Steganography: an Inside Look at Hiding Messages and Data

Ashley Anderson

University of Exeter

COM3500 Dissertation

5th December 2006

Abstract

Encryption is all around us these days, yet the security of private data on home computers is overlooked by many. The presence of cipher text in some instances can make the situation worse, therefore the only way to overcome this is to hide the data. The Ancient Greeks knew this, and as such fabricated some of the most ingenious methods of steganography ever used. However, steganography has come along way in the last 3000 years, and is more important today than ever before (due to tightening restrictions governing encryption). This paper looks at how different mediums have been, and still are, used to ensure messages and data remain private, and how modern methods guarantee the user plausible deniability, and what to expect in the future.

I certify that all material in this dissertation which is not my own work has been identified.

Ashley Anderson

1. Introduction

Steganography is the art (or science) of hiding sensitive information in such a way that its existence cannot even be proven [Cac98]. The word steganography is derived from Ancient Greek and literally means “covered writing”. Throughout history many different techniques have been used to hide messages, including – but in no way limited to – invisible inks, tattoos hidden under a head of hair, microdots and wax tablets. Now as we entered the digital age, a new opportunity has arisen for hiding data which was previously impossible. Computers enable information to be hidden within documents, data or even programs, using increasingly complex algorithms, thus becoming less susceptible to detection.

The need for steganography came about to prevent the interest which accompanied coded or enciphered messages. For instance, if a censor was to come across a message which was a single, almost meaningless, sentence, or random list of letters, number or other characters, then he can be almost sure that what he has is some form of encrypted message, and until within the last thirty years or so, all forms of encryption have eventually been broken by cryptologists the world over (substitution cipher, Vigenère Cipher, et cetera). It's only that modern encryption techniques are complex enough to ensure that a standard dictionary or brute force attack will take an unfeasibly long time.

§2.1 looks at how steganographic techniques have changed over the last few thousand years – ranging from some of the earliest stories of the Ancient Greek world, all through Medieval Europe right up until the Second World War. Some of the methods are bizarre, and others are still used today, albeit in a digital form. Some early methods of sending a message without incurring the penalty of detection relied upon invisible inks – writing the real message between the lines of an innocent cover message. Obviously sending a blank sheet of paper with your invisible message written on it will arouse just as much suspicion as any encrypted message. Some people even went so far as to tattoo the message onto the messengers head, then wait for the hair to grow back before sending the messenger on his way (2000 or so years ago the Ancient Greeks may not have thought speed to be a huge factor in relying information) [Sin99].

The digital age of steganography, §2.2, is divided into two separate subsections: techniques for hiding computer files in other computer files, and how a file system can be used to hide the existence of all the files stored on it. One of the ideas steganography tries to implement, especially with modern techniques, is plausible deniability: any act which leaves little or no evidence of wrongdoing or abuse. The term was first coined in the United States, during the 1950's when “black ops” were carried out without the authorisation of the President, thus when questioned about these operations, he is able to deny his involvement truthfully [Car03]. When this idea is used within computers it generally refers to data for which the owner can deny its existence. It is this idea of plausible deniability that separates digital encryption from digital steganography.

Using encryption on its own is not without its drawbacks: when a message is encrypted, the existence of the message can clearly be seen; whether this is the cipher text used hundreds of years ago, or the pseudo-random bits which are today's digital “cipher text”. Being able to hide the encrypted message adds another layer of security. This can be as simple as a null cipher being used to hide a message encrypted with a Caesar-shift substitution cipher, or as complex as a file first being encrypted using the Blowfish [Sch93] encryption algorithm, and then hidden inside a JPEG[1] image using something akin to least significant bit insertion.

2. Hiding Messages

History has shown us many different techniques that have been used to ensure that secret messages have reached their intended recipient without being tampered with along the way. Initially secure communication channels were used only by those in power, be it in charge of a country or commanding an army; these days however, more and more people are becoming aware of the threat to their own privacy that they feel the need to use (more commonly) encryption, and (more recently) steganography to overcome this threat [MB01].

