Steganography-The Art of Hiding Data

Kefa Rabah
Department of Physics, Eastern Mediterranean University, Gazimagusa, North Cyprus, via Mersin 10, Turkey

Abstract: Given the sheer volume of data stored and transmitted electronically in the world today, it is no surprise that countless methods of protecting such data have evolved. One lesser-known but rapidly growing method is steganography, the art and science of hiding information so that it does not even appear to exist. Moreover, in an ideal world we would all be able to openly send encrypted email or files to each other with no fear of reprisals. However, there are often cases when this is not possible, either because you are working for a company that does not allow encrypted emails or perhaps the local government does not approve of encrypted communication. This is one of the cases where Steganography can help hide the encrypted messages, images, keys, secret data, etc. This paper discusses the purpose of steganography. Explains how steganography is related to cryptography as well as what it can and cannot be used for. It also discusses a brief history of steganography. In addition, some of the tools and software used in steganography are demonstrated and including some discussion of the most popular algorithms involved in these tools. This paper further explains the advantages and disadvantages, as well as, strengths and weaknesses in the use of steganography.

Keywords: Steganography, cryptography, hiding information

INTRODUCTION

The word “steganography” can be defined as “covered writing” or the technique of hiding messages inside other messages. It is derived from a Greek word “steigamos”. This idea of data hiding is not a novelty, it has been used for centuries all across the world under different regimes - but to date it is still unknown to most people - is a tool for hiding information so that it does not even appear to exist. The practice dates back to ancient Greece, where messages were etched into wooden tablets and then covered with wax or created by shaving a messenger’s head and then tattooing a secret message on it, letting his hair grow back and then shaving it again after he arrived at the receiving party to reveal the message.

In modern times, steganography can be looked into as the study of the art and science of communicating in a way, which hides the existence of the communication, has until recently been the poor cousin of cryptography. Currently there are two directions within steganography: one of the branches is used for protection against detection while another is used for protection against removal. The latter option is used to place hidden “trademark” in images, music and software, a technique referred to as watermarking. Today, it is gaining new popularity with the current industry demands for digital watermarking and finger printing of audio and video. 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. Steganography is in the literature (especially military) also referred to as transmission security or short TRANSEC.

One of the most common uses of modern steganography in the digital world of computers is to hide information from one file in the contents of another file. For example, the least significant bits of an image or audio file can be replaced with data from a text file in such a way that a casual third-party observer would detect little if any loss of image or audio quality. An image posted on the internet, or public domain, for anyone in the world to see could secretly contain a highly sensitive text message but generate no suspicion at all. Recent articles in some national media sources such as USA today have pointed to this technique as a tool possibly used for communication between members of terrorist organizations planning attacks against the Western interests. Of course there are many other ways that steganography might prove useful to individuals with more benign motives.

Corresponding Author: Kefa Rabah, Department of Physics, Eastern Mediterranean University, Gazimagusa, North Cyprus, via Mersin 10, Turkey
In the field of steganography, some terminology has developed. The adjectives cover, embedded and steno were defined at the Information Hiding Workshop held in Cambridge, England. The information to be hidden in the cover data is known as the "embedded" data. The "steno" data is the data containing both the cover signal and the "embedded" information. Logically, the process of putting the hidden, or embedded data, into the cover data, is sometimes known as embedding. Occasionally and especially when referring to image steganography, the cover image is known as the container. The term "cover" is used to describe the original, innocent message, data, audio, still, video and so on. When referring to audio signal steganography, the cover signal is sometimes called the "host" signal. This process could be represented in Fig. 1 and the following formula:

\[
\text{cover medium} + \text{embedded message} + \text{stego-key} = \text{stego-medium}
\]

The possible carriers of these hidden messages are innocent looking carriers such as images, video, audio and text. A message transmitted between Alice and Bob could be plaintext, ciphertext, images, or anything that could be represented using a bit stream. Carrier and the message being transmitted create a stego-carrier or a container. Stego-carrier may use a secret-key to embed the information. Traditionally, Bob and Alice could be military jets, e-business or just friend trying to have a private conversation or exchange digital information. They can't stop Eve listening to their radio signals (or tapping their phone line, or whatever), so what can they do to keep their communication secret? One solution is for Alice and Bob to exchange a digital-key, so they both know it, but it's otherwise secret. This secret-key is referred to as stego-key. Alice uses this key to hide messages she sends and Bob reconstructs the original messages by decoding with the same key, (Fig. 2).

Encryption of emails is another example where the field of Steganography is used. In this case the encrypted messages and pictures are hidden in other pictures. It perhaps can be used in companies, which does not allow encryption of messages, pictures, etc. It can also be used in a cases where local governments do not allow the same situation, which is a true case in some countries where citizens are oppressed by the authoritarian regimes. But not only citizens of countries with authoritarian governments should worry about privacy. In the United States violation of privacy could also become an issue soon, especially since advent of 9/11. For example, the Justice Department have since asked the congress for new authority allowing Federal Agents armed with search warrants to secretly break into homes and offices to obtain decryption keys or passwords or to implant "recovery devices" or otherwise modify computers to ensure that any encrypted messages or files can be read by the local National Security Agencies (NSA).

In fact, the atmosphere of paranoia about messages being transmitted was so intense that several restrictions were put in place that might seem ridiculous today. In the USA, for example, the international mailing of postal chess games, knitting instructions, newspaper clippings and children's drawings was banned. Even international orders for flower deliveries were eventually banned by the US and British governments. With the advent of computer age, steganography has been given a marvelous boost. Old methods, such as hiding messages in images, have been given new leases of life brought about by digital age through the use of computers. We are sure to see a great expansion of steganographical techniques in the coming years.

One such way that steganography could be useful is by allowing someone to hide sensitive information on their computer in a way that makes it impossible to prove that the information exists. This is the purpose of a steganographic file system. Faced with such a file system, an attacker who does not possess the name of a file and the password (or stego-key) for accessing it cannot determine whether the file is even present. Note that the concept of steganography is different from the idea of a cryptographic file system, which enciphers or garbles files to protect their contents until a decryption key or password is entered. A limitation of cryptography is that once the presence of encrypted data has been established and until no encrypted files remain on a system, there are multiple ways for an attacker to gain access to such data. One tactic might be to persuade or coerce the data owner into revealing passwords or
Fig. 2: Implementation of image hiding technique using steganography.

decryption keys. Unfortunately, if the data owner has no way of plausibly denying that additional files remain encrypted, an attacker may become quite persuasive indeed. Of course, if it is impossible to prove that files remain hidden on a disk, an attacker might be less tempted to look for them or to coerce the owner into revealing them.

The concept of plausible deniability is in many ways the opposite of non-repudiation, a term that may be more well-known in the security industry. Non-repudiation, a critical component of e-commerce applications, makes it absolutely possible to prove that an individual sent a particular message, performed a given action, or is in possession of certain information. In contrast, plausible deniability makes it absolutely impossible to prove such things. Even if a skilled attacker has unlimited access to a computer and understands it completely, a steganographic file system allows the data owner to plausibly deny that any files are hidden within their computer system.

**Historical background:** The Greek historian Herodotus and date back to Greek times, recorded the earliest records of steganography. The writing medium of the time was text. He describes how a man named Harpagus killed a hare and hid a message inside its belly. Then he sent the hare with a messenger disguised as a hunter. Quite another clever way of hiding data was also demonstrated in Ancient Greece. When the Greek tyrant king Darius in Susa held Histiaeus as a prisoner during the 5th century BCE, he had to send a secret message to his son-in-law Aristagoras in Miletus. Histiaeus shaved the head of a slave and tattooed a message on his scalp. When the slave’s hair had grown long enough he was dispatched to Miletus. When the slave reached his destination, he was shaved and the message recovered. Further, in another incident Demeratus, a Greek, needed to notify Sparta that Xerxes intended to invade Greece. To avoid capture, he scraped the wax off of the tablets and wrote the message on the underlying wood. Then he covered the tablets with wax again. The tablets appeared to be blank and unused.
so they passed inspection. At that time, tablets covered with wax were used for writing messages. Since these scrolls consisted of a wooden base, a message was secretly written on this base before the base was covered with wax. So tablets appeared blank when the unexpecting sentries examined them.

Another oldest and the most fascinating and common way to hide messages is to use "invisible inks". Invisible inks have always been a popular method of steganography. These inks date as early as first century AD. It is possible to write an innocent looking letter having a secret message written between the lines, using such inks. Ancient Romans used to write between lines using invisible inks based on readily available substances such as fruit juices, urine and milk. However, milk was more often the common ingredient to write the secret message invisibly. To decode the message the letter could be heated, thus darkening the text and the secret message would magically reappear. Another way for a decoded message to remerge was to add small amounts of soot or carbon black on the paper that would stick to the milk. At later times in the history these processes have advanced as the science of chemistry progressed. Another example of such was to use gallotannic acid made from gallnuts, which becomes visible if copper sulfate is painted over it. Another reaction that would make a similar process was used during WWII. A message was written on a handkerchief using a solution of copper sulfate; it would become visible if it was exposed to ammonia fumes.

