

# Exploring Steganography: Seeing the Unseen

Steganography is an ancient art of hiding information. Digital technology gives us new ways to apply steganographic techniques, including one of the most intriguing—that of hiding information in digital images.



Neil F. Johnson  
Sushil Jajodia  
George Mason  
University

Steganography is the art of hiding information in ways that prevent the detection of hidden messages. *Steganography*, derived from Greek, literally means “covered writing.” It includes a vast array of secret communications methods that conceal the message’s very existence. These methods include invisible inks, microdots, character arrangement, digital signatures, covert channels, and spread spectrum communications.

Steganography and cryptography are cousins in the spycraft family. Cryptography scrambles a message so it cannot be understood. Steganography hides the message so it cannot be seen. A message in ciphertext, for instance, might arouse suspicion on the part of the recipient while an “invisible” message created with steganographic methods will not.

In this article we discuss image files and how to hide information in them, and we discuss results obtained from evaluating available steganographic software. For a brief look at how steganography evolved, see the “Steganography: Some History” sidebar.

## IMAGE FILES

To a computer, an *image* is an array of numbers that represent light intensities at various points (pixels). These pixels make up the image’s *raster data*. A common image size is  $640 \times 480$  pixels and 256 colors (or 8 bits per pixel). Such an image could contain about 300 kilobits of data.

Digital images are typically stored in either 24-bit or 8-bit files. A 24-bit image provides the most space for hiding information; however, it can be quite large

(with the exception of JPEG images). All color variations for the pixels are derived from three primary colors: red, green, and blue. Each primary color is represented by 1 byte; 24-bit images use 3 bytes per pixel to represent a color value. These 3 bytes can be represented as hexadecimal, decimal, and binary values. In many Web pages, the background color is represented by a six-digit hexadecimal number—actually three pairs representing red, green, and blue. A white background would have the value FFFFFFFF: 100 percent red (FF), 100 percent green (FF), and 100 percent blue (FF). Its decimal value is 255, 255, 255, and its binary value is 11111111, 11111111, 11111111, which are the three bytes making up white.

This definition of a white background is analogous to the color definition of a single pixel in an image. Pixel representation contributes to file size. For example, suppose we have a 24-bit image 1,024 pixels wide by 768 pixels high—a common resolution for high-resolution graphics. Such an image has more than two million pixels, each having such a definition, which would produce a file exceeding 2 Mbytes. Because such 24-bit images are still relatively uncommon on the Internet, their size would attract attention during transmission. File compression would thus be beneficial, if not necessary, to transmit such a file.

## File compression

Two kinds of compression are *lossless* and *lossy*.<sup>1</sup> Both methods save storage space but have different results, interfering with the hidden information, when the information is uncompressed. Lossless compression lets us reconstruct the original message exactly;

therefore, it is preferred when the original information must remain intact (as with steganographic images). Lossless compression is typical of images saved as GIF (Graphic Interchange Format) and 8-bit BMP (a Microsoft Windows and OS/2 bitmap file).

Lossy compression, on the other hand, saves space but may not maintain the original image's integrity. This method typifies images saved as JPEG (Joint Photographic Experts Group). Due to the lossy compression algorithm, which we discuss later, the JPEG formats provide close approximations to high-quality digital photographs but not an exact duplicate. Hence the term "lossy" compression.

### Embedding data

Embedding data, which is to be hidden, into an image requires two files. The first is the innocent-looking image that will hold the hidden information, called the *cover image*. The second file is the message—the

information to be hidden. A message may be plain text, ciphertext, other images, or anything that can be embedded in a bit stream. When combined, the cover image and the embedded message make a *stego-image*.<sup>2</sup> A stego-key (a type of password) may also be used to hide, then later decode, the message.

Most steganography software neither supports nor recommends using JPEG images, but recommends instead the use of lossless 24-bit images such as BMP. The next-best alternative to 24-bit images is 256-color or gray-scale images. The most common of these found on the Internet are GIF files.

In 8-bit color images such as GIF files, each pixel is represented as a single byte, and each pixel merely points to a color index table (a palette) with 256 possible colors. The pixel's value, then, is between 0 and 255. The software simply paints the indicated color on the screen at the selected pixel position. Figure 1a, a red palette, illustrates subtle changes in color varia-

## Steganography: Some History

Throughout history, people have hidden information by a multitude of methods and variations.<sup>1,2</sup> For example, ancient Greeks wrote text on wax-covered tablets. To pass a hidden message, a person would scrape wax off a tablet, write a message on the underlying wood and again cover the tablet with wax to make it appear blank and unused. Another ingenious method was to shave the head of a messenger and tattoo a message or image on the messenger's head. After the hair grew back, the message would be undetected until the head was shaved again.

