

## A Java Application for Data Hiding in Image Files: Design Choices

by Jessica Codr under guidance of Dr. Raj Jain

In the associated paper “Unseen: An Overview of Steganography and Presentation of an Associated Java Application,” I give a brief overview of a Java Application I am developing for embedding secret messages within image files. Here, I offer a bit more in-depth discussion of the reasons for my technique choices and the current capabilities of this application.

First of all, my steganography program uses a form of basic LSB altering. This form of steganography is believed to be weak and easily breakable, but I chose to use it for two main reasons:

1. It is simple so I can easily do numerous experiments related to it.
2. I wanted to test whether I could strengthen this weak technique by adding more randomness.

The method itself does still encode data by flipping LSBs, but I hope that by adding randomness to images and to how data is encoded, I can make the cover images and stego-images seem more similar.

There are several types of “randomness” that I employ in my application. First of all, the exact pixels altered to encode each letter are semi-random. When embedding a message within an image, the first step is to count the number of letters in the message and divide the total number of rows of pixels in the image by that number. For example, if the image is 1000 pixels tall and I want to embed a message 100 letters in length, I divide 1000 by 100 to get 10 rows per letter. This is the number of rows of pixels that could be associated with each letter. Next, I take the number of pixels the image is wide and divide by the number of letters in the alphabet (26). So if the image is 780 pixels wide, I divide by 26 to get 30 columns per letter of the alphabet.

By grouping these sets of (10 in the example) rows and (30 in the example) columns together, I get a grid of sets of pixels within the image. Each of these subgrids corresponds to an encoding of a different letter of the message. The groups of rows correspond to the index of the letter in the message (eg, first letter, second letter, etc) and the groups of columns correspond to the actual letter encoded. So, for example, to encode the third letter as the letter “b” I would look at the third set of rows (rows 20-29 in our example) and the second set of columns (columns 26-51 in our example). Now to actually encode that the third letter in the message is “b”, I take a random bit from this subgrid (from rows 20-29 and columns 26-51 in our example) and flip it. See table 1 below for a visual representation of how subgrids map to different letters in the message for part of the image.

Table 1: Encoding Locations for Part of an Image File/Grid

IMAGE GRID	Cols 0-25	Cols 26-51	Cols 52-77	Cols 78-103	Cols 104-129
Row 0-9	Letter 1 = a	Letter 1 = b	Letter 1 = c	Letter 1 = d	Letter 1 = e
Row 10-19	Letter 2 = a	Letter 2 = b	Letter 2 = c	Letter 2 = d	Letter 2 = e
Row 20-29	Letter 3 = a	Letter 3 = b	Letter 3 = c	Letter 3 = d	Letter 3 = e
Row 30-39	Letter 4 = a	Letter 4 = b	Letter 4 = c	Letter 4 = d	Letter 4 = e
Row 40-49	Letter 5 = a	Letter 5 = b	Letter 5 = c	Letter 5 = d	Letter 5 = e
Row 50-59	Letter 6 = a	Letter 6 = b	Letter 6 = c	Letter 6 = d	Letter 6 = e

Now someone decoding the message by comparing the altered image against the original can easily determine the message even with this randomness since the letters appear in the message in the same order we find altered bits going down the rows and the decoder knows which groups of columns correspond to each letter. However, with the added randomness, it may be more difficult for someone who lacks the original image to detect something is wrong with the bit patterns of the modified image. Additionally, by spreading the letters out across all the rows of the image, I reduce the chances of steganalysis detecting something wrong with a small portion of the image.

This method does have a rather low data capacity (at most the number of rows of the image), but this also increases its strength against steganalysis since there are fewer alterations to be detected.

In addition to the randomness built directly into the data embedding, my program offers options for adding random noise to the image before embedding data into it. The noise is added by selecting random pixels and flipping their LSB before the secret data is even embedded (and then using this altered cover as the new cover). The hope here is to create a cover with a bit pattern that is random in such a way that it is then difficult to tell covers apart from stego-objects, that is, covers that might seem themselves like stego-objects and thus confuse steganalysis tactics.

Taking this added randomness even one step further, I implemented the ability for my program to generate its own random images to be used for covers. A fully random image presents no patterns in its least significant bits, and so it is (nearly) impossible to reliably detect alterations in bit patterns caused by my steganography method since there are no patterns to begin with.

The problem with this is that the complete lack of a pattern in the cover image may itself arouse suspicion of an outside observer, as would the fact that communicators are sending images to one another that are composed of entirely random pixels. As a counter to this, I modified my random image generator to generate images with pixels that are similar to those surrounding them. The result (an example of which is seen in section 5.2 of “Unseen: An Overview of Secret Communications and Presentation of an Associated Java Application”) is a fairly random image that still displays visual patterns and thus could potentially pass as a form of modern art, which is less suspicious than a fully random image.

The goal of all of this is to generate both covers and stego-objects that are statistically very similar. Some specific results are presented in the associated paper (“Unseen”). I found overall that it is very possible to generate cover/stego pairs that are almost identical statistically, thus making it nearly impossible for steganalysis to identify a single image as either cover or stego. Not every random image will have this property, but due to the nature of randomness, as large number are expected to.

Future work on this application could include refining the random image generator, exploring different ways to add randomness to images and to the embedding of messages within images, and altering images in other (perhaps larger) ways, such as flipping or shifting pixels. Though altering pixel locations or groups of pixels may cause visible differences in the image, these will only be noticeable to those who have both the cover and stego-object, and may make it more difficult for a computer viewing only one image to determine whether that image is a cover or a stego-image, as can be seen from statistical analysis of runs of the program presented in “Unseen: An Overview of Steganography and Presentation of an Associated Java Application.”