Many corporations are also beginning to value steganography as a means to protect their innovative design

1 Joint Photographic Experts Group image format. JPEG is a standards committee that designed an image compression format. The compression format they designed is known as a lossy compression, in that it deletes information from an image that it considers unnecessary.

ideas. The following example shows how power could be abused by an one company seeking to gain information about another's upcoming products: An employee may be arrested on suspicion of a crime and asked to hand over a company laptop as evidence (this laptop may contain one or more encrypted partitions to keep either private details or company secrets safe) and required to provide the password to access the data else the disk will be used as evidence of guilt. This could be the result of the rival company bribing the police to acquire the employee's laptop and expose any trade secrets found in files on the disks.

2.1. Pre-Digital Age

There are many different techniques for hiding messages that have been used throughout history. The Ancient Greeks were among the first recorded civilisations to use steganography when sending messages. Perhaps the most drastic method they used was the combination of slaves and tattoos, whereby the slave's master would have the message tattooed upon the shaven head of his slave. Once the hair had grown back, and the message was hidden, the slave could be sent on his way – all he had to do at the other end was shave his head again and point it at the intended recipient[1].
The Ancient Greeks had many other methods for hiding their messages – perhaps the most popular was through the use of wax tablets. These tablets were made from small boards of wood which were covered in wax; the idea being that a message could be written on the wax until it was no longer needed and then the wax was scraped off, melted and reapplied. The Greeks eventually realised that they could send hidden messages to each other by writing their intended message on the wood itself before covering it with wax[2].
Eventually the remainder of Europe caught on to the whole "encryption" band-wagon and began developing ciphers of their own. It's unknown exactly when it happened, but at some time in history the idea of null ciphers began: the message to be hidden is spelled out within the body of the cover text. Such as using the first, second... last letter of each word, or perhaps after a punctuation mark. The use of pin holes under the required letters is another technique, which doesn't require both sender and receiver to agree prior to sending the message which method to use; it essentially allows the sender to disregard any systematic method for choosing which letters of each word/sentence to use.

> 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.

The above passage contains a message which can be found by extracting the second letter from each word. The resulting message was sent by a German spy during World War 2 [Kah67], it reads:

> Pershing sails from NY June 1.

The use of null ciphers hasn't halted as we've entered the digital age: least significant bit insertion (a common method for hiding files inside media) is essentially a digital form of the null cipher; detailed in §2.2.1.
Another tried and tested method for writing a hidden message was to employ the use of an invisible ink. By the end of The First World War, almost every invisible ink known today had then been discovered. It was at that time also noted that inks of this nature were also insecure. There are many liquids that can be used to write an invisible message, most citric juices for example. These, plus milk or liquids high in sugar content (honey, cola or similar, or just sugar water), can easily be developed with the use of heat. Others require the use of a specific chemical to reveal the hidden message – such as red cabbage water to reveal a message written in vinegar, or a message written with copper sulphate can be exposed using ammonium hydroxide.
During the years between WWI and WWII Germany developed microdot technology. This process involved photographing the document to send and decreasing the size of the photograph until the final photograph was

1 Herodotus (an Ancient Greek scholar) recounts the story of Histaiaeus, who wanted to encourage Aristagoras of Miletus to revolt against the king of Persia.

2 Again, this is a story documented by Herodotus involving Greece and Persia: Xerxes of Persepolis was amassing an army to expand his empire by conquering Greece; this was witnessed by Demaratus who used the wax tablet method to warn the cities of Athens and Sparta.

the size of a typographical dot, such as the tittle of a lower case *i* or *j*. Whilst all references to microdots infer that the microdot was of a document, it should be noted that the document could well be encrypted beforehand to hinder the extraction of information once discovered [Whi92].

2.2. (Post-)Digital Age

In the world of digital steganography multimedia has provided us with the medium required to transmit messages in the knowledge that the message will be undetectable. This is partly because, like many functions and manipulation techniques, everything is performed on the individual bits that comprise the resulting file. Images are by far the most utilised in this area. It can be clearly demonstrated that using text files or executables is an unwise choice: changing the bits of a plain text file will obviously change the letters or words of the text – in some instances resulting in unreadable characters being displayed instead. Using binary executables and changing the bits which are contained within can cause undesired program behaviour (destructive in some cases perhaps) but more likely, and at the very least, cause the executable to become non-executable.