Invisible inks offered a common form of invisible writing. Early in World War II, steganographic technology consisted almost exclusively of these inks.<sup>1</sup> With invisible ink, a seemingly innocent letter could contain a very different message written between the lines.<sup>3</sup>

Documents themselves can hide information: document text can conceal a hidden message through the use of null ciphers (unencrypted messages), which camouflage the real message in an innocent-sounding missive. Open coded messages, which are plain text passages, "sound" innocent because they purport to be about ordinary occurrences. Because many open-coded messages don't seem to be cause for suspicion, and there-

fore "sound" normal and innocent, the suspect communications can be detected by mail filters while "innocent" messages are allowed to flow through.<sup>1</sup> For example, the following null-cipher message was actually sent by a German spy in WWII<sup>1</sup>:

Apparently neutral's protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetable oils.

Decoding this message by extracting the second letter in each word reveals the following, hidden message:

Pershing sails from NY June 1.

Document layout may also reveal information. Documents can be marked and identified by modulating the position of lines and words.<sup>4</sup>

Message detection improved with the development of new technologies that could pass more information and be even less conspicuous. The Germans developed microdot technology, which FBI Director J. Edgar Hoover referred to as "the enemy's masterpiece of espionage."<sup>1</sup> Microdots are photographs the size of a printed period having the clarity of standard-sized type-

written pages, which permits the transmission of large amounts of data, including drawings and photographs.<sup>1</sup>

With every discovery of a message hidden with an existing application, a new steganographic application is being devised. Old methods are given new twists. While drawings have often been used to conceal or reveal information, computer technology has, in fact, sparked a revolution in such methods for hiding messages.

Space limitations prevent further discussion here. For more information on techniques for hiding information, see Peter Wayner's *Disappearing Cryptography*.<sup>5</sup>

### References

1. D. Kahn, *The Codebreakers*, Macmillan, New York, 1967.
2. B. Norman, *Secret Warfare*, Acropolis Books, Washington, D.C., 1973.
3. H.S. Zim, *Codes and Secret Writing*, William Morrow, New York, 1948.
4. J. Brassilet et al., "Document Marking and Identification using Both Line and Word Shifting," *Proc. Infocom95*, IEEE CS Press, Los Alamitos, Calif., 1995.
5. P. Wayner, *Disappearing Cryptography*, AP Professional, Chestnut Hill, Mass., 1996.

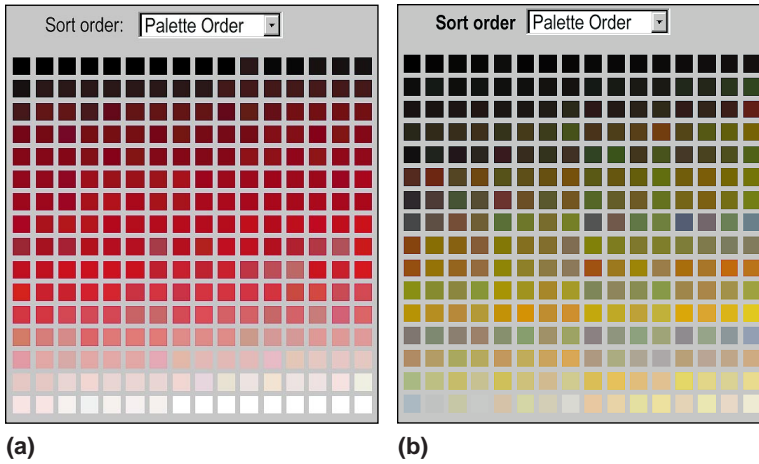


Figure 1. Representative color palettes. (a) A 256-color red palette and (b) a 256-color Renoir palette. The Renoir palette is so named because it comes from a 256-color version of Pierre-Auguste Renoir's "Le Moulin de la Galette."

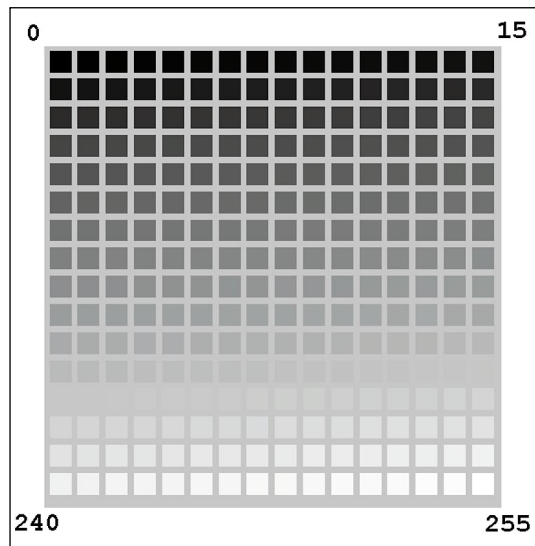
tions: visually differentiating between many of these colors is difficult. Figure 1b shows subtle color changes as well as those that seem drastic.

