Global Information Assurance Certification Paper

Steganography

Where the information is hidden
and
how it´s done

by Michael Meister
03.06.2003

Abstract

As information hiding has become more and more popular, many programs have emerged on the internet dealing with steganography. This document describes the structure of the hostfiles and the placement of the data that the user wants to hide.
The necessary tools and how to use them will be covered in this document. Countermeasures removing the hidden information from the hostfile will be briefly explained. The description of spreading and encrypting, before hiding the message within the file is beyond the scope of this document, but is done by most of the tools anyway.

History

The word steganography is derived from greek and means "coverted writing", from stegein, which means to cover. It has the same root as in stegosaur, a quadrupedal, herbivorous ornithiscian dinosaur of jurassic films and early cretaceous times, well known for being quite 'covered' through an armor of triangular bony plates on its back spine.
Steganography is the art (and science) of communicating by hiding the existence of communication; this is in contrast to cryptography. Ideally, your enemies, or the one you are fighting against, or even your friends, should not even imagine that there is a message concealed somewhere.
This very characteristic makes steganography the ideal science for hiding messages on the web, which is flooded by non-significant data. All your passwords and everything you need can be hidden inside three or four 'fake' pages you have uploaded somewhere, within images or pictures such as “my sister Christine and her lovely kids” or whatever.
You download fake images from the web, you modify them (the greatest risk for steganography is the confrontation between 'original' image without the concealed message and steganogated image with the message, of course), and only after these modifications you will hide your concealed message inside the images with one of the many programs that are available.
Basically, using steganography, you can smuggle any file(s) over any communication-line (BBS, network, modem, etc.) in a format which leaves the smuggled data untraceable and unreadable.
But there is not only a military or privacy benefit of information-hiding. Many companies use steganography for copyright-issues.

How things work

To understand, how stego-tools alter the host-file it is necessary to know how the data is stored in the original file. For uncompressed files, things are straight forward. We will discuss steganography on the most commonly used file formats.

uncompressed audio

Sound samples are quantized, inaccurate estimates of the analog value of the sound wave at a given moment in time. Windows WAV files can have different samplingresolutions: 8bit, 16bit and Steinberg Research [23] offers sounddrivers (ASIO-drivers), that can have resolutions of 20bit, 24bit and 32bit float. For the more commonly used 8 bit samples this means that the values can range between 0 and 255. 16 bit samples range between 0 and 65535.
[2] Let´s take a WAV file with a samplingrate of 44.1kHz sampled in mono and having a resolution of 16 bits. In these files, the data is stored sequentially byte by byte.
When we take a look on the properties, and bearing in mind that we want to alter the file in a way that we can´t notice the difference, there are two basic factors that influence the quality of the hostfile.

- The samplingrate – The higher the sampling rate, the more data is needed for the same samplingtime. This means, that there is a greater number of samples to hide the information in, making the hidden data more difficult to track.
- The other parameter is the samplingresolution. In our example file, we have a resolution of 16 bits. That makes 65536 different samplevalues.

If we change the samplevalue by one, you can´t really hear the differrence.
However, if the sampleresolution is 4 bits, doing the same change on this sample would have a much greater effect and therefore increase the level of disturbing samplenoise.
It is easy to see, that it is much simpler to hide information in a huge hostfile.
To make things easier suppose that a 16bit sound sample had the following eight bytes of information in it somewhere:

132 134 137 141 121 101 74 38

The binary numbers are shown in the table below. Only the second byte is represented, because of the less space. The second line shows only the LSB of the corresponding byte.

1000010**0**	1000011**0**	1000100**1**	1000110**1**	0111100**1**	0110010**1**	0100101**0**	0010011**0**
0	**0**	**1**	**1**	**1**	**1**	**0**	**0**

Suppose that we want to hide the binary byte 11010101 (213) inside this sequence. We simply replace the LSB (Least Significant bit) of each sample byte with the corresponding bit from the byte we are trying to hide. So the above sequence will change to:

1000010**1**	1000011**1**	1000100**0**	1000110**1**	0111100**0**	0110010**1**	0100101**0**	0010011**1**
1	**1**	**0**	**1**	**0**	**1**	**0**	**1**