In the research sector, an early researcher in steganography and cryptography was Johannes Trithemius (1462-1526), a German monk. His first work on steganography, Steganographia, described systems of magic and prophecy, but also contained a complex system of cryptography. It was only published posthumously, as Trithemius had feared the reaction of the authorities if it was published. The earliest actual book on steganography was a four hundred page work written by Gaspari Schotti in 1665 and called Steganographica. Although most of the ideas came from Trithemius, it was a start.

Steganography continued to develop during the fifteenth and sixteenth centuries. Because they were often afraid of the wrath of powerful factions, authors of books such as historians often concealed their names in their work. A treatise on this concept was written by Bishop John Wilkins, later the master of Trinity College, Cambridge. He devised a number of schemes ranging from coding messages in music and string knots to invisible inks, described the principles of cryptanalysis by letter frequencies and argued against those who opposed publication in the field.

Major development in the field occurred in 1883, with the publication of Auguste Kerckhoffs' Cryptographic militaire. Although this work was mostly about cryptography, it describes some principles that are worth keeping in mind when designing a new steganographic system. Later, Les Filigranes, written by Charle Briquet in 1907, was a historical dictionary of watermarks.

As an interesting example of steganography of this era, many scholars suspect the authorship of the Shakespearean plays can be attributed to Francis Bacon, the noted Elizabethan statesman and writer. They back this up with the discovery of several hidden texts-steganographies - in the plays, which contain the name of Bacon. These ciphers, together with some interesting background information on Shakespeare and Bacon, makes for a convincing argument. Penn Leary, in his book "The Second Cryptographic Shakespeare" discusses this in detail.

Another form of data hiding technique also prevalent at that time was implemented through modulation of document layout to conceal/reveal information. By modulating the position of lines and words, messages could be marked and identified. Techniques such as writing messages in typewriter correction ribbon and using pin punctures to mark selected letters were used. As new technologies that could pass more information and be even less conspicuous were developed, message detection was also improved.

When photography was discovered, it allowed for an image or message to be greatly reduced when photographed. Such a case was in a Franco-Prussian war. When the city of Paris was under siege people could take photographs of their message or a letter and reduce the image to about an inch by half an inch of film. Then they wrapped this film around the legs of the pigeons and let them fly out of Paris. With the advancements in photography, in lens making and in film processing, people could resize a photo to a size of a printed period (dot), that have the clarity of standard-sized typewritten pages. This process became known as microdot technology originally developed by a German inventor. The German invention of the microdot was dubbed by FBI Director J. Edgar Hoover as "the enemy's masterpiece of espionage". Microdot photograph due to their size permits the transmission of large amounts of data, including drawings and photographs. Using this technology the message is hidden not drawing attention to itself. At the same time, microdots could store enormous amounts of information, including drawings and photographs.

However, it was during the twentieth century that steganography truly flowered. An example of this comes from early in the century, during the Boer War. The British employed Lord Robert Baden-Powell, founder of the Boy Scout movement, as a scout. His job was to mark the positions of Boer artillery bases. To ensure the Boers did
not suspect him if he was caught, he would work his maps into drawings of butterflies. Appearing innocent to a casual observer, certain markings on the wings were actually the positions of the enemy military installations. World War II marked a period of intensive steganographical experimentation. Early in the war, steganographic technology consisted almost entirely of invisible inks. An alternative data hiding method in use was open codes technique. Open codes use code words. As an example, German spies used to fake invoices that indicated a certain quantities of goods written on them. So if an invoice would order 4,000 pens from London, might indicate that there are 4 enemy ships at that location.

Later, the use of Cardan Grille and null ciphers was also widespread. Cardan Grille, a table of characters, is an encryption device invented around 1550. Gaps in the table ensure different-length words. The encoder places this grille on a sheet of paper and writes the secret message in the gaps. Then the grille is removed and a letter is formed around these secret words to produce a nice sounding letter. Here is a good example of such: "We explore new steganographic and cryptographic algorithms and techniques throughout the world to produce wide variety and security in the electronic web called the Internet". By taking out the bolded words out of the letter, we get a secret message: "Explore the Wide Web".

Null ciphers (unencrypted messages) were also used to hide secret messages inside an innocent looking container. The null cipher, which often appeared to be an innocent message about ordinary occurrences, would not alert suspicion and would thus not be intercepted. For example, a German spy sent the following message during WWII: "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 taking at the second letter in each word reveals the following secret message: "Pershing sails from NY June 1."

Other forms of hidden communication include semagrams. Semagrams do not use writing to hide the messages. For instance during WWII a shipment of watches was intercepted where each watch was stopped and showing different time. In fear that there could be a message hidden using this technique, position of the hands on all watches was changed before the shipment continued on its way.

Another important task that can be achieved with steganography is the image-downgrading problem. In multilevel security systems, such as the ones used by the military, it sometimes becomes necessary to declassify some information from a high level of access to a lower level. Unfortunately, downgrading of images can present a problem. Information could be covertly hidden in a "top secret" image for later retrieval when the image is declassified. Unfortunately, downgrading of images can present a problem. This problem has been pointed out by Kurak and McHugh.

**Steganography-Kerckhoff principle**: Recall that steganography simply takes one piece of information and hides it within another. Often, although it is not necessary, the hidden messages will be encrypted. As such a good steganography system should, in principle, fulfill the same requirements posed by the "Kerckhoff principle" in cryptography. This principle states that the security of the system has to be based on the assumption that the "enemy" has full knowledge of the design and implementation details of the steganographic system. The only missing information for the "enemy" is a short easily exchangeable random number sequence, the secret-key. Without this secret-key, the "enemy" should not have the chance to even suspect that on an observed communication channel, hidden communication is taking place. Most of the software in use today meets this principle.

When embedding data, Bender et al. reminds us that it is important to remember the following restrictions and features:

- The cover data should not be significantly degraded by the embedded data and the embedded data should be as imperceptible as possible. (This does not mean the embedded data needs to be invisible; it is possible for the data to be hidden while it remains in plain sight.)
- The embedded data should be directly encoded into the media, rather than into a header or wrapper, to maintain data consistency across formats.
- The embedded data should be as immune as possible to modifications from intelligent attacks or anticipated manipulations such as filtering and resampling.
- Some distortion or degradation of the embedded data can be expected when the cover data is modified. To minimize this, error-correcting codes should be used.
- The embedded data should be self-clocking or arbitrarily re-entrant. This ensures that the embedded data can still be extracted when only portions of the cover data are available. For example, if only a part of image is available, the embedded data should still be recoverable.

The mechanics of steganography: Hiding data, or, more appropriately, the art of hiding information within other messages or images, is basically achieved by replacing unused bits of a file with bits of hidden messages. Steganography takes advantage of these areas, replacing
them with information (encrypted mail, for instance). The files can then be exchanged without anyone knowing what really lies inside of them. An image of a grating card from “Out of Africa Safari” (as earlier noted in Fig. 2) might contain a private letter to a friend, a secret message, or a very dangerous terrorist hit list. Or a recording of a short sentence might contain your company’s plans for a secret new product.

In modern steganography images are represented in computers as an array of numbers that represent light intensities at various points (or pixels). If a color image is used, then there is such an array for each of the three primary colors, red, green and blue (RGB). Colored image is obtained by superposing these three arrays; each pixel is the sum of these three colors. Since computer files, images, etc. do not use all of the bits inside the file to store the data, an idea of data hiding comes about. One could replace the least significant bit of the original image with the secret bits and the image will not be distorted. The algorithm on how this can be accomplished is described later in the paper. For example, using most popular image size (640×480 and 256 colors) you can hide 300KB worth of data.

Technically speaking: pixels, bits, binary and hexadecimal (hex): Before discussing the kernel elements of steganographic application and implementation, we want to review the technical description of pixels, bits, binary and Hex. A pixel (picture element) is a point in an image. Graphic video cards display images on monitors made up of thousands or even millions of pixels. The pixels are so small and jam-packed that, at a distance; they appear as one image on-screen. The number of bits used to represent each pixel decides how many colors can be displayed. Using an 8-bit mode means that eight bits are used for each pixel and the more bits that you use per pixel, the better is the quality of the image. For example, 32-bit true-color video cards use 32 bits per pixel, enabling them to display more than 16 million different colors.

A bit is actually an abbreviation for binary digit, which represents the smallest unit of information on a computer. A bit is comprised of value 0 and 1. On a similar note, a byte is eight consecutive bits. Data entered into applications running on a computer commonly use decimal format. Decimals are numbers that we use in everyday life that do not have points in them (e.g., 1, 16, 26 and 30) - any random number. After decimal numbers are entered into the computer, the computer converts them into binary format, 0s and 1s, which basically correlates to electrical charges - charged versus uncharged. Internet Protocol (IP) addresses, for example, are subnetted and calculated by using binary notation.