Many steganography experts recommend using images featuring 256 shades of gray.<sup>3</sup> Gray-scale images are preferred because the shades change very gradually from byte to byte, and the less the value changes between palette entries, the better they can hide information. Figure 2 shows a gray-scale palette of 256 shades. Some images are 4-bit, created with 16 shades of gray; obviously these images offer many fewer variations.

While gray-scale images may render the best results for steganography, images with subtle color variations are also highly effective, as Figure 1 showed.

When considering an image in which to hide information, you must consider the image as well as the palette. Obviously, an image with large areas of solid

Figure 2. Representative gray-scale palette of 256 shades.



colors is a poor choice, as variances created from the embedded message will be noticeable in the solid areas. We will see that Figure 1b, the palette for the Renoir cover image, makes a very good cover for holding data.

Once you've selected a cover image, you must decide on a technique to hide the information you want to embed.

### CONCEALMENT IN DIGITAL IMAGES

Information can be hidden many different ways in images. To hide information, straight message insertion may encode every bit of information in the image or selectively embed the message in "noisy" areas that draw less attention—those areas where there is a great deal of natural color variation. The message may also be scattered randomly throughout the image. Redundant pattern encoding "wallpapers" the cover image with the message.

A number of ways exist to hide information in digital images. Common approaches include

- least significant bit insertion,
- masking and filtering, and
- algorithms and transformations.

Each of these techniques can be applied, with varying degrees of success, to different image files.

#### Least significant bit insertion

*Least significant bit (LSB) insertion*<sup>4</sup> is a common, simple approach to embedding information in a cover file. Unfortunately, it is vulnerable to even a slight image manipulation. Converting an image from a format like GIF or BMP, which reconstructs the original message exactly (*lossless compression*) to a JPEG, which does not (*lossy compression*), and then back could destroy the information hidden in the LSBs.

**24-bit images.** To hide an image in the LSBs of each byte of a 24-bit image, you can store 3 bits in each pixel. A  $1,024 \times 768$  image has the potential to hide a total of 2,359,296 bits (294,912 bytes) of information. If you compress the message to be hidden before you embed it, you can hide a large amount of information. To the human eye, the resulting stego-image will look identical to the cover image.

For example, the letter A can be hidden in three pixels (assuming no compression). The original raster data for 3 pixels (9 bytes) may be

```
(00100111 11101001 11001000)
(00100111 11001000 11101001)
(11001000 00100111 11101001)
```

The binary value for A is 10000011. Inserting the binary value for A in the three pixels would result in

```
(00100111 11101000 11001000)
(00100110 11001000 11101000)
(11001000 00100111 11101001)
```

The underlined bits are the only three actually changed in the 8 bytes used. On average, LSB requires that only half the bits in an image be changed. You can hide data in the least and second least significant bits and still the human eye would not be able to discern it.

**8-bit images.** 8-bit images are not as forgiving to LSB manipulation because of color limitations. Steganography software authors have devised several approaches—some more successful than others—to hide information in 8-bit images. First, the cover image must be more carefully selected so that the stego-image will not broadcast the existence of an embedded message.

When information is inserted into the LSBs of the raster data, the pointers to the color entries in the palette are changed. In an abbreviated example, a simple four-color palette of white, red, blue, and green has corresponding palette position entries of 0 (00), 1 (01), 2 (10), and 3 (11), respectively. The raster values of four adjacent pixels of white, white, blue, and blue are 00 00 10 10. Hiding the binary value 1010 for the number 10 changes the raster data to 01 00 11 10, which is red, white, green, blue. These gross changes in the image are visible and clearly highlight the weakness of using 8-bit images. On the other hand, there is little visible difference noticed between adjacent gray values, as Figure 2 shows.

**Implementing LSB.** Steganography software processes LSB insertion to make the hidden information less detectable. For example, the EzStego tool arranges the palette to reduce the occurrence of adjacent index colors that contrast too much—before it inserts the message. This approach works quite well in gray-scale images and may work well in images with related colors.

S-Tools, another steganography tool, takes a different approach by closely approximating the cover image, which may mean radical palette changes. As with 24-bit images, changing the pixels' LSB may create new colors. (New colors may not be added to an 8-bit image due to the palette limit.) Instead, S-Tools reduces the number of colors while maintaining the image quality, so that the LSB changes do not drastically change color values.