2.2.1. Single File Based

It was mentioned in §2.1 that the null cipher set is akin to the digital method known as least significant bit insertion. This similarity is because of how least significant bit insertion works. Where a null cipher uses a specific letter of each word to spell out the secret message, its digital counterpart hides the bits which make up the message in the least significant bits of the cover data. For example, the letter 'A' (expressed as binary) can be hidden inside of 8 cover bytes as follows [JJ98]:

```
10000011

00100111 11101001 11001000 00100111
11001000 10101111 11100110 10101101
```

becomes

```
00100111 11101000 11001000 00100110
11001000 10101110 11100111 10101101
```

Unfortunately, this method of hiding data, does have its drawbacks. Using least significant bit insertion with bitmap images poses no problem, however as soon as any kind of image manipulation begins, the original message can easily become lost and subsequently overwritten with subsequent image data. Image manipulation of this sort includes resizing the image, drawing over the existing image, or saving in a format which uses a lossy compression algorithm, such as the popular format for images on the web, JPEG.

The recommended image format for steganographic use is a 24-bit bitmap. This is due to the way colours are referenced in bitmaps. In 24-bit colour bitmaps, the bytes which determine each pixel's colour store 8-bit RGB[1] values, therefore 24-bit images can make use of every colour offered by graphics cards and operating systems. Lower bit bitmaps make use of a palette to determine

Figure 2.2-1 ~ a 24-bit colour palette

1 Red-Green-Blue. In web design and design for computer monitors, colours are defined in terms of a combination of these three colours. Usually expressed in hexadecimal.

which colours each pixel displays. The bits which define each pixel reference an RGB value in the palette contained in the header of the image file. This is demonstrated the in figure 2.2-1 to the right. The next best alternative to using 24-bit images is either 8-bit (256 colour) or greyscale. 8-bit images lack the number of colours available and increase the risk of colour changes being detected although they do allow for smaller image files size as well as the use of GIF[1] images.

Using a palette with only 256 colours can still cause problems, especially if the palette is not optimised, causing drastically different colours to be ordered "next" to each other. Altering the least significant bit can still cause vast variations in colour, the referenced value will be either increased or decreased by one, and as figure 2.2-2[2], demonstrates this could potentially be enough for someone to notice the change: the blue and red indexes in the centre region of the palette are neighbouring colours on their left and right which are visibly different (green or purple, and grey or orange). Large areas of any of these colours would be a noticeable change within the image.

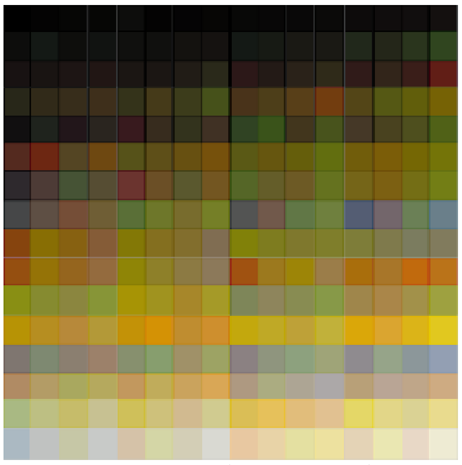


Figure 2.2-2 ~ an 8-bit Renoir colour palette

The 8-bit greyscale palette of figure 2.2-3 is an ideal palette to use with steganography and images, as it has both advantages of using a 24-bit palette and of an 8-bit palette. One major advantage of using only 256 colours results in a smaller file size, and thus arouses less suspicion when being distributed over the web: a 24-bit BMP[3] image 1,024 pixels wide by 768 pixels high (a common resolution for standard digital photographs) is still over 2MB, which is immense in comparison to a JPEG of the same original image at 32KB.