The eight bits that make a potential number are equated in the following manner (Table 1).

<table>
<thead>
<tr>
<th>Digit No.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Binary</td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
<tr>
<td>Decimal</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^{23}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>255</td>
</tr>
</tbody>
</table>

Table 2: Bit to decimal conversion

<table>
<thead>
<tr>
<th>Bit</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>$2^7$</td>
<td>$2^6$</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
<tr>
<td>Decimal</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^{23}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>224</td>
</tr>
</tbody>
</table>

Table 3: Binary to decimal to hex conversion

<table>
<thead>
<tr>
<th>Bits</th>
<th>11100000</th>
<th>~</th>
<th>1110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Hex</td>
<td>E</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Hex comparison chart

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
<tr>
<td>16</td>
<td>0001000</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>0001001</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>0001010</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>0001011</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>0001010</td>
<td>14</td>
</tr>
</tbody>
</table>

Let’s look at the number 224. If a bit is used it is charged with 1, otherwise, use 0, as shown in Table 2.

Add the decimal values of the 1s bits: $11100000 = 128 + 64 + 32 + 224$. The binary value 11100000 equates to the decimal value 224.

Examining the mechanics of hexadecimal: The hexadecimal system is a form of binary shorthand. Internet-working equipment, such as routers, use this format while formulating headers to easily indicate Token Ring numbers, bridge numbers, networks and so on to reduce header sizes and transmission congestion. Typically, hex is derived from the binary format, which is derived from decimal. Hex was designed so that the eight bits in the binary, e.g., 11100000 (decimal = 224) equals to only two hex characters, each representing four bits.

To clarify, take a look at the binary value: 11100000 (decimal 224) (Table 3). In hex, you break this eight-bit number into four-bit pairs, i.e., 1110 0000 = 1110 0000. Each bit in the four-bit pairs has decimal value, which according to Table 1 and 2.
Table 5: Decimal to binary to hex conversion

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>185</td>
<td>1011001</td>
<td>F9</td>
</tr>
<tr>
<td>224</td>
<td>11100000</td>
<td>E0</td>
</tr>
</tbody>
</table>

Table 6: In the RGB color three are total of $256 \times 256 \times 256 = 16,777,216$ possible

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>G</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>Blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>White</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Black</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

To convert this to actual hex (Table 4). Using this chart, the hex conversion for the decimals 14 and 0 (14 for the first 4-bits pair and 0 for the second 4-bit pair), gives E0 in hex.

Next let us convert the decimal number 185 to binary, using the above procedure (Table 5).

Examining the binary of hidden messages: All computer-based pictures are composed of an array of dots (pixels (A pixel is an instance of color, a point in a picture)) that make up a very fine grid. Each one of these pixels has it own color, represented internally as separated quantities of red, green and blue (RGB). Within Windows, each of these color levels may range from 0 (none of the color) to 255 (a full amount of the color). A pixel with a red, green, blue (RGB) value of 000 is black and one with a value of 255 255 255 is white (Table 6).

For example, suppose we have a 24-bit image $1024 \times 768$ (this is a common resolution for satellite images, electronic astral photographs and other high resolution graphics). This may produce a file over 2 megabytes in size ($1024 \times 768 \times 24/8 = 2359,296$ bytes). All color variations are derived from three primary colors, red, green and blue. Each primary color is represented by 1 byte (8 bits). 24-bits images use 3 bytes per pixel. If information is stored in the least significant bit (LSB) of each byte, 3-bits can be stored in each pixel. The “container” image will look identical to the human eye, even if viewing the picture side by side with the original. Unfortunately, 24-bit images are uncommon and are quite large in size. They would draw attention to themselves when being transmitted across a network. Compression would be beneficial if not necessary to transmit such a file. But file compression may interfere with the storage of information. Hence, the usage of 24-bit image should be avoided in steganographic application.

An alternative and common image size is 640 by 480 and 256 colors (or 8 bits per pixel). Such an image could contain about 300 kilobits of data. When working with 8-bit images however, one must be more cautious about the amount of data encoded and the methods used to do so. Often, steganographic software will attempt to tweak the color palette of the image in question, but this can result in artifacts such as patches of blue in a region of mostly reds, which can easily enough away the presence of an image.

An image could be compressed using one of the two compression algorithms: lossless or lossy. Both methods save storage space but have different results. Lossless compression of an image is stored in either GIF (Graphical Interchange Format) or in BMP format (Windows Bitmap). This compression lets a receiver reconstruct the original message exactly. Therefore, this compression would be used when it is desired for original information to remain intact.[1]

To be able to use GIF image technique of steganography, we need a graphic format that support image with a palette of color. GIF is certainly not the best graphic format, but it is well known and supported (even if it uses a patented compression algorithm). As a matter of fact the algorithm will have to decode a GIF file, apply our hiding algorithm and finally generate a new GIF file with a permuted palette. For the decoding process, we only need to read the palette of the GIF file, so this can be done in a more easy way.

Another type of compression is a lossy compression. JPEG (created by Joint Photography Experts Group) image is an example that uses this type of compression. The latter image format saves space but may not maintain the original message integrity. User needs to keep in mind that by using this type of compression on an image, some information may be lost. Neil F. Johnson performed an experiment testing how each type of graphics compressions modifies the size and quality of the file[13]. Table 7 summarizes this experiment.

Another concern is how to deal with images that use lossy compression such as JPEG or PNG format. There are many algorithms that use Fast Fourier analysis to determine properties of the spatial frequencies of source images that can be used to encode information in the visible image without affecting perception of it. Another method is a Patchwork approach that embeds the same small message repeatedly in an image in order to ensure redundancy. Interestingly enough, similar methods can be used to provide digital watermarking capabilities.

Table 7: Various photographic file format

<table>
<thead>
<tr>
<th>Image type</th>
<th>JPEG</th>
<th>BMP</th>
<th>GIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>175,808 bytes</td>
<td>2.64 MB</td>
<td>775,252 bytes</td>
</tr>
<tr>
<td>Image size</td>
<td>1073 x 790</td>
<td>24-bit</td>
<td>8-bit</td>
</tr>
<tr>
<td>No. of colors</td>
<td>16 mil colors</td>
<td>16 mil colors</td>
<td>256</td>
</tr>
</tbody>
</table>

Where are those data hidden?: The data we want to hide are stored into the palette of an image file. The picture is in no way modified; just the order of the color within the
Fig. 3: Implementation of steganographic hiding techniques

palette is slightly modified in order to keep our secret message. The compression or size of the file won't be changed at all. A viewer won't be able to know what have changed. Of course, someone with the program will be able to retrieve the secret message. The only clue is that someone looking at the right place might notice that the order of the color within the palette is a bit unusual (at least not sorted). So it is important to add an element of encryption.

There are usually two type of files used when embedding data into an image: the innocent looking image which will hold the hidden information is a "container" file and the "message" file that contains the information to be hidden. A message may be plain-text, ciphertext, other images or any thing that can be embedded in the least significant bits (LSB) of an image.

Digital steganography is based on the fact that artifacts like bitmaps and audio files contain redundant information. That's why lossy compression techniques such as JPEG and MP3 work. Such techniques eliminate part of the redundancy, allowing the image or wave file to be compressed. The idea behind steganography is that instead of eliminating the redundant information, you replace it with other data.

For example, one can hide a message in three pixels (24-bit colors). Suppose the original 3 pixels are:

\[(00010011 11101001 11011000) (00100011 11001000 11101000) (11001000 00100111 11100101)\]

A simple steganographic program could hide the letter "A" which has a position 65 into ASCII character set and have a binary representation "01000001", by changing the least significant bit in each of the first eight bytes to reflect the binary letter "A". The result (affected bits in bold):

\[(01100111 11101001 11011000) (01001001 11001000 11101000) (11001000 00100111 11100101)\]

Note that we did not have to change all of the LSB's (undetected), just those that didn't match, which means that on average 50% of the pixels of an image will not be affected by embedding.