For example, eight color values are required for each color if values 000 through 111 are to be stored. Reducing the number of unique colors to 32 ensures that these values can be used and that the number of colors will not exceed 256 ( $256/8 = 32$ ). Each of the 32 unique colors in the palette may be expanded to eight colors having LSB values of the red, green, blue (RGB) triples ranging from 000 to 111. This results in multiple colors in the palette that look the same



Figure 3. Image “painted” with the watermark: “Invisible Man” © 1997, Neil F. Johnson. Traditional steganography conceals information; watermarks extend information and become an attribute of the cover image.

visually but that may vary by one bit.<sup>5</sup>

These tools take a similar approach with gray-scale images. However, the resulting stego-images as applied with S-Tools are no longer gray-scale. Instead of simply going with adjacent colors as EzStego does, S-Tools manipulates the palette to produce colors that have a difference of one bit. For example, in a normal gray-scale image, white will move to black with the following RGB triples

```
(255 255 255), (254 254 254), . . . ,
(1 1 1), (0 0 0)
```

After processing with S-Tools, the value for white will be spread over a range of up to eight colors such as

```
(255 255 255), (255 255 254), and
(255 254 255)
```

Visually, the stego-image may look the same as the gray-scale cover image, but it has actually become an 8-bit color image.

### Masking and filtering

Masking and filtering techniques, usually restricted to 24-bit and gray-scale images, hide information by marking an image, in a manner similar to paper watermarks. Watermarking techniques may be applied without fear of image destruction due to lossy compression because they are more integrated into the image.

Visible watermarks are not steganography by definition. The difference is primarily one of intent. Traditional steganography *conceals* information; watermarks *extend* information and become an attribute of the cover image. Digital watermarks may include such information as copyright, ownership, or license, as shown in Figure 3. In steganography, the object of communication is the hidden message. In digital water-

Steganography's niche in security is to supplement cryptography, not replace it. If a hidden message is encrypted, it must also be decrypted if discovered, which provides another layer of protection.

marks, the object of communication is the cover.

To create the watermarked image in Figure 3, we increased the luminance of the masked area by 15 percent. If we were to change the luminance by a smaller percentage, the mask would be undetected by the human eye. Now we can use the watermarked image to hide plaintext or encoded information.

Masking is more robust than LSB insertion with respect to compression, cropping, and some image processing. Masking techniques embed information in significant areas so that the hidden message is more integral to the cover image than just hiding it in the "noise" level. This makes it more suitable than LSB with, for instance, lossy JPEG images.

**Algorithms and transformations.** LSB manipulation is a quick and easy way to hide information but is vulnerable to small changes resulting from image processing or lossy compression. Such compression is a key advantage that JPEG images have over other formats. High color quality images can be stored in relatively small files using JPEG compression methods; thus, JPEG images are becoming more abundant on the Internet.

One steganography tool that integrates the compression algorithm for hiding information is Jpeg-Jsteg. Jpeg-Jsteg creates a JPEG stego-image from the input of a message to be hidden and a lossless cover image. According to the Independent JPEG Group, the JPEG software we tested has been modified for 1-bit steganography in JFIF output files, which are composed of lossy and nonlossy sections. The software combines the message and the cover images using the JPEG algorithm to create lossy JPEG stego-images.

JPEG images use the discrete cosine transform to achieve compression. DCT<sup>6</sup> is a lossy compression transform because the cosine values cannot be calculated exactly, and repeated calculations using limited precision numbers introduce rounding errors into the final result. Variances between original data values and restored data values depend on the method used to calculate DCT. For details, see Wayne Brown and Barry Shepherd's book on graphic file formats.<sup>7</sup>

In addition to DCT, images can be processed with fast Fourier transformation and wavelet transformation.<sup>8</sup> Other image properties such as luminance can also be manipulated. Patchwork<sup>9</sup> and similar techniques use *redundant pattern encoding* or spread spectrum methods<sup>10</sup> to scatter hidden information throughout the cover images ("patchwork" is a method that marks image areas, or patches). These approaches may help protect against image processing such as cropping and rotating, and they hide information more thoroughly than by simple masking. They also support image manipulation more readily than tools that rely on LSB.

In using *redundant pattern encoding*, you must trade off message size against robustness. For example, a small message may be painted many times over an image as shown in Figure 3 so that if the stego-image is cropped, there is a high probability that the watermark can still be read. A large message may be embedded only once because it would occupy a much greater portion of the image area.

Other techniques *encrypt and scatter* the hidden data throughout an image. Scattering the message makes it appear more like noise. Proponents of this approach assume that even if the message bits are extracted, they will be useless without the algorithm and stego-key to decode them. For example, the White Noise Storm tool is based on spread spectrum technology and frequency hopping, which scatters the message throughout the image. Instead of having *x* channels of communication that are changed with a fixed formula and passkey, White Noise Storm spreads eight channels within a random number generated by the previous window size and data channel. Each channel represents 1 bit, so each image window holds 1 byte of information and many unused bits. These channels rotate, swap, and interlace among themselves to yield a different bit permutation. For instance, bit 1 might be swapped with bit 7, or both bits may rotate one position to the right. The rules for swapping are dictated by the stego-key and by the previous window's random data (similar to DES block encryption).