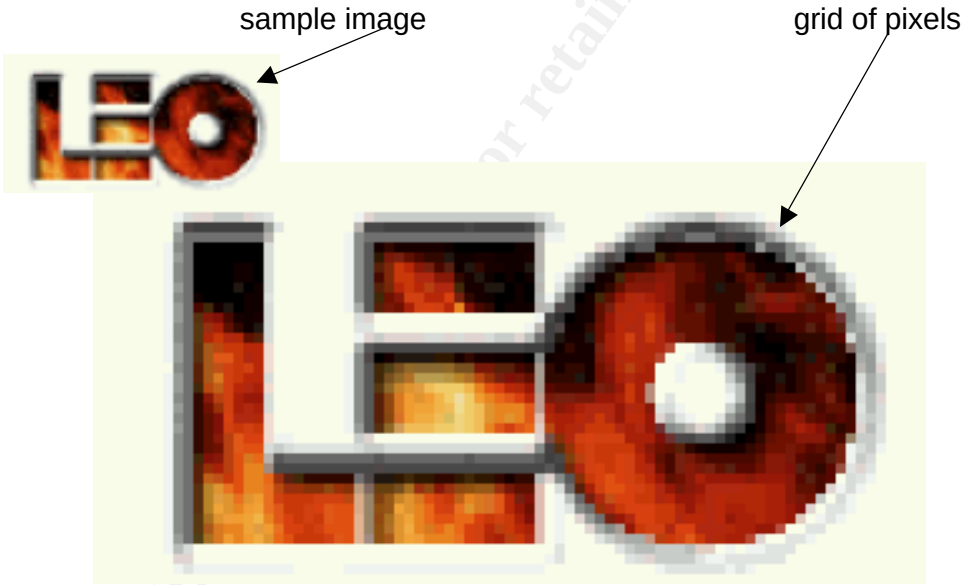
This leads to the following sequence:

133 135 136 141 120 101 74 39

As you can clearly see, the values of the sound samples have changed by, at most, one value either way. This will be almost inaudible to the human ear, yet we have concealed 8 bits of information within the sample.

uncompressed video

[2] Another hostfile-format is the BMP file. All computer based pictures are composed of an array of dots, called pixels, that make up a very fine grid. Each one of these pixels has its own colour, represented internally as separate quantities of red, green and blue. Each of these colour levels may range between 0 (none of the colour) and 255 (a full amount of the colour). A pixel with an RGB value of 0 0 0 is black, and one with a value of 255 255 255 is white. This leads to 16777216 different colours in total.



As covered with the WAV files, it is easier to hide information within a huge hostfile. The more pixels you have in your file is one factor to achieve this. The other is the resolution per pixel. For a 24 bit image, information-hiding is simple because 24 bit images are stored internally as RGB triples, and all we need to do is spread our bits out and save the new file. The drawback to this is that uncompressed 24 bit images are uncommon, and would therefore attract the attention of those whose attention you are trying to avoid attracting! They are also very large as they contain 3 bytes for every pixel (for a 640x480 image this is 640x480x3=921.6 kbytes). It is considerably more difficult to hide anything within a 256 colour gif-image. This is because the image may already have over 200 colours which by changing, will carry way over the absolute maximum of 256.

Looking at a little theory it is easy to see that an image with 32 or less colours will never exceed 256 colours, no matter how much we tamper with it. To see this, visualise the 3 LSB's of an RGB triple as a 3-bit number. As we pass through it in our hiding process, we can change it to any one of 8 possible values, the binary digits from 000 to 111, one of which is the original pattern. If one colour can “expand” up to 8 colours, how many distinct colours can we have before we are in danger of exceeding the limit of 256? Simple, 256/8=32 colours. However, there is no guarantee that 32 colours is the upper limit, for every file that you want to hide. If you're lucky the file will not change a colour to all of its 8 possible combinations and then we are able to keep one more of the original colours. In practice, however, you will often find pictures being reduced to the minimum of 32 colours.
Typical tools are StegoDos, S-Tools, Mandelsteg, EzStego, Hide and Seek (versions 4.1 through 1.0 for Windows 95), Hide4PGP, Jpeg-Jsteg, White Noise Storm and Steganos. The image formats typically used in such steganography methods are lossless and the data can be directly manipulated and recovered.