Examining the mechanic of the hex of a hidden message:
As discussed above, an encoded image or file can replace unused bits of a file with bits of a hidden message. Figure 3 shows a picture of a screen dump from software used to implement image-hiding technique, which was used to hide message within an aerial photo of earth as the cover image. Figure 4a and b show the image (cover) of earth plain image without and with hidden message, respectively. By using any hex editor we can clearly view/ read the plaintext message to be embedded (Fig. 5). Hence, one must take some precaution, which would include password generated encryption option, ranging from simple substitution (permutation) algorithm to strong encryption algorithms like DES, RSA etc. Obviously, the file size of the image (Fig. 4b) increases to accommodate the hidden message. What's more, the message is encrypted - to further protect it from prying eyes. Moreover, Fig. 4 demonstrates that when this technique is properly applied, its effects on the resulting image are almost impossible to detect. You could receive a message embedded in a graphic, but no one else could make out more than an image.
The secure mechanics of hiding data: If you have something to hide and want to protect the message at the same time, then you better have to send an encrypted hidden message and, make sure your encryption algorithm put as few structures as possible into the encrypted message, so that it will be difficult to recognize that this is an encrypted message. Therefore, not attracting any undue attention from the unwarranted persons. Permutation algorithm which leaves the text in human readable form fits the bill.
The mechanics of permutation algorithm: The substitution or reduction of color via addition byte/character can be easily implemented via permutation algorithm. Permutation or transposition technique involves rearrangement of the characters of a plaintext message to convert the message into an anagram that looks like a message with random characters. For example, here is a permutation of \{1,2,3,4,5\} to \{4,5,1,2,3\}, which can be used to transform (encrypt) a string of five characters by moving each character from its original position to the position defined by the permutation. For example, the string “APPLE” would be translated to the string “PLEAP”. What about decrypting the text? Well permutation has an inverse, another permutation that does the exact opposite of the original permutation. For example, the permutation \{4,5,1,2,3\} has an inverse \{3,4,5,1,2\} which should unscramble the string “PLEAP” to get back the original string “APPLE”. In real-time application, for example, most messages consist of 7-bit ASCII characters or 8-bit in hexadecimal which we can represent with integer numbering (01234567) according to Table 8, to give permutation representation e.g., \{7,5,4,6,3,0,2,1\}. By scrambling the bits to create a random set of bits (using inverse of our permutation i.e., \{5,7,6,4,2,1,3,0\}) you can get the desired encryption. For example, an 8-bit character, the byte/character ‘O’, which have position 71 into ASCII character set and have the binary representation ‘0100 0111’ can be encrypted/decrypted (Table 8).

In using the permutation algorithm, the hidden message will be put inside the palette part of a graphic file as previously performed. As the order of the color into the palette is not important for the rendering of the image, data can be hidden into the permutation of the color inside this palette. For graphic file that use a palette, this palette is a table of color. Then, every pixel is represented as an index to this table. The trick is to change the order of the color into the palette and also change every reference to the new index after permutation.

So we will encode our hidden message into a permutation. With a palette of 256 colors, we have factorial of 256 possible permutations (256). As we will translate a message into a permutation, we will be able to hide as much as factorial of 256 possible different messages. All of that message will have an associated permutation. We will be able to also make the reverse mapping, from the permutation to the message.

As we need to be able to identify the permutation that has been used, we need a reference order for the color into the palette. So before you apply a permutation, you need to “sort” the palette into a reference representation. From there, you can apply the permutation to the new-sorted palette. Your message has now been hidden.

On the other side, to get back the message, we will have to sort the palette and figure out the permutation to get from the sorted palette to the palette we received. Some kind of inverse permutation is required. In using the inverse mapping we can decode back the message that was hidden and it will be clear text for us (Table 8).

A few question still need to be answered to fully understand how it actually works within our implementation. In the following sections we will see how we can get from a message to a permutation and then back to the message again. We will explain in detail the process and algorithm to implement this technique.

| Table 8: An example of permutation algorithm for encryption/decryption |
|------------------|------------------|------------------|
| byte/character 'O' | Permutation   | Inverse permutation |
| Binary           | 0100 0111       | (5,7,6,4,2,1,3,0) |
| Decimal          | 1110 0100       | (5,7,6,4,2,1,3,0) |
| Hex              | 71              | 71               |
|                  | 228             | 71               |
|                  | 47              | 47               |

From a message to a number (and from this number back to a message): To stay as general as possible, we need to assume that the message is a stream of bit at zero and one and size of that stream is not fixed. If the message is a string, then we will use binary representation of all those character from the string. From a stream of bit it’s easy to get to a number, we just have to consider that we have the binary representation of our number. We also need to be able to differentiate between two equivalent and different message: 110010 and 0110010 that both map to 50. In fact we need to encode the length of the message to be able to get back to the original one. To do this we will always put a leading one (1) in front of this stream of byte. On the other side we just have to remove this extra bit. So the above messages will be coded as 1110010 and 10110010, which map, respectively to 114 and 178. So we only use one bit to encode the size of the message.

Most of the time, our message is only made of stream of byte and then we will put them in the Most Significant Bit (MSB). Again let us consider the byte/character ‘A’, which have position 65 into ASCII character set and have the binary representation ‘01000001’, will be encoded 101000001 (so 321 which is 65 + 256). As a matter of fact, the empty message will be encoded as 1. Because there is no message associate with the value 0, we will always remove one to the encoding of the message. So now the empty message will be a value of 0 and ‘A’ will be a value of 320.

So on a practical point of view we start with one into an accumulator and for every byte/character from the message, we add the value of that byte to the accumulator.
multiply by 256 (except for the last byte). At the end we remove one from the big value we get. To decode the message we first add one to the value. Then we just have to compute the remainder from dividing by 256, which is the last character. Then divide again (and keep the remainder) for the previous one and again until we get one or zero. If we get one then this is a message with a size multiple of 8. Otherwise, the last byte we get is to be decoded into a certain number of bits from one to seven to be further analyzed.

**From a message to a permutation:** We will put an extra step between a message and the permutation. We will number all the permutation and we will use an algorithm that will generate from any number between 0 and n-1, a permutation. Also, from the permutation we need to be able to reverse the process. Now we need to map a message to a number between 0 and n-1. And we need to reverse the process. A permutation is always a permutation of something, so we need to know how much ‘distinct’ element we have available. So how much color we have in the palette. We will suppose that the palette contains only unique color. We also need to make sure the number we want to encode is lower than factorial of this number of element. We won’t work with permutation of a set of element where you have duplicate because in a palette of color you should not find any duplicate. But it could be interesting to work with duplicate algorithm in future research work.

**Adding strong cryptography:** From above we can see that the permutation algorithm leaves the embedded message in human readable text, which is prone to cryptanalyst technique. As a matter of fact, a steganographic technique should also provide a built-in encryption algorithm so that the knowledge of the technique is not the only secret (as for security by obscurity). If you want to compress your message, make sure that is the first operation you do, then only encrypt and finally hide the result into a graphic file. Because encryption introduce confusion within your data, it will be very hard to compress afterwards.

However, we can argue on whether or not encryption should be part of the steganographic process. Steganography isn't meant to replace cryptography, but to complement it; its purpose is to avoid raising suspicions. Let’s return to the invisible ink example, suppose I was having an affair with my maid (let’s name her Alice). I want to tell Alice how beautiful she is, but don’t want my wife to find out. I could write Alice a love letter using invisible ink. Switching to visible ink, I could then write another note (perhaps asking her to pick up my laundry and to remind my wife that I would be coming late tonight) over the secret message. Should my wife find it, she would see only a banal exchange about housekeeping matters and my turning up late. Alice, expecting more, would hold my note over a candle to expose the hidden message.

If we return to the case of cheating on my wife, suppose I was concerned that my wife might already be suspicious; I would take further security steps, such as using a less easily discovered ink. Like a suspicious wife looking for secret messages, analysis techniques can penetrate a simple bit-swapping (permutation) scheme presented earlier, which leaves the ciphertext message in human readable characters has detectable patterns.

A much realistic case is where Alice is a dissident in a land where the government suppresses free speech and monitors all communications. Bob is an activist living in a freer state and is trying to tell Alice’s story to the world. Luckily, Alice has an Internet connection and can send e-mails to Bob, but her government is advanced enough that they can monitor e-mails as well. Alice does not despair however; she managed to get her hands on a contra-band copy of Pretty Good Privacy (PGP) cryptosystem and can encrypt all of her e-mails, much to the chagrin of her local spy agency. Unfortunately for Alice, though she has managed to maintain a low-profile with them in the past, her use of encryption has flagged her and her communications within the spy agency’s computers as “suspicious”. Alice is arrested and charged with suspicious behavior and Bob never hears from her again. The story above may sound somewhat Orwellian, but the underlying moral is one that cannot be ignored. In a world where most people do not encrypt their communications, the use of encryption can easily mark a person as having something to hide. There is hope for Alice (and us all) however in steganography, a millennia old technique for hiding information under a seemingly innocuous cover.