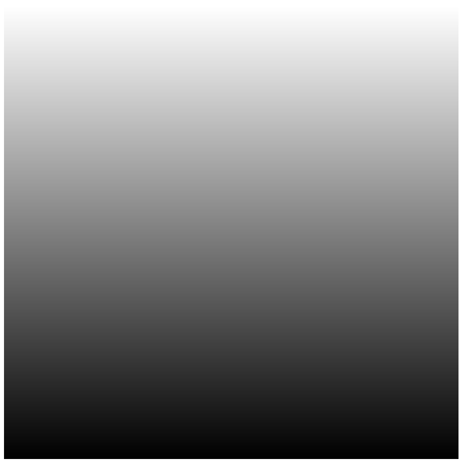


Figure 2.2-3 ~ an 8-bit greyscale colour palette

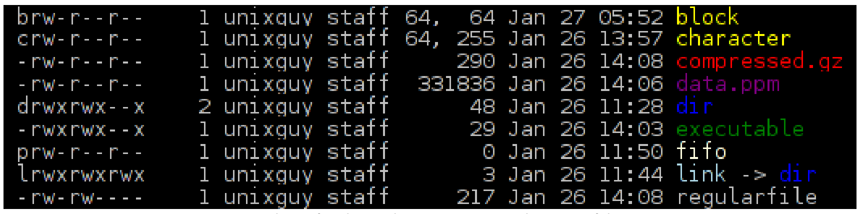
Stegotools[4], a suite of Unix[5] command-line applications to read/write hidden information from/in files using steganography, uses least significant bit insertion to hide data inside of 24-bit bitmaps; however it doesn't just support using the least significant bit, but up to the fourth least significant bit; thereby allowing more data to be hidden, but at the cost of the quality of the resulting steganographic image. It uses nothing more than a systematic step through of each byte of the cover file. Another tool for Unix systems is Steghide[6]. Steghide is more complex than Stegotools: it is able to hide files inside of either BMP, JPEG, WAV[7] or AU[8] file formats, and before being hidden the data file is compressed and then encrypted using the Rijndael algorithm [DR98] with a 128-bit key. Steghide uses a graph-theoretic approach to hiding the original file. A sequence of positions of pixels in the cover file is created based on a pseudo-random number generator initialized with the passphrase – the secret data will be embedded in the pixels at these positions. Of these positions those that do not need to be changed because they already contain the correct value by chance are sorted out. Then a graph-theoretic matching algorithm finds pairs of positions such that exchanging their values has the effect of embedding the corresponding part of the secret data. If the algorithm cannot find any more such pairs all exchanges are actually performed.

1 Graphic Interface designed by CompuServe for using images on line.
2 The Renoir palette is named after Pierre-Auguste Renoir's painting "Le Moulin de la Galette"
3 Bitmap: A proprietary Microsoft Windows image format.
4 Stegotools is available from SourceForge.net (http://sourceforge.net/projects/stegotools/)
5 An operating system co-created by AT&T researchers Dennis Ritchie and Ken Thompson. Some of the popular Unix flavours are: Linux, Solaris, HP-UX, AIX, and BSD.
6 Steghide is available from SourceForge.net (http://sourceforge.net/projects/steghide/)
7 Waveform Audio: a file format was developed jointly by Microsoft and IBM as the standard format for sound on PCs.
8 A sound file format on Sun (and other UNIX) systems.

The pixels at the remaining positions are also modified to contain the embedded data, (but this is done by overwriting them, not by exchanging them with other pixels). The fact that the embedding is done by exchanging pixel values implies that the first-order statistics are not changed. For audio files the algorithm is the same, except that audio samples are used instead of pixels[1].

2.2.2. File System Based

Hiding files inside of other files will always have its limitations, the obvious restriction not mentioned in §2.2.1 is size. As larger files need to be hidden, there is only so much compression that can be achieved, and only so big cover files can be before it becomes inconvenient. The evident solution to hiding files in files is to alter the way files are sorted on any given medium; the file system. There are many different file system implementations, and all have their advantages and disadvantages. There are even encrypted file systems – such as TrueCrypt[2] – yet again though, the presence of cipher text remains. (With cipher text yet to be decrypted, an attacker can continue to demand passwords from a user, and with sufficient motivation, is likely to do so.) File systems have many similar basic file related features: besides the name of the file and its size, there are file permissions, who owns the file and who else can read or modify it. All Unix based file systems have these basic attributes (as shown by a simple 'ls -l' from the terminal in figure 2.2-4 above): type of file, access permissions, number of hard links, owner, group, size, creation time/date, modification time/date, and name. A steganographic file system needs to store all of this information without giving away the existence of any of the files, as well as where to look on the device itself for the data without actually knowing itself. Successfully storing all of this metadata allows the steganographic file system to integrate seamlessly with the host operating system and all applications which look for the metadata to determine how to act on a particular file when encountered.



```
brw-r--r--    1 unixguy staff 64,  64 Jan 27 05:52 block
crw-r--r--    1 unixguy staff 64, 255 Jan 26 13:57 character
-rw-r--r--    1 unixguy staff     290 Jan 26 14:08 compressed.gz
-rw-r--r--    1 unixguy staff  331836 Jan 26 14:06 data.ppm
drwxrwx--x    2 unixguy staff      48 Jan 26 11:28 dir
-rwxrwx--x    1 unixguy staff      29 Jan 26 14:03 executable
prw-r--r--    1 unixguy staff       0 Jan 26 11:50 fifo
lrwxrwxrwx    1 unixguy staff       3 Jan 26 11:44 link -> dir
-rw-rw----    1 unixguy staff     217 Jan 26 14:08 regularfile
```