Scattering and encryption helps protect against hidden message extraction but not against message destruction through image processing. A scattered message in the image's LSBs is still as vulnerable to destruction from lossy compression and image processing as is a clear-text message inserted in the LSBs.

Steganography's niche in security is to supplement cryptography, not replace it. If a hidden message is encrypted, it must also be decrypted if discovered, which provides another layer of protection.

## EVALUATION EXAMPLES

To determine the limitations and flexibility of available software, we evaluated several steganographic packages. Here we discuss only three: StegoDos, White Noise Storm, and S-Tools for Windows. For details on other tools, see the sidebar "For More Information."

First, we selected message and cover files. In some cases, we had to alter the selected images to fit into the constraints of the software or had to use other cover images. In all, we tested 25 files as cover images. For purposes of this article, we discuss the results we obtained with only two message files and two cover image files.

The first message file contained this plain text:

Steganography is the art and science of communicating in a way which hides the existence of the

communication. In contrast to cryptography, where the “enemy” is allowed to detect, intercept and modify messages without being able to violate certain security premises guaranteed by a cryptosystem, the goal of steganography is to hide messages inside other “harmless” messages in a way that does not allow any “enemy” to even detect that there is a second secret message present [Markus Kuhn 1995-07-03].

The second message file was an image, the satellite image shown in Figure 4.

The cover images we used were Pierre-Auguste Renoir’s “Le Moulin de la Galette” as shown in Figure 5 and a Droeshout engraving of William Shakespeare as shown in Figure 6.

The image of Shakespeare is too small to contain the Airfield, but we embedded the text message without any image degradation. All the software we tested could handle the 518-byte plain-text message and embed it into the Shakespeare cover; however, some could not process the Renoir cover and Airfield images.

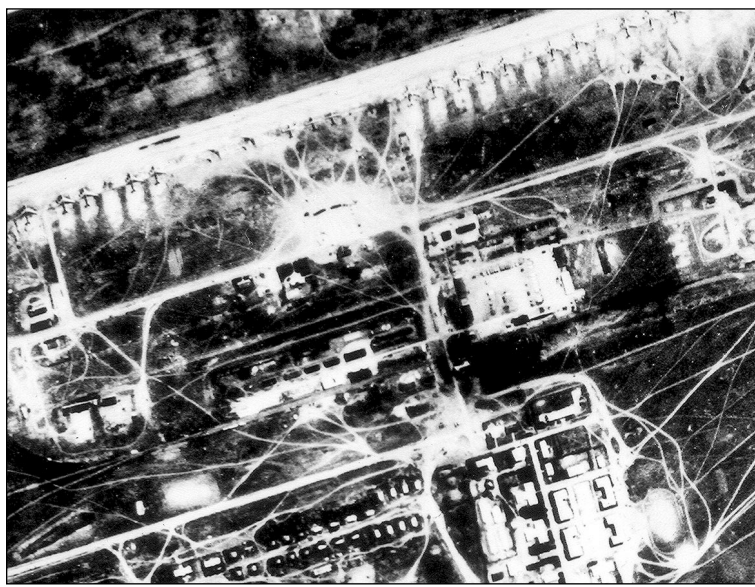


Figure 4. The satellite image we tested is of a major Soviet strategic bomber base (<http://edcwww.cr.usgs.gov/dclass>).

## For More Information

Additional readings, software, and resources are available at <http://isse.gmu.edu/~njohnson/Steganography> and <http://patriot.net/~johnson/Steganography>.

Other tools are being developed that take advantage of emerging technologies and the understanding of steganography. Steganography applications are available on the Internet that run on a variety of platforms, including DOS, Windows, OS/2, Mac and Unix. Stego for the Mac, EzStego for Java, and Stego Online are steganography tools by Romana Machado (<http://www.stego.com>). Stego is limited to Pict files. EzStego and Stego Online, written in Java, are limited to 8-bit GIF images. Other applications now being developed take advantage of broader band multimedia transmissions such as video and voice.