compressed audio

[11] In 1987, the Fraunhofer IIS started to work on perceptual audio coding in the framework of the EUREKA project EU147, Digital Audio Broadcasting (DAB). In cooperation with the University of Erlangen, the Fraunhofer IIS finally devised a very powerful algorithm that was standardized as ISO-MPEG Audio Layer-3. By using MPEG audio coding, you may shrink down the original sound data from a CD by a factor of 12, without noticably losing sound quality. Basically, this is realized by perceptual coding techniques addressing the perception of sound waves by the human ear.
The perceptual model mainly determines the quality of a given encoder implementation. It uses either a separate filter bank or combines the calculation of energy values (for the masking calculations) and the main filter bank. The output of the perceptual model consists of values for the masking threshold or the allowed noise for each coder partition. If the quantization noise can be kept below the masking threshold, then the compression results should be indistinguishable from the original signal.
A system of two nested iteration loops is the common solution for quantization and coding in a Layer-3 encoder.
Quantization is undertaken via a power-law quantizer. In this way, larger values are automatically coded with less accuracy and some noise shaping is already built into the quantization process. The quantized values are coded by Huffman coding. As a specific method for entropy coding, Huffman coding is **lossless**. This is called noiseless coding because no noise is added to the audio signal.
The process to find the optimum gain and scalefactors for a given block, bit-rate and output from the perceptual model is usually done by two nested iteration loops in an analysis-by-synthesis way.

- The inner iteration loop (rate loop):

The Huffman code tables assign shorter code words to (more frequent) smaller quantized values. If the number of bits resulting from the coding operation exceeds the number of bits available to code a given block of data, this can be corrected by adjusting the global gain, which results in a larger quantization step size, and leads to smaller quantized values. This operation is repeated with different quantization step sizes until the resulting bit demand for Huffman coding is small enough. The loop is called rate loop because it modifies the overall coder rate until it is small enough.

- The outer iteration loop (noise control/distortion loop):
  To shape the quantization noise according to the masking threshold, scalefactors are applied to each scalefactor band. The system starts with a default factor of 1.0 for each band. If the quantization noise in a given band is found to exceed the masking threshold (allowed noise) as supplied by the perceptual model, the scalefactor for this band is adjusted to reduce the quantization noise. Since achieving a smaller quantization noise requires a larger number of quantization steps and thus a higher bitrate, the rate adjustment loop has to be repeated every time that new scalefactors are used. In other words, the rate loop is nested within the noise control loop. The outer (noise control) loop is executed until the actual noise (computed from the difference of the original spectral values minus the quantized spectral values) is below the masking threshold for every scalefactor band (i.e. critical band).

The hiding process takes place at the heart of the layer III encoding process, namely in the inner rate loop. The bits are encoded as its parity by changing the end loop condition of the inner loop.

As we can see, it is not possible to modify the mp3-file itself to hide the information. We have to decompress the file first. Most of the stego-audio-tools available use a wave-file as source.

compressed video

[12] JPEG (created by Joint Photography Experts Group) image is a file-format that uses lossy compression. This image format saves space but may not maintain the original message integrity, so users need to keep in mind that by using this type of compression on an image, some information is lost.

Below is the outline of the baseline compression algorithm:

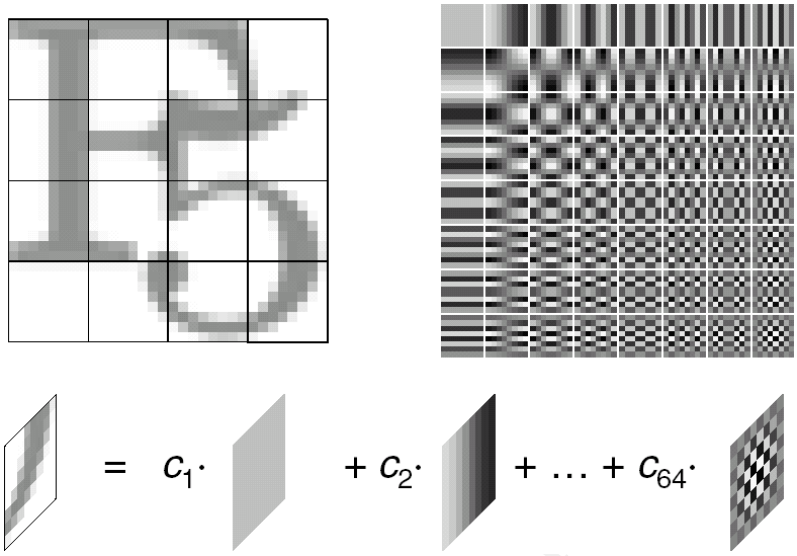
1. Transform the image into a suitable colour space. This is a no-op for grayscale, but for colour images you generally want to transform RGB into aluminance/chrominance colour space (YCbCr, YUV, etc). The luminance component is grayscale and the other two axes are colour information. The reason for doing this is that you can afford to lose a lot more information in the chrominance components than you can in the luminance component: the human eye is not as sensitive to high-frequency chroma info as it is to high-frequency luminance. You don't have to change the colour space if