Figure 2.2-4 ~ an example of what data is stored on a file system

There are two methods for a steganographic file system [ANS98], both have one major similarity in their design. The idea of different levels of security. Files are stored in levels much like files are stored in directories on standard file systems, with each level requiring a password to access it in some way.
The first of these methods makes no assumption about the use of a strong encryption algorithm to obscure the data. It aims to provide protection using linear algebra by requiring anyone who wishes to access a file to know the name of the file exists, and the password used when the file was encrypted and written to disk. Without satisfying both of these preconditions an attacker can gather no information about whether such a file is present – unless he already knows some of the contents of the file or tries all possible file name/ password combinations.
The idea is to have a number of cover files already on the file system that are initialized as random data, and then embed the user's files as the exclusive or of a subset of the cover files (the subset is chosen by the user supplied password). If there are m cover files ($C_0, \ldots, C_{m-1}$) and they are all of the same length, and the user inserts a file F with password P. Then select the bits from the cover files such that C_n for the nth bit of P is 1 and perform a bitwise exclusive or; which is – in turn – XOR'd with the file to be hidden, F, the results of which is again XOR'd with one of C_n. This process results in the user's file F now being the exclusive or of the subset of the C_n selected by the non-zero bits of P. Symbolically expressed [AGJ+92] as:

$$F = \bigoplus_{P_j=1} C_j$$

1 Information about Steghide has been taken from its manual page
2 TrueCrypt is available from SourceForge.net (http://sourceforge.net/projects/truecrypt)

One important aspect of this technique is that if access is linearly hierarchical (storing a file at a given security level requires knowledge of all passwords for all files stored at lower levels) then files can be added in such a way that they do not disturb existing hidden files.
To ensure that the implementation of this design provides the plausible deniability that is sought the user needs to use a number of levels with a number of passwords, ideally more than two or three. As of this stage in the design, the data on the file system is not encrypted, only masked with a cover file. This is essentially the same as using a simple mono-alphabetic substitution cipher, or at best a Vigenère cipher, which these days provide no protection against even a frequency analysis attack. Linear algebra also demonstrates that if the password is *n* bits long, and an attacker knows more than *n* bits of plain text, then after obtaining all of the files he can write *n* linear equations in the *n* unknown bits of the key. For example, a tax inspector might know where to look in a file to find a tax reference number, and this could be enough to break the system [ANS98]. Hence it is assumed that an attacked has no prior knowledge of what could be stored in the system. However, the damage from this kind of attack can be limited by restricting knowledge of what exists on the file system. There is a simple solution to this problem: to pre-encrypt all plain text data before being written to disk, using a key derived from the password. This method's implementation of levelling files allows a user to reveal *x* number of levels (on tax payment manipulation, affairs, et cetera) – and thus the files under them – yet remain quiet about the remaining *y* levels (on industrial trade secrets).
This construct contains one more major flaw: performance. Reading or writing a file would involve reading or writing fifty times, and if this were to access a 10MB file then it is the equivalent of accessing a 500MB file on a standard file system. This issue has arisen because the design, in its initial state, has a complexity of 2^{100} in guessing linear combination. This complexity also provides the file system with a hierarchy with 100 levels in which to hide data. As this is likely to be too large it is propositioned that the complexity is reduced to 2^8 (or 251 – being the largest single byte prime), therefore using sixteen cover files and providing eight levels in the hierarchy. Ultimately this still results in a sixteen-fold read/write penalty.
The second construct makes a different initial assumption: the existence of a good block cipher, for which an attacker can not distinguish cipher text from pseudo-random data. This infers that the presence or absence of a block at any location cannot be determined without knowledge of the encryption key. Unfortunately if the usage of a block cannot be ascertained then the concept of implementing numerous security levels does not hold. Even if files are written to pseudo-random blocks – calculated from a one way hashing function derived from the password used during encryption – eventually after very few blocks have been written "collisions" will occur (where data written begins to overwrite previously written data). This behaviour is because of the birthday theorem[1]. After only $\sqrt{m}$ blocks are written to a file system, with a total of *m* blocks, existing files will be corrupted by subsequent writes. An obvious solution to this problem is to write a file to multiple pseudo-random blocks. Again, there's a catch: imagine that *m* blocks are written, and each block is written *n* times. Now consider the probability that any given block will be overwritten on a subsequent write; if there are *M* of these then the total number of unique blocks being written n_i times from all blocks *N* is [AGJ92]:

$$\pi(n_1, n_2, \ldots, n_M) = 1/N^M \binom{N}{n_1, n_2, \ldots, n_M} \frac{M!}{\prod_{i=1}^{M} (i!)^{n_i}}$$

which can be approximated for computational purposes as:

$$\pi(n_1, n_2, \ldots, n_M) \simeq 1/(N^m \mathrm{e}^{M^2/2\mathrm{N}}) \frac{M^{M-n_1}}{\prod_{i=1}^{M} (i!)^{n_i} n_1!}$$

so the total number of unique blocks being overwritten is:

1 In probability theory, the birthday theorem states that in a group of 23 randomly chosen people, the probability is more than 50% that at least two of them will have the same birthday. For 60 or more people, the probability is greater than 99%, although it cannot actually be 100% until there are over 366 people.

$$\Phi(M, N) = \sum_{1}^{M} \pi(n_1, n_2, \ldots, n_M)$$

where the sum is over all possible combinations of the n_j. Therefore the probability that block k_j is not overwritten is calculated from

$$1 - [\Phi((j-1)m, N)]^k$$

Thus the probability of a file K, being overwritten is:

$$p = \sum_{j=1}^{K} [\Phi((j-1)m, N)]^k$$

There exists no analytical solution for the above equations, nevertheless in practice if the disk is considered to be "full" the first time a corrupt file is read back – all locations reveal overwritten data – then the load factor[1] of the system will be about 7%. This result is somewhat tentative, and can – in practice – be improved upon by allowing a greater percentage of blocks to become corrupt – 10% corruption provides a 20% load factor.

As with the first method there is a read/write penalty, but it's now only five times, instead of from sixteen upwards. However this design idea utilises a Larson table [LK84] which indicates which block to seek out first. Larson table's were designed to allow only one disk access to read from a database. In this construct, the use of a Larson table enables – at any given security level – the first uncorrupted location of each file. Likewise when the file is written back to the file system, this block can be the first to be written to. This list can be generated when a user first enters a new security level and can reside in memory for the duration; when the user leaves the level for a lower level the table in memory is discarded. Therefore there is only a small performance penalty for files which the user accesses immediately upon entering the new security level, and if for a particular file all blocks are corrupt then a warning message can be returned to the user that the file system is (nearly) full. Experiments by Larson and Kajla showed that the disk would appear full once 80-90% of the blocks were occupied. When files are written to the disk the block stored in the Larson table is the first to be written, with the remainder being written in the background, unless the user immediately descends to a lower security level.

Figure 2.2-5 [MK00] shows how this second method for a steganographic file system implements the idea of security levels. Each white rectangle is an inode pointing to a file – where some of the files are in fact directories and therefore deeper security levels which point to more files, and so on. The coloured blocks represent the same file being replicated at different hardware addresses.