In such circumstance, one can always include a “build in” encryption at almost no cost. Rather than use the sorted palette as a reference permutation of the palette, we can start with a sorted palette and apply a secret permutation based on a secret message. Then only we apply the “hidden text” permutation. This is like hiding two messages at the same place. That is, one message on top of the other one. Like an XOR between the hidden message and the secret message. Of course the secret message should be changed often, otherwise it will loose its forte. Also if the clear text and hidden message are to be known, then the secret is not a secret anymore. Maybe the secret (or encryption) message should be generated from a random number generator based on a secret key and a cryptographic algorithm.
Fig. 6: Illustration of image hiding technique using steganography

The exclusive OR is an example of an encryption function, which is very popular method for performing simple block encryption[12]. The XOR-function is used to indicate that if there are two conditions (say condition A and condition B), then either condition A is true or condition B is true, but not both. The complete set of possibilities for two values being XORed and their result is as follows:

\[
\text{XOR}(0,0) = 0; \text{XOR}(0,1) = 1; \text{XOR}(1,0) = 1; \text{XOR}(1,1) = 0
\]

The best thing about the XOR-function is that it can be used to reverse itself and can therefore be used for encryption purposes. Suppose that we take the values: \(A = 10101000\) and \(B = 00111001\). Therefore, \(C = \text{XOR}(A,B) = 10010001\). Now if we take B and XOR it with C, we will obtain: \(A = \text{XOR}(B,C) = \text{XOR}(00111001, 10010001) = 10101000 = A\). For complete software implementation of XOR-function[12].

The implementation of steganographic algorithms: The image formats typically used in such Steganography methods are lossless so that the data can be directly manipulated and recovered. The transform domain tools include those that involve manipulation of algorithms and image transforms such as discrete cosine transformation and wavelet transformation. These methods hide messages in more significant areas of the cover and may manipulate image properties such as luminance.

There are many algorithms and tools to embed or to hide data into images. Most Stego tools are hybrids which use set properties of classical image and transform domain models. Many of these are freeware and may be downloaded from the net without spending anything! 'Stego DOS', 'S-Tools', 'Mandelsteg', 'EzStego', 'Hide and Seek', 'White Noise Storm', 'Steganos', 'Picture Marc', 'JK-PGS', 'Sys Cop' and 'Sure Sign' are some of the names of Steganography tools which can be easily procured through the Net by anybody.

You can hide the data using the above softwares and still the human eye would not be able to recognize the difference, see Fig. 6 obtained with S-Tool software. The image on the left is the original container, while image in the center contains a message displayed on the right. The point of displaying these images is to illustrate that the difference between the container and the message embedded into the container is not detectable by the human eye. Although these images are not displayed to scale, they do demonstrate the idea of what can be embedded inside the “container” image. To further illustrate this point, you may refer back to the “flow chart” on how steganography software works (Fig. 2).

Most steganography software, however, neither supports nor recommends the usage of JPEG images. Instead, use of lossless 24-bit images is recommended[13]. Hide and Seek software, like most steganographic tools above, uses GIF and BMP images to hide information. When a message is embedded inside a smaller sized container, black stuffing pixels are added to the container image to make it the acceptable size of 320×480 pixels. Other software might stretch the container; instead, so all the pixels are filled with information. Since it takes 8-bits to represent one character and 8-bits to represent one pixel, container that is 320×480 pixels could store maximum of 19 kilobits of data.

The most common approaches to hiding information in digital images are masking and filtering, algorithms and transformations and least significant bit (LSB) insertion. We will discuss, in detail, how the LSB insertion algorithm is implemented a bit-streaming. This algorithm provides a common, simple approach to embed information in a container file. According to Neil Johnson's “Steganography: Seeing the Unseen” the following claim is made regarding the LSB: “Unfortunately, LSB is vulnerable to even a slight image manipulation. Converting an image from a format like GIF or BMP to JPEG could destroy the information hidden in the LSBs”. Nevertheless, LSB is still the most common algorithm.
used in steganography software today\(^{21}\). On average, LSB requires that only half of the bits in an image be changed. You can hide the data in at least and second least significant bits and still the human eye would not be able to recognize the difference (Fig. 6).

Hide and Seek software explained here uses LSB algorithm that is based on a private-key. This private-key is used with pseudorandom permutation of container bits to hide the data. The permutation function must depend on a secret-key K. Therefore, we need a pseudorandom permutation generator. To respect the Kerchoff’s Principle, this generator must be secure, that is nobody can guess the container bits where information is embedded without knowing the secret stego-key K\(^{22}\). Using this function the data is inserted by modifying the chosen bits. Let \( N \) be the number of container bits available and let \( P_0^N \) be a permutation of the numbers from 0 to \( N-1 \). Having a message with the length \( n \), we can hide the secret message into the container bits as follows:

\[
P_0^N (0), P_0^N (2), K, P_0^N (N-1)
\]

This function allows us to select bits in an apparently random order. As a result, the hiding bits will be spread all over the container.

This process of using a key can be considered as a black box that produces different unpredictable functions for each secret stego-key value. It can be easily constructed from any secure hash function \( H \) by concatenating the argument \( I = (I, 0,1,K, n-1) \) with the secret-key \( K \), where, \( K+i \), is the concatenation of the key \( K \) and the argument \( I \). Let \( f_i(i) = H(K\cdot i) \) so that we obtain a pseudorandom function \( f_i(i) \) that depends on the key \( K \).

Richard Popa describes the following scheme using pseudorandom function generator\(^{24}\). The \( a \oplus b \) operation denotes the bit-by-bit exclusive OR of \( a \) and \( b \), the result having the length of \( a \). Having the binary argument \( i \) of length \( 2\lambda \), we divide it in two parts, \( A \) and \( B \), of length \( \lambda \) and the key \( K \) is divided into four parts, \( K_1, K_2, K_3 \) and \( K_4 \). The scheme works as follows:

\[
Y = Y \oplus f_{K_1} (X)
\]

\[
X = X \oplus f_{K_2} (Y)
\]

\[
Y = Y \oplus f_{K_3} (X)
\]

\[
X = X \oplus f_{K_4} (Y)
\]

return \( Y \cdot X \)

Running the algorithm \( 2^{24} \) times yields a pseudorandom permutation of \( \{0,1,K,2^{24}-1\} \). The author of this scheme shows that the permutation is as secure as the pseudorandom function generator.

The following algorithm is based on the scheme above. Having the image \( I \) with the dimensions \( x \) and \( y \), to get the index of the \( i \)th hidden bit the sequence is:

\[
Y = i \div x
\]

\[
X = i \mod x
\]

\[
Y = (Y + f_{K_1}(X)) \mod y
\]

\[
X = (X + f_{K_3}(X)) \mod x
\]

The returned values \( Y \) and \( X \) are the coordinates where the \( i \)th secret bit will be inserted \( K_1, K_2, K_3, K_4 \). Having a 800×600 image, a secret message of 1 Kbytes and the key \( K = 123,456,789 \) then the container image \( N \) consists of 480,000 bits and the secret message consists of \( 1024 \cdot 8 = 8192 \) bits. This means that less than 2 percent of the pixels will be altered. This obviously will not make the distortion of the original container visible to a human eye. Thus, to hide the 500th bit, the algorithm for finding its position is:

\[
Y = 500 \div 800 = 0
\]

\[
X = 500 \mod 800 = 0
\]

\[
Y = (0 + H(123\cdot 500)) \mod 600 = 7566 \mod 600 = 366
\]

\[
X = (500 + H(456\cdot 366)) \mod 800 = (500 \cdot 3562) \mod 800 = 62
\]

\[
Y = (366 \cdot H(789\cdot 62)) \mod 600 = (366 \cdot 1563) \mod 600 = 129
\]

The result is 129×800, that means that the 500th bit will be embedded into the LSB of the container pixel whose \( x \)-coordinate is 129 and \( y \)-coordinate is 62.

It should be noted that this scheme could not be used to embed long messages without degrading the picture. The size of the message that could be embedded in the container image is dependent on the container and its size, as well as variations of intensities in the neighboring pixels. Although there are possible attacks on this algorithm, one could add extra level of security to this algorithm as explained earlier, as well as, error detection-correction mechanism. But this will only work in cases where a lossless compression algorithm is used.

Tools such as this will help steganography gain momentum in cryptography world. When popular encryption mechanisms are used with steganographic techniques in data hiding, this definitely adds an extra layer of protection on the cryptosystem. Later in the text we will demonstrate how hidden channels of
communication could be established using volatile and user data properties. There are numerous techniques for using these ideas in watermarking and fingerprinting\cite{2,4,26}. We could also use steganography methods for data hiding in document images, sound files, movie files, binary files, etc. More information on these works can be found in references\cite{13}, which certainly give reasons why the field of steganography deserves a great deal of attention from computer science and cryptography communities.

**The properties of a steganographic file system:** A steganographic file system must meet three basic requirements\cite{4,13,26}. First, like any file system, it must be able to manage the storage of files on a disk. Second, it must provide a mechanism for hiding files. This allows a user to plausibly deny the number of files stored on disk. Third, it must provide a mechanism for accessing files that have been hidden. In addition, to be truly effective, a steganographic file system should exhibit the following traits:

1. Deletion of both hidden and non-hidden files would lead to secure destruction. This way, deletion of hidden data would look just like deletion of non-hidden data. Suspicion would arise if only hidden files were replaced with random characters after deletion. Furthermore, encrypted data usually consists of random characters, so suspicion also would arise if some sections of a disk contained random characters and some sections contained the remains of deleted plaintext. For all of these reasons, replacing non-hidden files with random data would help to hide the presence of hidden files.