### Steganography tools: Comments

Henry Hastur has created two tools: Mandelsteg and Stealth. Mandelsteg generates Mandelbrot fractal images. If a file name is passed as a parameter, the file is hidden in the mandelbrot image. Mandelsteg does not manipulate any cover images other than the fractal images it creates ([\[sec.dsi.unimi.it/pub/security/crypt/code/\]\(http://sec.dsi.unimi.it/pub/security/crypt/code/\)\).](ftp://idea.</a></p></div><div data-bbox=)

Stealth (<http://des.ex.ac.uk/~aba/stealth/>) is not a steganographic program but is coupled with steganographic software on the Internet because it complements steganographic methods. Stealth is a filter that strips off the PGP header in a PGP-encrypted file, leaving the encrypted data. Why is this important? Applying steganography to an encrypted message is more secure than a plain text message. However, many encryption applications add header information to the encrypted message. This header information identifies the method used to encrypt the data. For example, if a cracker has identified hidden data in an image and has successfully extracted the encrypted message, a header for the encryption method would point the cracker in the right direction for additional cryptanalysis. But if the header is removed the cracker cannot determine the method for encryption. Some steganography programs provide this step in security, but others do not. Two that do are Ray Arachelian’s White Noise Storm (<ftp://ftp.csua.berkeley.edu/pub/cypherpunks/steganography/wns210.zip>) and Andy Brown’s S-Tools, at [<ftp://idea.sec.dsi.unimi.it/pub/security/crypt/code/s-tools4.zip> \(version 4\).](ftp://idea.sec.dsi.unimi.it/pub/security/crypt/code/s-tools3.</a></p></div><div data-bbox=)

Hide and Seek by Colin Maroney had some problems with minimum image sizes (320 × 480). In version 4.1 of the software (<ftp://ftp.csua.berkeley.edu/pub/cypherpunks/steganography/hdsk41b.zip>), if the image was smaller than the minimum, then the stego-image was padded with black space. If the cover image was larger, the stego-image was cropped to fit. In version 5.0, the same was true with minimum image sizes. If any image exceeded 1,024 × 768, an error message was returned. The Hide and Seek Version 1.0 for Windows 95 seems to have these issues resolved.

### Additional steganography tools

*StegoDos—Black Wolf’s Picture Encoder v0.90B*; public domain: <ftp://ftp.csua.berkeley.edu/pub/cypherpunks/steganography/stegodos.zip>.

*Jpeg-Jsteg V4*, <ftp://ftp.funet.fi/pub/crypt/steganography>.

*PictureMarc*, Digimarc, Portland, Ore., <http://www.digimarc.com>.

*SureSign*, Signum Technologies, Cheltenham, England, <http://www.signumtech.com>.



Figure 5. The Renoir cover file example (access <http://www.hs.port.ac.uk/wm/paint/auth/re noir/moulin-galette/>).

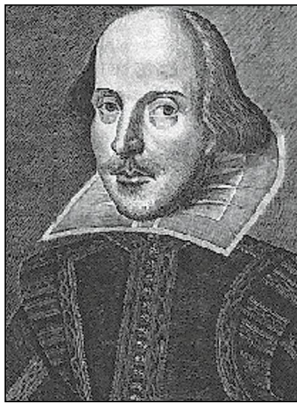


Figure 6. The Shakespeare cover file example (access <http://daphne.palomar.edu/shakespeare/life.htm>).

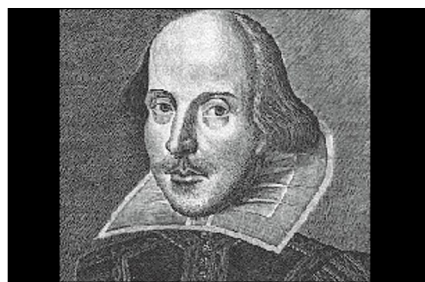


Figure 7. The result of embedding the text in the Shakespeare cover with StegoDos.

### StegoDos

StegoDos is public-domain software that works only with  $320 \times 200$ -pixel images with 256 colors. It required too much effort when measured against the results we obtained. Message encoding and decoding required many steps, and we had to track the original and modified files. Decoding the message required a third-party program.

Because of the size restrictions, the Airfield and the Renoir cover could not be used. We successfully hid the text message in the Shakespeare cover. Little distortion occurred within the Shakespeare image itself, but it was cropped and fitted into a  $320 \times 200$  pixel

image. Figure 7 illustrates this distortion when the text message is embedded.

StegoDos uses LSBs to hide messages, and it is less successful than other tools. It also appends an end-of-file character to the end of the message. But even with the EOF character, the message retrieved from the altered image is very likely to contain garbage at the end. The original text file was 518 bytes. The extracted file was around 8 Kbytes, and most of it was garbage.

### White Noise Storm

White Noise Storm is a very effective steganography application for DOS. We easily embedded the text message in the cover images and could detect no degradation. White Noise Storm could embed the Airfield into the Renoir cover image. However, there were problems with noise interfering with the image integrity, which severely shifted the image's palette, as shown in Figure 8.