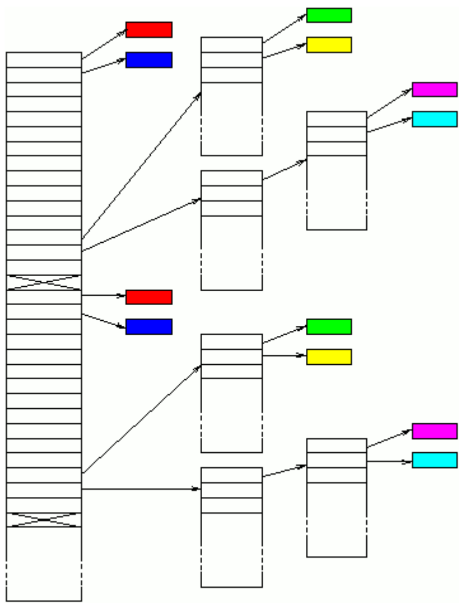


Figure 2.2-5 ~ an inode structure showing files at different security levels

There are already numerous existing steganographic file systems, all inspired by [ANS98] and the initial implementation by [MK00]; they all implement the latter of the two design ideas in their own way. The first implementation of the scheme was StegFS [MK00]. The second NSteg [PTZ03], attempts to eliminate the need to replicate the same file multiple times. It uses a bitmap to set which blocks have been allocated, however this then removes the plausible deniability by proving that there are files in such blocks. There is a second StegFS, which is based on the two previous StegFS implementations[2]. It aims to build upon the word by [MK00]. Magikfs[3] is a fourth steganographic file system, based on the work in [ANS98], [MK00] and [PTZ03] – continuing the work

1 The ratio of total unique blocks stored against the total space allocated.
2 StegFS (the second) is available from SourceForge.net (http://sourceforge.net/projects/stegfs)
3 Magikfs is available from SourceForge.net (http://sourceforge.net/projects/magikfs)

from [PTZ03].
Unfortunately, many of these projects appear to be no longer under active development. StegFS [MK00] was last updated in 2004, and NSteg is even older (2003); lastly, authors of StegFS number two are appealing to anyone who wishes to take over project development.
All of the proposed projects that have released a file system driver have written kernel modules to work with the virtual file system (VFS), alongside other drivers such as Ext2fs. (Magikfs is a FUSE[1] module.) It is also unanimous that the resulting steganographic file system resemble a standard Unix as much as possible: with support for multiple users, fully supported file permissions, et cetera. This ambition has driven project authors to start with the Second Extended File System [CTT93] – commonly known as Ext2 – and add the steganographic functionality. Ext2 has advantages over Ext3 for use in a steganographic system as file operations could be journalled thus hindering the claim of plausible deniability with journalled data becoming accidentally stored at a different level to the file it refers to.
The one major difference between each proposal was how to determine whether a block is being used by a file. To fulfil the plausible deniability aspect, looking at the file system bitmap should not suggest whether a particular block is being used, however if a block is defined as either not used or “maybe used” then this adds some ambiguity. NSteg does just this: when the file system is first placed on a device a number of random files are stored, and further inodes are marked as occupied, in a attempt to allow the user to claim that the file system did it and no real files have been hidden. Magikfs uses a similar technique: users are allowed to write files without setting the block as used. This does add the possibility that files could be overwritten without there being a replicated copy elsewhere on the system. There is a trade-off to be made: increased plausible deniability on one side, increased file integrity on the other.

3. Summary

Throughout history people have needed to communicate with each other, and now an ever increasing number of people are using electronic means to send their messages. These same people are also becoming more aware of their right to privacy, and how many governments are introducing new laws to combat terrorism without thinking about the right to privacy of the average Joe. A thousand years ago, if a private conversation was needed it was sufficient to walk to the middle of a clearing or locked room, and with a quick check, ensure that no one was eavesdropping. Even twenty years ago messages could be sent with the strong belief that they were not going to be read by anyone along their path. It is far from economical to open, read and reseal all mail which passes through a sorting office (except in a dictatorship where citizens are employed to spy on fellow citizens), yet intercepting thousands of emails and searching for key words and phrases can be performed in minutes, if not seconds.
In 1993 Philip Zimmermann took the first step to providing people with a secure way to communicate using email with Pretty Good Privacy [Zim96]. Friends and colleagues could now send and receive emails safe in the knowledge that their messages could not be read by anyone else and guarantee emails were from who they claimed. Now, just encrypting emails is not sufficient, and with more computer viruses in circulation and the advent of spyware being used to steal personal information and documents from home computers, it has become necessary to ensure that a users personal files are secure from attack. Encrypting individual files solved this problem but in doing so introduced another: if it’s encrypted it must be worth encrypting. The presence of cipher text, to some, may seem like fun or a challenge, to others it could imply guilt, therefore to overcome this, hiding the data, not being able to prove that there is any data in the first place, is the solution. Steganography, the art (or science) of hiding messages is this solution.
As far as encryption goes, the code makers have won by devising a method of encryption based on quantum physics and the uncertainty theory [BBS+92], which essentially provides a one-time pad[2] key. As such, provide absolute plausible deniability (even against quantum computers) because for any one-time pad