you don't want to, since the remainder of the algorithm works on each colour component independently, and doesn't care what the data is. However, compression will be less since you will have to code all the components at luminance quality. Note that colourspace transformation is slightly lossy due to roundoff error, but the amount of error is much smaller than what we can see later on.

2. (Optional) Downsample each component by averaging together groups of pixels. The luminance component is left at full resolution, while the chroma components are often reduced 2:1 horizontally and either 2:1 or 1:1 (no change) vertically. In JPEG-speak these alternatives are usually called 2h2v and 2h1v sampling, but you may also see the terms "411" and "422" sampling. This step immediately reduces the data volume by one-half or one-third. In numerical terms it is highly lossy, but for most images it has almost no impact on perceived quality, because of the eye's poorer resolution for chroma info. Note that downsampling is not applicable to grayscale data; this is one reason colour images are more compressible than grayscale.
3. Group the pixel values for each component into 8x8 blocks. Transform each 8x8 block through a discrete cosine transform (DCT). The DCT is a relative of the Fourier transform and likewise gives a frequency map, with 8x8 components. Thus you now have numbers representing the average value in each block and successively higher-frequency changes within the block. The motivation for doing this is that you can now throw away high-frequency information without affecting low-frequency information.
4. In each block, divide each of the 64 frequency components by a separate "quantization coefficient", and round the results to integers. **This is the fundamental information-losing step.**
5. Encode the reduced coefficients using either Huffman or arithmetic coding. Notice that this step is **lossless**, so it doesn't affect image quality. The arithmetic coding option uses Q-coding; it is identical to the coder used.
6. Tack on appropriate headers, etc, and output the result. In a normal "interchange" JPEG file, all of the compression parameters are included in the headers so that the decompressor can reverse the process.
7. The decompressor multiplies the reduced coefficients by the quantization table entries to produce approximate DCT coefficients. Since these are only approximate, the reconstructed pixel values are also approximate, but if the design has done what it's supposed to do, the errors won't be highly visible.

A high-quality decompressor will typically add some smoothing steps to reduce pixel-to-pixel discontinuities. The JPEG standard does not specify the exact behavior of compressors and decompressors, so there's some room for creative implementation. In particular, implementations can trade off speed against image quality by choosing more accurate or faster-but-less-accurate approximations to the DCT. Similar tradeoffs exist for the downsampling/ upsampling and colourspace conversion steps.

Tools, like JPG-JSTEG, circumvent the lossy part of the compression algorythm. The compression is done in two steps. After the first and lossy DCT, the image is converted to its frequency components.



The LSBs of all nonzero frequency components (c1 ... c64) are replaced by the bits of the message that is to be hidden. The second step is the lossless part of the compression anyway. This operation modifies the frequency coefficients, so a secret message is not easy to find within the jpg-file. Since Jsteg is easy to detect by a statistical attack, further generations of the program have made the hidden message much more difficult to detect. The latest version, F5, uses matrix encoding and permutative straddling. The details on this are beyond the scope of this document, but can be found here:

http://os.inf.tu-dresden.de/~westfeld/publikationen/f5.pdf

Now that we have learned how the files are structured and where the modification takes place, we are able to understand what the tools are doing. Most of the tools are doing an encryption on the information before it is hidden within the hostfile.

Destructive countermeasures

When a suspicious image is found, one must decide whether (potentially) hidden information has to be erased or not.
As we have learned, the lossy compressed image-files carry the information within the DCT-coefficients, so simply decompressing and recompressing with the same properties, should remove the information while preserving almost the same filesize.

More effort is necessary to remove the hidden message from an uncompressed or lossless compressed file. The following filters can be applied to an image to erase hidden information:

- rotation
- smoothing
- bluring effects
- sharpen
- noise reduction
- resampling
- ...