White Noise Storm also includes an encryption routine to randomize the bits within an image. (White Noise's use of encryption with steganography is well integrated but beyond the scope of this article.) The software uses the LSB approach and applies this method to IBM Paintbrush (PCX) files. The software extracts the LSBs from the cover image and stores them in a file. The message is encrypted and applied to these bits to create a new set of LSBs. The modified bits are then injected into the cover image to create the new stego-image. The main disadvantage of applying White Noise Storm's encryption method to steganography is the loss of many bits that can be used to hold information. Relatively large files must be used to hold the same amount of information that other methods provide with much smaller cover images.

### S-Tools

S-Tools for Windows is the most versatile steganography tool of all that we tested. Version 3 includes programs that process GIF and BMP images and audio WAV files. S-Tools will even hide information in the "unused" areas on floppy diskettes. Version 4 incorporates image and sound file processing into a single program. In addition to supporting 24-bit images, S-Tools also includes encryption routines with many options.

Although S-Tools applies the LSB method to both images and audio files, we tested only images. The user interface and prompts are easy to use, and the online documentation is well developed. A useful feature is a status line displaying the largest message size that can be stored in an open cover file. This avoids wasting time attempting to store a message that is too large for a cover.

After the user hides the message, the software displays the new stego-image, and the user can toggle between the new and original images. At times the stego-image

may appear to be grossly distorted; however, after saving the stego-image it looks nearly identical to the original. The distorted appearance may be due to memory limitations or a bug in S-Tools. On occasion a saved image was actually corrupted and could not be read.

S-Tools provided the most impressive results of any package we tested because S-Tools maintained remarkable image integrity. We noticed no distortions when the text message was embedded in the Shakespeare cover. Similarly, S-Tools yielded the result closest to the original when we embedded the Airfield in the Renoir cover.

S-Tools provides many options for hiding and encrypting data. The best covers are made from 24-bit images and are processed quickly in S-Tools 4.0. Figure 9 shows the “after” image using a 24-bit BMP file. The original file contains 195,891 unique colors, while the resulting stego-image contains 312,340 unique colors. To the naked eye, these images are the same.

S-Tools handles 8-bit images, such as GIF files, a bit differently. Two options are available: boost it to a 24-bit image or color reduction. To boost it to a 24-bit image, the cover image is converted from an 8-bit image to a 24-bit image. The result is similar to that of Figure 9.

To apply color reduction instead, S-Tools creates an 8-bit stego-image by hiding a message in the 8-bit cover. Before “spreading” the message across the LSBs of the color levels in the image, S-Tools tries to reduce the number of colors in the image. The reduction process allows colors to be spread over several byte ranges so that shifts of the LSBs cause little impact in the image resolution. Visually differentiating between a 256-color image and one reduced to 32 colors is difficult, according to Andy Brown.<sup>5</sup>

Figure 10 illustrates the use of S-Tools on 8-bit images. First, we converted the original image to a GIF file. The conversion reduced the colors from 195,891 unique colors to 248 unique colors. In the process of hiding the Airfield image, the 8-bit Renoir cover image was reduced from 248 to 32 unique colors. Even with these apparently severe modifications, the resulting stego-image is impressively similar to the original Renoir cover image. The final GIF image has 256 unique colors versus 248 for the original converted GIF.

Figure 11 shows both the 256-color palette with 248 unique colors of original image converted to an 8-bit GIF image and the 256-color palette created from the S-Tools color reduction and hiding the Airfield image.

**S**teganography goes well beyond simply embedding text in an image. It also pertains to other media, including voice, text, binary files, and communication channels. For example, the plans of a top-secret project—device, aircraft, covert operation, or trade secret—can be embedded, using some



Figure 8. The Renoir cover file after the Airfield image was embedded with White Noise Storm. The palette has shifted severely.



Figure 9. The result of embedding the Airfield image in the Renoir cover with S-Tools.



Figure 10. Result of embedding the Airfield image in the 8-bit Renoir with S-Tools. In the process, the cover image was reduced from 248 to 32 unique colors.



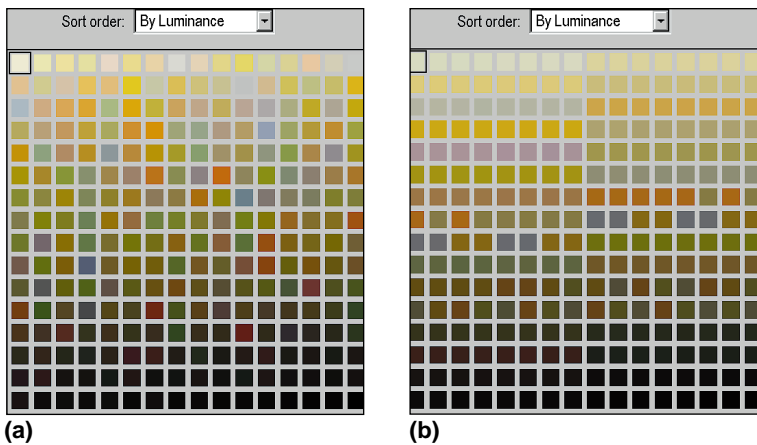


Figure 11. (a) Palette of the 8-bit Renoir cover image before embedding the Airfield image; (b) palette of the 8-bit Renoir cover image after embedding the Airfield image.

steganographic method, on an ordinary audio cassette tape. The alterations of the expected contents of the tape cannot be detected by human ears and probably not easily by digital means. Part of secrecy is of course in selecting the proper mechanisms.

