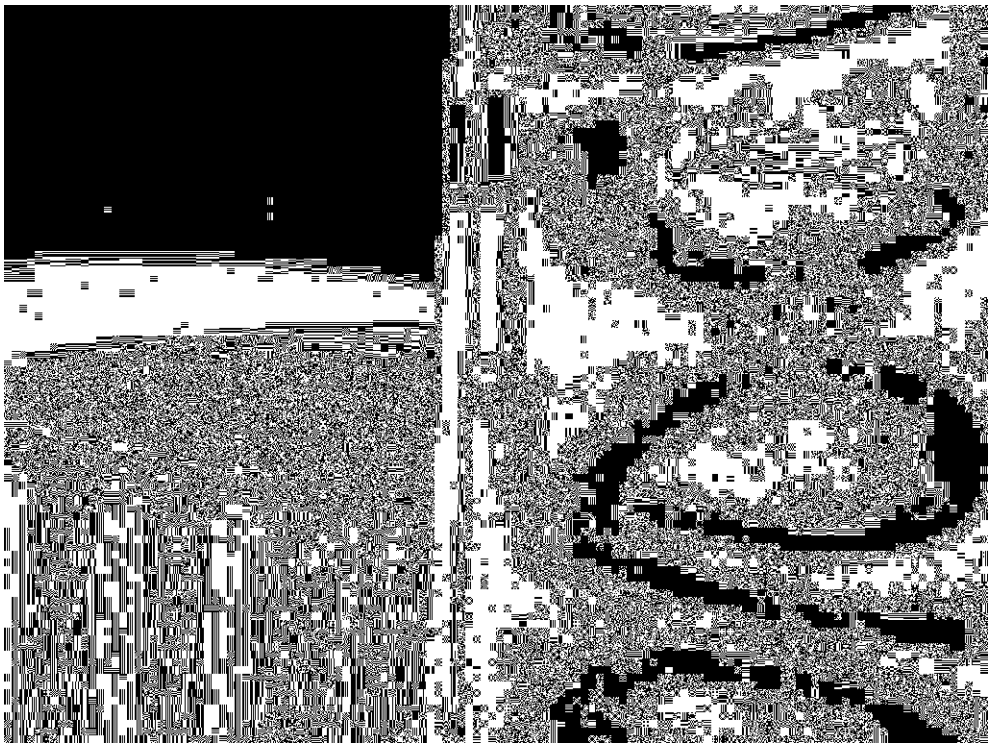
S-Tools is also discussed in the paper. This technique spreads the message across the entire image, and with small message sizes relative to image size the technique foils the simple visual and statistical tests. If the algorithm by which the pixels selected to be used is known, the same attacks should work, subject to the caveat that the message must be long enough for statistical analysis to be applied.

5 Conclusion

What can we take from these findings? Well, the simple point is that steganography is not easy; with knowledge of the algorithm/method (which we should always assume), it is very hard to hide messages in an undetectable way. This difficulty increases with the size of the message and the desired robustness of the scheme — a single bit could be hidden trivially (and not robustly) by changing a random LSB of the image to alter the parity of the image's bits, but once we want to encode enough pseudorandom data to make a statistical attack possible, things swiftly become more difficult. The flaw in the systems discussed is that they assumed certain parts of an image (either least significant bits of LSBs or DCT coefficients) were pseudorandom when they in fact are not. A possible approach to future techniques is to investigate ways of finding pseudorandom data in cover works, possibly by applying focussed tests such as the chi-square test, and inserting information in those parts of the image. Unfortunately, the amount of such data in most cover works is likely to be small, as natural data tends to not be truly random, and good compression schemes will destroy such pseudorandom data, as it carries no important perceptual information that cannot be recreated. Even if, for example, we hide data in the thermal noise in a digital photo, this may change or destroy properties of the sensor fingerprint, and by examining other images from the same camera, the modification may be detected.

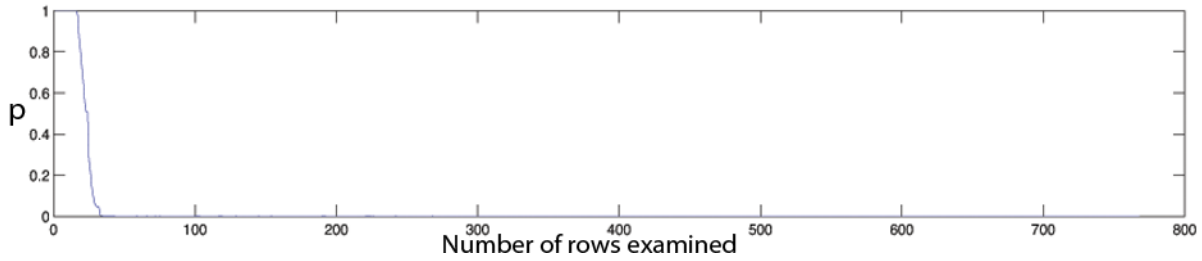


(a) Image after DCT/inverse DCT application, no message embedded

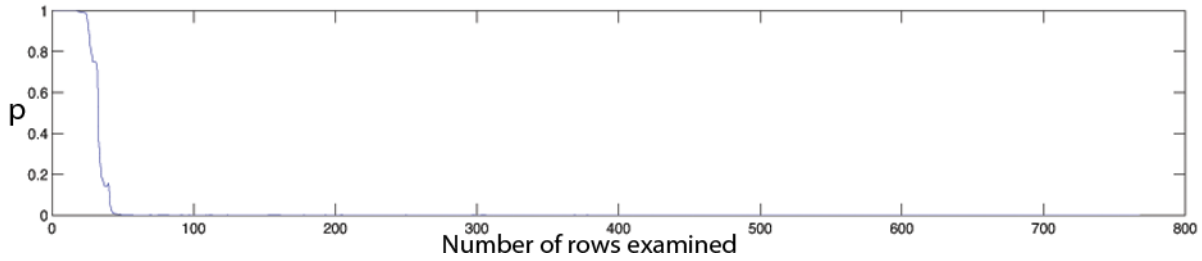


(b) Image with embedded message

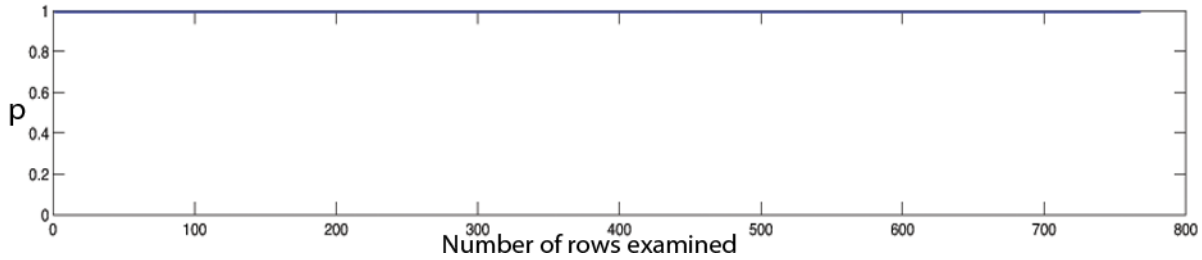
Figure 8: JSTEG: Least significant bits



(a) Unmodified Image



(b) Image after DCT/inverse DCT application, no message embedded



(c) Image with embedded message

Figure 9: JSTEG: Cumulative probabilities of embedding, derived from χ^2