In other words filters have to be applied, that modify the pixelproperties.
The same countermeasures have to be applied to sound-files. Like the image-files a lossy compressed audio-file can be recompressed to remove the potentially hidden message. Lossless wav files have to be modified with filters such as one of the following:

- equalisation
- compression
- slight reverb
- low amplified delays
- stereo enhancements
- resampling
- ...

Not every filter applied to a file removes a hidden message, so cropping an image or soundfile is not always a good choice. Even amplification of colours or wavesamples may not erase the message, because the information might be the position of zerocrossings, which is not changed by this type of modification. On .gif-files all colours in the colourtable are modified the same way, so the number of different colours remains the same.

Using simple tools

Hiding information within another file is no real problem these days. The tools that you need, can easily be found on the internet. The following tools modify the files mentioned above.

F5

F5 is a commandline-tool, which is used to hide information within bmp- or jpg-files. To hide information just use the following command:

jwiew Embed -e text.txt -p password image.bmp outimage.jpg

To get the imformation back, the recipient must know the “password”. He/she then types:

jview Extract image.jpeg -p password

The embeded text can be found in the file output.txt in the F5 directory.

MP3Stego

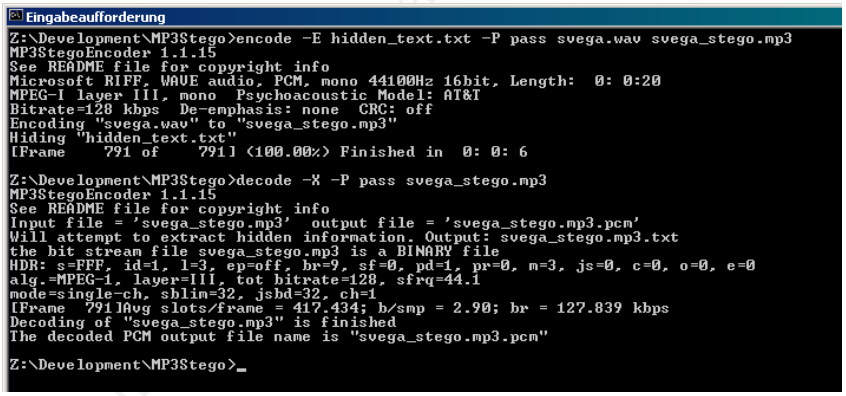
[3] Another easy to use program is MP3Stego. As mentioned before, it is necessary to have an original wave-file from which an mp3-file is then generated. MP3Stego consists of two programs: encode and decode. The usage is very simple:

encode -E hidden_text.txt -P pass source.wav stego.mp3

This compresses source.wav (mono, 44.1 kHz, 16bit encoded) and hides hidden_text.txt. The hidden text is encrypted using “pass” as a password. It then produces the output called “stego.mp3”.

decode -X -P pass stego.mp3

uncompresses svega_stego.mp3 into svega_stego.mp3.pcm and attempts to extract hidden information. The hidden message is decrypted, uncompressed and saved into svega_stego.mp3.txt.



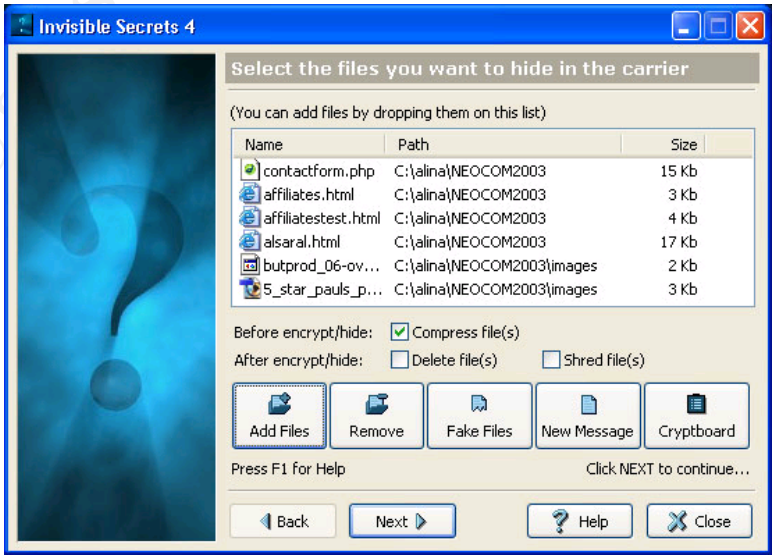
There is also another interesting project called “stego-lame” hosted at source-forge [14]. This is an alpha-version program in Windows C source code which hides data in various audio streams (MP3, Ogg Vorbis, MPEG 2/4 AAC, G.72x). But there are no precompiled binaries yet.