Steganography by itself does not ensure secrecy, but neither does simple encryption. If these methods are combined, however, stronger encryption methods result. If a message is encrypted and then embedded in an image, video, or voice, it becomes even more secure. If an encrypted message is intercepted, the interceptor knows the text is an encrypted message. But with steganography, the interceptor may not know that a hidden message even exists.

Digital image steganography and its derivatives are growing in use and application. Where cryptography and strong encryption are outlawed, steganography can circumvent such policies to pass messages covertly. Commercial applications of steganography—digital watermarks and digital fingerprinting—are now in use to track the copyright and ownership of electronic media.

Steganography's ease of use and availability has law enforcement concerned with trafficking of illicit material via Web page images, audio, and other files. Researchers are investigating methods of message detection and testing the thresholds of current technology.

Development in covert communications and steganography will continue, as will research in building more robust digital watermarks that can survive image manipulation and attacks. The more information that is made available on the Internet, the more owners of such information need to protect themselves from theft and false representation. ❖

#### References

1. C. Kurak and J. McHugh, "A Cautionary Note On Image Downgrading," *Proc. IEEE Eighth Ann. Computer Security Applications Conf.*, IEEE Press, Piscataway, N.J., 1992, pp. 153-159.
2. B. Pfitzmann, "Information Hiding Terminology," *Proc. First Int'l Workshop Information Hiding, Lecture Notes in Computer Science No. 1, 174*, Springer-Verlag, Berlin, 1996, pp. 347-356.

3. T. Aura, "Invisible Communication," *EET 1995*, technical report, Helsinki Univ. of Technology, Finland, Nov. 1995; [http://deadlock.hut.fi/ste/ste\\_html.html](http://deadlock.hut.fi/ste/ste_html.html).
4. I. Cox et al., "A Secure, Robust Watermark for Multimedia," *Proc. First Int'l Workshop Information Hiding, Lecture Notes in Computer Science No. 1, 174*, Springer-Verlag, Berlin, 1996, pp. 185-206.
5. A. Brown, *S-Tools for Windows*, 1994, <ftp://idea.sec.dsi.unimi.it/pub/security/crypt/code/s-tools3.zip>.
6. E. Koch, J. Rindfrey, and J. Zhao, "Copyright Protection for Multimedia Data," *Proc. Int'l Conf. Digital Media and Electronic Publishing*, Leeds, UK, 1994.
7. W. Brown and B.J. Shepherd, *Graphics File Formats: Reference and Guide*, Manning Publications, Greenwich, Conn., 1995.
8. X-G. Xia, C.G. Boncelet, and G.R. Arce, "A Multiresolution Watermark for Digital Images," *IEEE Int'l Conf. Image Processing*, IEEE Press, Piscataway, N.J., 1997.
9. W. Bender et al., "Techniques for Data Hiding," *IBM Systems J.*, Vol. 35, Nos. 3 and 4, 1996, pp. 313-336.
10. I.J. Cox et al., "Secure Spread Spectrum Watermarking for Multimedia," Tech. Report 95-10, NEC Research Inst., Princeton, N.J., 1995.

*Neil F. Johnson is a research associate for the Center for Secure Information Systems (<http://isse.gmu.edu/~csis>) in the School for Information Technology Engineering at George Mason University and a PhD candidate in information technology. His current research interest is steganography. He received an MS in information systems from George Mason University and a BBA in computer information systems from James Madison University.*

*Sushil Jajodia is director of the Center for Secure Information Systems and professor of Information and Software Systems Engineering at George Mason University. His research interests include information security, temporal databases, and replicated databases. Jajodia received a PhD in mathematics from the University of Oregon, Eugene. He is the founding co-editor-in-chief of the Journal of Computer Security, a member of the editorial boards of IEEE Concurrency and International Journal of Cooperative Information Systems, and a contributing editor of Computer & Communication Security Reviews. Jajodia is a senior member of the IEEE and a member of the IEEE Computer Society and the ACM.*

*Contact Johnson and Jajodia at the Center for Secure Information Systems, Dept. of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444; {johnson, jajodia}@gmu.edu.*