1 FUSE: a Linux module for file systems in Userspace. FUSE is available from SourceForge.net (http://sourceforge.net/projects/fuse)
2 The one-time pad is an encryption algorithm where the plain text is combined with a random key or "pad" that is as long as the plain text and used only once. A modular addition (for example XOR) is used to combine the plain text with the pad. If the key is truly random, never reused, and kept secret, the one-time pad can be proven to be unbreakable.

resulting cipher text, an infinite number of keys can be used to derive an infinite number of plain text possibilities. Unfortunately, quantum cryptography uses the polarisation of light to determine the value of each bit, and until disks are able to store bit values in a similar fashion this will remain an infeasible method of data storage. Therefore steganographic techniques will have to suffice, and whether files are hidden in other files, or the file system driver itself is modified, digital steganography (for now) guarantees that private data stays secure.

References

Cac98	Christian Cachin, *An Information-Theoretic Model for Steganography*, David Aucsmith (ed.) *Proceedings of 2nd Workshop of Information Hiding*, Springer-Verlag, 1998
Sin99	Simon Singh, *The Code Book: The Secret History of Codes and Code-Breaking*, p. 5, Fourth Estate, 1999
Car03	Rodney P Carlisle, *The Complete Idiot's Guide to Spies and Espionage*, p. 213, Alpha Books, 2003
Sch93	Bruce Schneier, *Fast Software Encryption*, 1993, *Cambridge Security Workshop Proceedings*, pp. 191-204, Springer-Verlag, 1994
Kah67	D. Kahn, *The Codebreakers*, Macmillan New York, 1967
Whi92	William White, *The Microdot: History and Application*, Phillips Publications, 1992
JJ98	Neil Johnson and Sushil Jajodia, *Exploring Steganography: Seeing the Unseen*, *Computing Practices*, pp. 26-34, February 1998
MB01	H. X. Mel and Doris Baker, *Cryptography Decrypted*, Addison-Wesley, 2001
DR99	Joan Daemen and Vincent Rijmen, *AES Proposal: Rijndael*, AES Submission, 1999
ANS98	Ross Anderson, Roger Needham and Adi Shamir, *The Steganographic File System*, David Aucsmith (ed.), *Information Hiding, Second International Workshop*, pp. 73-82, Springer-Verlag, 1998
AGJ$^+$92	R. Anderson, R. Gibbens, C. Jagger, F. Kelly and M. Roe, *Measuring the Diversity of Random Number Generators*, preprint, 1992
LK84	P. Å. Larson and A. Kajla, *File Organisation: Implementation of a Method Guaranteeing Retrieval in One Access*, *Communications of the ACM*, pp. 670-677, 1984
MK00	Andrew McDonald and Markus Kuhn, *StegFS: A Steganographic File System for Linux*, 1999, A. Pfitzmann (ed.), *Information Hiding*, pp. 463-477, Springer-Verlag, 2000
PTZ03	HweeHaw Pang, Kian-Lee Tan and Xuan Zhou, *StegFS: A Steganographic File System*, *Proceedings of the 19th International Conference on Data Engineering*, pp. 657-668, 2003
CTT93	Remy Card, Theodore Ts'o and Stephen Tweedie, *Design and Implementation of the Second Extended Filesystem*, 1993, Frank B. Brokken et al. (eds.), *Proceedings of the First Dutch International Symposium on Linux*, State University of Groningen, 1995
Zim96	Philip Zimmermann, *The Official PGP User's Guide*, MIT Press, 1996
BBS$^+$92	Charles Bennett, Gilles Brassard, François Bessette, Louis Salvail, John Smolin, *Experimental Quantum Cryptography*, 1991, *Scientific America*, pp. 26-33, 1992