Invisible Secrets 4

[19] A really great program suite is “Invisible Secrets 4”. Like steganos it covers most of the topics that digital steganography deals with. Invisible Secrets 4 not only encrypts the data and files for safe keeping or for secure transfer across the net, it also hides them in places that on the surface appear

totally innocent. With Invisible Secrets 4 you may encrypt and hide files directly from Windows Explorer, and then automatically transfer them by e-mail or ftp.
The program features strong encryption algorithms (including AES - Rijndael), a shredder that helps you destroy files beyond recovery, a password management solution that stores all your passwords securely and helps you create secure passwords, folders and internet traces, a locker that allows you to password protect certain applications, the ability to create self-decrypting packages and mail them to your friends or business partners, a tool that allows you to transfer a password securely over the internet, and a cryptboard to help you use the program from Windows Explorer.
Invisible Secrets 4 offers a wizard that guides you through all the necessary.
After starting up the application you´ll see the following screen:

This screen offers all the programoptions. As you can see, this is a GUI-based “all in one” program. The hiding features can be found behind the first menu item. Clicking on this item brings a file selector box on top of the screen.



Here you can choose the files you want to hide. When all files of interest are selected, the encryption settings can be made on the next dialog. After setting up the encryption, the user can choose the way he wants to send the file. Either by mail or by ftp.

Where to get the software

There are, of course, some commercial tools, but there are also a huge number of free tools on the internet. A good starting point is [1], [4], [6] and [12]. As many of the tools available are command line oriented they are very usefull in scripts. On some of the mentioned nntp servers you can find good ideas for writing scripts, that hide your information in an email automatically.

The systemrequirements

As there are many tools available that can run under almost any operating system (including the old MS-DOS), the requirements of the system are extremely low. Even on a smartphone, like the Nokia Communicator 9210, it is quite simple to download a file from the internet and steganogate it within an XTM emulator window, that is able to run DOS, Windows 3.1 and even LINUX [25].

Pros and cons

Imagine a journey to a foreign country where the customs service is very tight. But the only media they can see is an audio CD. Nobody would even think of you having stored critical data on it. Apart from the problem, that jitter correction may destroy the data during the extraction process, this is another good way of hiding data from the eyes of those you do not want to see it. Steganography has endured bad publicity recently, with unsubstantiated rumors of missives from Osama bin Laden, hidden in images on websites.
The negative aspect for the “normal” user is the fact that little information has to be hidden within huge hostfiles. For the ones with a dial in connection this seems to be a very expensive solution, since the files need a long time to be transferred.
With a little perl-script, on the other hand, one can easily have a substitute for the UNIX “mail” command, that prepares a jpg-file before it is then transferred using the standard mail handler.
A part of the script might look like the one I am using on my windows-PC:

```
print 'Whats the email-address [me@home.com]: ';
chop ( $eingabe = <STDIN> );
if ( ! $eingabe ) {
   $eingabe = 'me@home.com';
}
$address = $eingabe;
```

```
print 'Where is the message [e:\outmessage]: ';
chop ( $eingabe = <STDIN> );
if ( ! $eingabe ) {
   $eingabe = 'e:\outmessage';
}
$outmsg = $eingabe;

print 'What is the password [secret]      : ';
chop ( $eingabe = <STDIN> );
if ( ! $eingabe ) {
   $eingabe = 'secret';
}
$password = $eingabe;

print 'Where is the image   [e:\srcimage.bmp]: ';
chop ( $eingabe = <STDIN> );
if ( ! $eingabe ) {
   $eingabe = 'e:\srcimage.bmp';
}
$image = $eingabe;

print 'Where to put the image [e:\stegimage.bmp]: ';
chop ( $eingabe = <STDIN> );
if ( ! $eingabe ) {
   $eingabe = 'e:\stegimage.bmp';
}
$outimage = $eingabe;

system("start -w jwiew Embed -e $outmsg -p $password $image \\
$outimage");

system("rundll32.exe shell32.dll,MailToProtocolHandler \\
$address");
```

The script first collects the necessary information and embeds it within a jpg-file. The `start -w` command waits until the process is done. After this step the standard mailclient is opened, so you just have to attach the picture.
But this should only give an idea of the process. One can easily do an encryption before the message is hidden etc.