2. Write access to hidden files between inspections by an attacker would look much like the creation and deletion of non-hidden files. Otherwise an attacker would know that hidden files have been accessed.

3. Hidden files would share the same partition as normal files. Suspicion might arise from the presence of a separate area on a disk just for hidden files.

4. The existence of hidden data would not impair the reading and writing of non-hidden data. This means that no blocks would be explicitly reserved for the secure file system. Of course, this also means that there would be a danger of hidden files being overwritten. It would be preferable if hidden files were not overwritten by non-hidden data, but there is no way to protect blocks from being written to without raising suspicion.

5. The steganographic file system would behave just like a normal file system once access is gained to the secure files. For example, after providing the name of a hidden file and the key or password for accessing it, the user would be able to use any standard software application to read from and write to the file. In a Unix environment, commands like grep, ln, sed and awk would perform just as they would for regular files.

6. The fact that steganography is being used on a machine would be concealed, or if this is not possible, there would at least be a plausible justification for the presence of such a file system. One possible approach to the first scenario would be to conceal the existence of the driver for the hidden file system, perhaps by attaching it like a Trojan Horse to some other application. A possible approach for the second scenario would be to make steganography (and cryptography) standard tools on all systems.

7. Non-hidden files would remain accessible even if the driver for the steganographic file system is removed from the system.

8. There would be several layers of plausibly deniable access. If the driver cannot be hidden, this is especially helpful. This would allow the owner of the data to reveal lower levels of secure files and perhaps compromise moderately sensitive data without providing any means of proving that higher levels of security exist.

9. There would be few if any performance issues when writing to or reading from a file. In other words, a steganographic file system would perform nearly as well as a regular file system despite the entire extra overhead of encrypting data and hiding it on the disk.

Of course, different implementations of steganographic file systems might behave quite differently from one another, but any implementation should at least consider these points in order to be effective. To understand how these points might be addressed in a real application, it is useful to examine the design of one such file system, StegFS.

StegFS is an implementation of a steganographic file system for the Linux operating system, inspired by a construction proposed by Anderson et al\cite{10,16}, but with several practical improvements. StegFS works closely with a slightly modified version of the driver for the Second Extended File System (Ext2fs), the standard Linux file system. StegFS contains the full functionality of Ext2fs and behaves almost exactly like Ext2fs when no hidden security levels are open\cite{18,29}. To create a StegFS partition on the disk, there first must be a regular Ext2fs partition on which to create it. If the StegFS driver is ever removed, the Ext2fs partition will continue to behave normally.
Fig. 7a: There are three important parts: Superblock (only one of these); Inodes (one per file); Data blocks (the rest of the disk). The super block represents the entire filesystem. Each inode represents a single file, these belong to the system; they have fixed format and you cannot read or write them directly. The data blocks belong to the user and that’s where the data is stored.

Fig. 7b: Ext2fs is a little different. There are some other structures between the superblock and the inodes. There are bitmaps recording which blocks are in use and which inodes, one bit per block or inode. The disk is portioned into ‘block groups’. Each block group has a copy of the superblock and bitmaps, some of the inodes and some data blocks. That way if the superblock or bitmaps are destroyed they can be restored from a copy. So the picture above shows a single block group instead of the entire disk.

Fig. 8: Schematic representation of inodes

The mechanics of second extended file system: To understand the design of StegFS, it is important first to understand the design of Ext2fs and review important terminology and concepts dealing with file systems in general[21-28]. The first important thing to understand about Ext2fs is that it divides the physical disk into pieces called blocks. A block is the smallest unit of space on the disk that can be allocated or marked as used or free. The size of each block, usually 1024 bytes, is determined when the file system is created. A fixed number of sequential blocks are gathered together to form block groups (Fig. 7).

Within each block group, most of the blocks are used for storing data from files (Fig. 7). These blocks are called data blocks. In addition, some blocks are set-aside for special purposes. For example, a portion of each block group is set aside for inodes. The numbers corresponding to these inodes can be determined by the number of the block group. Likewise, the location of a particular inode on the disk (that is, the block group in which it is located) can be determined by the inode number. A portion of each block group also is set aside for a block allocation bitmap table, or bitmap block. This table contains one bit \(e \{0,1\}\) for every data block in the block group. A portion also is set aside for an inode bitmap table, which contains one bit for every inode in that block group. These bitmap tables help the file system determine which blocks and inodes in that block group are free and which have been allocated. Finally, each block group also contains redundant (extra) copies of important file system to control information storage. This makes it possible to recover a file system if the primary superblock becomes corrupted.

Another important Ext2fs concept to understand is that of an inode (Fig. 8). As mentioned above, an inode is just a special type of block on the disk. An inode block
The mechanics of steganographic file system: StegFS:
The design of steganographic file system (StegFS) must meet three basic requirements: (i) it must be able to manage the storage of files on a disk; (ii) it must provide a mechanism for hiding files and, (iii) it must provide a mechanism for accessing files that have been hidden. We will discuss each of these in turn.

The StegFS is meant to replace both hidden and non-hidden files with random data to help hide the presence of hidden files and to make hidden files appear as normal files that have been deleted. Hidden files share the same partition as normal files. No blocks are explicitly reserved for the secure file system, so the existence of hidden data does not impair the reading and writing of non-hidden data. Of course, this also means that hidden files can be overwritten. StegFS addresses this problem by making several copies of the inodes and data blocks for each file. StegFS also behaves like a normal file system once access is gained to the secure files. Finally, non-hidden files remain accessible even if the StegFS driver is removed from the system.

The table shows the name of the files and the corresponding inode number:

<table>
<thead>
<tr>
<th>Inode</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name 1</td>
</tr>
<tr>
<td>2</td>
<td>name 2</td>
</tr>
<tr>
<td>3</td>
<td>name 3</td>
</tr>
<tr>
<td>4</td>
<td>name 4</td>
</tr>
</tbody>
</table>

Fig. 9: Schematic representation of Inode table and its link to directory

contains information about one particular file, along with pointers to the data blocks that store the contents of that file. Every inode is numbered and every file is assigned its own unique inode number to identify the inode for that file.

The way the Ext2fs file system locates a file on disk is by looking up the filename and inode number in a directory file (Fig. 8 and 9). A directory file in Ext2fs contains the names of the files in that particular directory, the type of each file and the inode number for each file. Once the inode number is identified, Ext2fs looks in the inode to find pointers to the data blocks where the file is stored. When a file is deleted, the inode for the file and the data blocks that stored its contents are marked as unallocated (in the inode bitmap table and block bitmap table, respectively). They then are free to be used by other files. Normally, nothing actually gets erased from the disk. Instead, a new file eventually overwrites the data in those blocks. Having gained a basic understanding of Ext2fs, we can now examine the design of StegFS. Since StegFS basically is just an extension of Ext2fs, many of the terms and concepts will be very similar to those just discussed.

The block table in StegFS is a regular, non-hidden file that contains one entry (128 bits long) for every block on the disk. This block table is present whenever the StegFS driver is installed on a machine, whether StegFS is being used or not. The table serves several functions. For one, it acts as both a block allocation bitmap and an inode allocation bitmap. If any one of the first 47 bits of an entry is not set to 0, the file system knows that the corresponding block is not in use for either purpose. However, if the first 47 bits all are set to 0, things change. In this case, if the 48th bit also is set to 0, this means the corresponding block is a data block currently in use by StegFS. If the 48th bit is set to 1, the corresponding block is being used as an inode. If the corresponding block is being used as an inode, the last 32 bits of the entry for that block will provide the inode number for the inode in that block.

The block table also is used to check the integrity of hidden files on the disk. If the block is being used as a data block, bits 49 through 96 in the entry for that block are used to determine a checksum for the data stored in that block. This checksum is very important to StegFS because there is no way to prevent blocks containing hidden data from being overwritten by non-hidden files.
Doing so would give away the fact that the block contains hidden data. Because of this, whenever a hidden file is written to disk, StegFS makes several copies of that file at random locations on the disk. Of course, this introduces other management issues.

One issue related to the copying of files on the disk has to do with inodes. Recall that in Ext2fs, an inode contains information about a file along with pointers to the blocks that store the data for that file. In StegFS, an inode also contains information about a file. However, instead of pointing to one set of blocks, a StegFS inode contains pointers to every block containing a copy of the data for that file. Another management issue related to inodes is that the inodes themselves have to be hidden to avoid revealing information about a hidden file. Furthermore, because the inode is hidden, there is a danger that the block containing the inode could be overwritten by non-hidden data. This means that the inode for a file also has to be duplicated. StegFS handles this by creating, for every inode, several entries in the block table.

Now that we understand how StegFS manages the storage of files on the disk, we can look at how StegFS allows data to be hidden in one of 15 separate security levels, with 1 being the lowest and 15 being the highest. For each of these security levels there is a pass phrase that gives access to that and every lower level and every hidden file on the system belongs to one of these 15 levels. The hidden files are stored in directories such as /stegfs/1/, /stegfs/2/, etc., which are only available when the given security level has been opened. When the user creates a new hidden file in one of these directories, StegFS first encrypts that file using the key for that security level. StegFS then uses a random number generator to determine the first block on the disk that will be allocated for the first copy of that file. Assuming the file spans more than one block, StegFS will try to store that file in sequential blocks as much as possible. Every other copy of that file then starts from its own random block somewhere else on the disk, but also is stored in sequential blocks.

It is important also to add an extra security component which involves encryption of the data file before performing StegFS algorithm. The reason why the file is encrypted before being written to disk is two-fold. The first and most obvious reason is to protect the contents of the file from being examined. The second and less obvious reason is that most encryption schemes usually produce output consisting of random-looking data. Since StegFS fills the empty blocks on a partition with random data at the time the partition is created, there is no way for an attacker to differentiate between random data and encrypted data on the disk. Furthermore, when StegFS is running, deleted regular files are overwritten with random data, so even those blocks look the same as all the others.

Before StegFS actually allocates a block to a hidden file or inode, it first checks the Ext2fs bitmap to see if a non-hidden file is already using that block. If it is not, it checks its own block table to see if a file in an equal or lower security level is using the block. Since the entries in the block table are encrypted with the same key as the data in the corresponding block and because entries corresponding to empty blocks consist of random data, there is no way to determine if the block is being used by a file in a higher security level. If no conflicts are found, the encrypted data are written to that block and an encrypted entry containing information about the data in that block is made in the block table.

In order to access a hidden file, the user first must provide a pass phrase and open the security level under which the file is stored. When the user then specifies the file to be accessed, StegFS looks in the proper directory file to find the inode number. Since the block location of StegFS inodes can not be determined by their number alone, StegFS uses its block table to determine the block in which that inode is stored. If StegFS determines the inode in that block has been overwritten (i.e., does not match the checksum), it looks for the next copy of that inode in the block table until it finds an inode that is intact. By examining the inode, StegFS can locate the data blocks that contain the data for that file. If the checksum test again fails, StegFS examines backup copies of the data blocks until intact data can be found.

Steganography in network model architecture: In the first branch of steganographic techniques discussed above and cover protection against detection, we have already seen that there are several ways we could use steganography to hide the information in user data e.g., image file etc. Another important steganographic technique involves information hiding in volatile data. Information hiding in volatile data is referred to as data hiding in network model architecture in a way so that covert channels of communication can be established. Information hiding in user data, unlike volatile data is stored on the user’s hard drive or in memory. This can be used later by authorities to prove that secret communication took place between Alice and Bob if this data is not erased.

Steganography in communication network is closely related to the problem of “hidden channels” in secure operating system design, a term which refers to all communication paths that can not easily be restricted by access control mechanisms (e.g., two processes that communicate by modulating and measuring the CPU
Steganography in network is also closely related to spread spectrum radio transmission, a technique that allows us to receive radio signals that are over 100 times weaker than the atmospheric background noise, as well as TEMPEST, technique which analyzes RF transmissions of computer and communication equipment in order to get access to secret information handled by these systems.

Common communication systems have a huge number of characteristics and only a small fraction of what looks like noise can actually be replaced by the statistically very clean noise of a cryptographic ciphertext. Most communication channels like telephone lines and radio broadcasts transmit signals, which are always accompanied by some kind of noise. This noise can be replaced by a secret signal that has been transformed into a form that is indistinguishable from noise without knowledge of a secret key and this way, the secret signal can be transmitted undetectable. Noise in communication systems is often created by modulation, quantization and signal crossover and is heavily influenced by these mechanisms and in addition by all kinds of filters, echo cancellation units, data format converters, etc. Many steganographic systems have to work in noisy environments and consequently require synchronization and forward error correction mechanisms that also have to be undetectable as long as the secret-key is unknown.

The noise on analog systems has a large number of properties very characteristics of the channel and the equipment used in the communication system. A good steganographic system has to observe the channel, has to build a model of the type of noise which is present and, has then to adapt the parameters of its own encoding algorithms so that the noise replacement fits the model parameters of the noise on the channel as close as possible. Whether the steganographic system is really secure depends on whether the "enemy" has a more sophisticated model of the noise on the channel than the one used in the steganographic system.

However, really good steganography is much more difficult and usage of most of the currently available steganographic tools might be quite easily detected using sufficiently careful analysis of the transmitted data. Further, it is important to note that this basic design principle of steganographic systems, i.e., replacing high entropy noise with a high entropy secret transmission, is quite obvious. There are a number of simple software tools that have been published which can be used for e.g., hiding files in the least significant bits of digital images or for transforming Pretty Good Privacy (PGP) messages into files resembling pure random byte sequences and this can be implemented under the OSI system.

An Open Systems Interconnection Reference model (OSI) uses packet structures to send information across the network from one layer to another as well as from one network terminal to another. A network packet consists of packet headers, user data and packet trailers. All the packets sent across the network, following OSI model, have the same packet structure.

Here is an overview of an OSI networking model, (Fig. 10)[21,30] OSI (Open Systems Interconnection) is a standard description or "reference model" for how messages should be transmitted between any two points in a telecommunication network. Its purpose is to guide product implementors so that their products will consistently work with other products.

In order to explain how network can be used to hide data, we have to assume that Alice and Bob are on the same network and Joe is a system administrator who can observe the network traffic and of course Eve is the bad guy. We will also need to assume that sending any encrypted data over the network is prohibited. Thus, Alice and Bob would be reprimanded if Joe detects that some type of encrypted communication is passed through his network. Alice and Bob can utilize the first five layers in such a way that hidden communication can occur without drawing much suspicion from Joe.

Physical layer: At the physical layer the communication is done with the help of control signals and timing. Since there is a handshaking mechanism that controls data flow, Alice in sending message to Bob could alter the throughput by controlling the tension of the lines. Under normal circumstances, 5 volt tension represents a "1". By modifying this tension to 5.5 volts Bob would see a hidden "1" and a 5 volt tension would represent a hidden "0". This scheme would only work if a network channel between Alice and Bob does not contain bridges, routers, or repeaters. Under most networks, however, a user would encounter at least one type of signal amplification or modifying devices.

Data link layer: At the data link layer, Alice and Bob can modify the collision detection technique by setting the delay of packet's retransmission to either maximum or zero. This modification does require, however, that either Alice or Bob would jam any signal generated by a third party. This provides a bit-per-bit hidden channel.

Alice and Bob can utilize the Internet Message Control Protocol (ICMP) to establish another way of hidden communication. By using the source-quench commands they can accomplish that quite easily. The source-quench command adjusts the transmission data rate when the destination host or intermediate hop cannot keep up with a current rate of data transmission. Thus,
Alice and Bob can agree, for instance, that every tenth packet could signal a "1" or a "0" bit. To be more precise, Bob would request the retransmission of every tenth packet if he wants to send a "1". Following this scheme, if the tenth packet is not requested then there is a hidden "0" that is seen by the sender. Alice. Packet fragmentation could also be used at this network layer. Alice and Bob could use the fragment offset field as being the hidden bit.

**Transport layer:** At the transport layer, like in Network layer, additional headers are used for routing information and error checking. The Transmission Control Protocol (TCP) works at this layer of the OSI networking model. Each TCP packet segment begins with a uniformly-formatted 20-byte header. There is a six-bit field that is not utilized by the TCP packets. All these unused bits could be used to store the secret information. The following picture Fig. 11 shows that the thirteenth and fourteenth bytes are not utilized\(^{[2]}\).

**Session layer:** The session layer allows two machines to establish sessions over the network. These sessions allow ordinary data transfer plus enhanced services for some applications e.g., creation of session-key in private-key cryptographic application\(^{[3]}\). This function is achieved via software that can "mount" remote discs on a local machine. Richard Popa, who has conducted research in this area at the University of Timisoara, has described the following scheme that can be used to establish covert communication channel\(^{[4]}\):
"Suppose we have two files on the disk of Alice, Bob can read one of them. If he reads the first file then Alice records a zero and if he reads the second file she records a one."

The fact that Joe can see this traffic should not arouse his suspicion, since it is irrelevant to him that Bob reads one file rather than another. Richard Popa has also mentioned that a demonstration of this scheme was successfully conducted, obtaining a 300 bits per second rate.