Conclusion

Steganography and cryptography combined, are a powerful weapon to get information to the recipient. With tools like PGP you can guarantee, that the message is not touched or seen on its way. Hiding this message within any hostfile (e.g. .pdf, .html, .txt, .jpg, .exe, .dll, ...) makes it almost impossible for someone to even have an idea of its existence.
When taking a look around in the masses of tools, you can find some interesting ones to even hide the message within the headers of IP-traffic. There is even a LINUX-filesystem called StegFS [22], that shows up almost like an ext2fs (with some additional features, of course). However, the drawback of this filesystem is that the movement of the heads, which is

necessary for spreading the bits across the harddrive, makes this filesystem very slow.
Steganography is just starting to become popular. As more and more people become concerned with their privacy and more and more regulations are being passed, steganography will increase in popularity.

References

[1] *Neil Johnson: Steganography and Digital Watermarking Tool Table*
http://www.jjtc.com/Steganography/toolmatrix.htm

[2] *Andrew Brown: Helpfile from s-tools4*

[3] *Fabien A.P. Petitcolas: MP3Stego*
http://www.cl.cam.ac.uk/~fapp2/steganography/mp3stego/

[4] *Fabien A.P. Petitcolas: steganographic software*
http://www.cl.cam.ac.uk/~fapp2/steganography/stego_soft.html

[5] *Ross J. Anderson, Fabien A.P. Petitcolas: On The Limits of Steganography*
http://www.cl.cam.ac.uk/~fapp2/publications/jsac98-limsteg.pdf

[6] *Hide data in oblivious channels. (Linux cryptography software.)*
http://munitions.vipul.net/dolphin.cgi?action=render&category=06

[7] *COTSE-Steganography Tools*
http://www.cotse.com/tools/stega.htm

[8] *free-it.org: linux kryptografie datenschutzlinks open source*
http://www.free-it.org/archiv/0_Produkte/krypto-datenschutz.html

[9] http://www.demcom.com/english/steganos/index.html

[10] *Neil F. Johnson: Steganography*
http://www.jjtc.com/Steganography/

[11] *Fraunhofer IIS - AMM - Layer-3 Info*
http://www.iis.fraunhofer.de/amm/techinf/layer3/

[12] *Joint Photography Experts Group*
http://www.jpeg.org/

[13] *StegoArchive.Com - Steganography Information, Software to Enhance Your Privacy*
http://www.jjtc.com/stegoarchive/stego/software.html

[14] *Sourceforge: stego-lame*
http://sourceforge.net/projects/stego-lame/

[15] *Burkhard Schröder: Kryptographie und Steganographie*
http://www.burks.de/krypto.html

[16] *Heise Verlag: Cryptocampagne*
http://www.heise.de/ct/pgpCA/stego.shtml

[17] *Heise Verlag: Cryptocampagne*
http://www.ct.heise.de/ct/01/09/links/170.shtml

[18] *GSEC Security Essentials Toolkit*
http://www.securityhaven.com/settools.html

[19] *Neobyte Solutions: Invisible Secrets*
http://www.invisiblesecrets.com

[20] *Neil F. Johnson: Steganography: Seeing the Unseen*
http://www.jjtc.com/pub/r2026.pdf

[21] *Neil F. Johnson: An Introduction to Watermark Recovery from Images*
http://www.jjtc.com/pub/nfjidr99.pdf

[22] *Andrew D McDonald: StegFS, Partition level steganographic filesystem for Linux.*
http://www.mcdonald.org.uk/StegFS/

[23] *Steinberg Research: Professional Audio Editing Tools*
http://www.steinberg.net

[24] *NBInfo: XTM PC-Emulator ER6*
http://www.nb-info.co.uk/xtminfo.htm

[25] *Collin Jörg: AIKON-Reviews – 9210 with DOS, Windows3.1 and LINUX*
http://www.aikon.ch/reviews/sw_review.php3?softwareid=230