Cryptography layer: The purpose of steganography is to avoid drawing suspicion to the transmission of a hidden message between traditionally Alice and Bob. Steganography, as cryptographic systems, usually involve both an algorithm and a secret-key. Reason for having a secret-key is that keeping the algorithm secret is very difficult. In fact a good cryptographic system would follow the Kerchoff’s principle: the security of the algorithm resides in the secret-key, without the knowledge of the secret-key, any attack has very little chance to succeed. So any “good” steganographic system or tool would allow for everyone to know the algorithm used. If a text message is being sent through an insecure channel and using a good steganography technique to hide the message, Joe (system administrator) and for that matter Eve (the eavesdropper), would not get suspicious that there is a hidden message-passing or going on between two parties. In addition, an extra layer of security could be added if a text message is being sent. Sender could encrypt a message before applying steganographic tool on this message, thus adding extra layer of security. So even if Joe detects that there is message-hiding going on, he would first need to extract the original message. However, without a secret-key it should be impossible to accomplish that if a “good” cryptographic tool is used. Even then, if Joe does extract the original message, he would then somehow need to decrypt the message to get the plaintext. Although, detecting the message hiding would defeat the purpose of steganography, this still adds an extra layer of protection on the cryptosystem and the OSI protocol.

Covert channels can also be established using the control data, timing properties of transmission or of the user data. In this approach it is very difficult or almost impossible to prove the existence of covert channels, because the information is stripped off at the receiver. But if the information is hidden using user data, it remains on the hard disk until it is specifically deleted. Thus, network systems can be utilized in cryptography to establish hidden channels of communications.

Steganography in audio: Because of the range of the human auditory system (HAS), data hiding in audio signals is especially challenging. The HAS perceives over a range of power greater than one billion to one and range of frequencies greater than one thousand to one. Also, the auditory system is very sensitive to additive random noise. Any disturbances in a sound file can be detected as low as one part in ten million (80 dB below ambient level). However, while the HAS has a large dynamic range, it has a fairly small differential range-large sounds tend to drown quiet sounds. When performing data hiding on audio, one must exploit the weaknesses of the HAS, while at the same time being aware of the extreme sensitivity of the human auditory system.

Audio environments: When working with transmitted audio signals, one should bear in mind two main considerations. First, the means of audio storage, or digital representation of the audio and second, the transmission medium the signal might take.

Digital representation: Digital audio files generally have two primary characteristics:

Sample quantization method: The most popular format for representing samples of high-quality digital audio is a 16-bit linear quantization, such as that used by WAV (Windows Audio-Visual) and AIFF (Audio Interchange File Format). Some signal distortion is introduced by this format.

Temporal sampling rate: The most popular temporal sampling rates for audio include 8 kHz (kilohertz, 9.6, 10, 12, 16, 22.05 and 44.1 kHz). Sampling rate puts an upper bound on the usable portion of the frequency range. Generally, usable data space increases at least linearly with increased sampling rate.

Another digital representation that should be considered is the ISO MPEG-Audio format, a perceptual encoding standard. This format drastically changes the statistics of the signal by encoding only the parts the listener perceives, thus maintaining the sound, but changing the signal.

Transmission medium: The transmission medium, or transmission environment, of an audio signal refers to the environments the signal might go through on its way from encoder to decoder. Bender in his article identifies four possible transmission environment.

Digital end-to-end environment: If a sound file is copied directly from machine to machine, but never modified,
then it will go through this environment. As a result, the sampling will be exactly the same between the encoder and decoder. Very little constraints are put on data hiding in this environment.

**Increased/decreased resampling environment:** In this environment, signals is resampled to a higher or lower sampling rate, but remains digital throughout. Although the absolute magnitude and phase of most of the signal are preserved, the temporal characteristics of the signal are changed.

**Analog transmission and resampling:** This occurs when a signal is converted to an analog state, played on a relatively clean analog line and resampled. Absolute signal magnitude, sample quantization and temporal sampling rate are not preserved. In general, phase will be preserved.

**“Over the air” environment:** This occurs when the signal is “played into the air” and “resampled with a microphone”. The signal will be subjected to possible unknown nonlinear modifications causing phase changes; amplitude changes, drifting of different frequency components, echoes, etc.

The signal representation and transmission environment both need to be considered when choosing a data-hiding method.

**The mechanics of audio data hiding:** We now need to consider some methods of audio data hiding. Already we have considered techniques on how data is stored in the least-significant bit of images. These techniques can easily be extended to study how binary data can be stored in the least-significant bit of audio files.

**Low-bit encoding:** Ideally the channel capacity is 1 kb per second per kHz, so for example, the channel capacity would be 44 kbps in a 44 kHz sampled sequence. Unfortunately, this introduces audible noise. Of course, the primary disadvantage of this method is its poor immunity to manipulation. Factors such as channel noise and resampling can easily destroy the hidden signal. A particularly robust implementation of such a method is described by Bassia and Pitas\[2\]. The result is a slight amplitude modification of each sample in a way that does not produce any perceptual difference. Their implementation offers high robustness to MPEG compression plus other forms of signal manipulation, such as filtering, resampling and requantization.

**Phase coding:** The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the data. The procedure for phase coding is as follows:

- The original sound sequence is broken into a series of N-short segments.
- A discrete Fourier transform (DFT) is applied to each segment, to break and create a matrix of the phase and magnitude.
- The phase difference between each adjacent segment is calculated.
- For segment \( S_n \), the first segment, an artificial absolute phase \( P_1 \) is created.
- For all other segments, new phase frames are created.
- The new phase and original magnitude are combined to get a new segment, \( S_n \).
- Finally, the new segments are concatenated to create the encoded output.

For the decoding process, the synchronization of the sequence is done before the decoding. The length of the segment, the DFT points and the data interval must be known at the receiver. The value of the underlying phase of the first segment is detected as 0 or 1, which represents the coded binary string.

**Spread spectrum:** Most communication channels try to concentrate audio data in as narrow a region of the frequency spectrum as possible in order to conserve bandwidth and power. When using a spread spectrum technique, however, the encoded data is spread across as much of the frequency spectrum as possible. One particular method discussed in\[19\], the Direct Sequence Spread Spectrum (DSSS) encoding, spreads the signal by multiplying it by a certain maximal length pseudorandom sequence, known as a chip. The sampling rate of the host signal is used as the chip rate for coding. The calculation of the start and end quanta for phase locking purposes is taken care of by the discrete sampled nature of the host signal. As a result, a higher chip rate and, therefore, a higher associated data rate are possible. However, unlike phase coding, DSSS does introduce additive random noise to the sound.

**Echo data hiding:** Echo data hiding embeds data into a host signal by introducing an echo\[20\]. The data are hidden by varying three parameters of the echo: initial amplitude, decay rate and offset, or delay characteristics. As the offset between the original and the echo
decreases, the two signals blend. At a certain point, the human ear cannot distinguish between the two signals and the echo is merely heard as added resonance. This point depends on factors such as the quality of the original recording, the type of sound and the listener.

By using two different delay times, both below the human ear's perceptual level, we can encode a binary one or zero. The decay rate and initial amplitude can also be adjusted below the audible threshold of the ear, to ensure that the information is not perceivable. To encode more than one bit, the original signal is divided into smaller portions, each of which can be echoed to encode the desired bit. The final encoded signal is then just the recombination of all independently encoded signal portions.

As a binary one is represented by a certain delay \( y \) and a binary zero is represented by a certain delay \( x \), detection of the embedded signal then just involves the detection of spacing between the echoes. A process for doing this is described in the work by Gruhl et al.\(^{[20]}\). Echo hiding was also found to work exceptionally well on sound files where there is no additional degradation, such as from line noise or lousy encoding and where there is no gaps of silence. Research work to eliminate these drawbacks is still underway.

**Steganographic in watermarking:** Another branch of steganography in use today is in the area of document watermarking, which involves the protection against removal technique\(^{[17]}\). Protection against removal is mostly used for document marking. This process is usually done in order to embed information about the author(s) or a serial number into a file, in other words to embed a copyright information. The goal in this branch of steganography is to protect some kind of a document or a file against removal of copyright information. Even if a reader does discover the watermarking it should be impossible to remove this watermark without distorting or destroying the original object. There are two techniques for document marking: watermarking and fingerprinting. The difference between the two is only in what kind of information is being embedded. Watermarking is the process of embedding marks into digital documents (sounds, images, binaries, etc.) the same way money is marked. Each electronic document has identical mark. Fingerprinting is the process of embedding serial numbers into every copy of an object. This serial number can be used to detect the break of licensing agreement. In both of these cases it could be possible to detect the markings, however, it should be practically impossible to remove them.

One problem identified by Brassil and others is the illegal distribution of documents through modern electronic means, such as via electronic mails. Means such as this allow infringers to make identical copies of documents without paying royalties or revenues to the original author. To counteract this possible wide-scale piracy, Brassil et al.\(^{[37]}\) in this article discusses a method of marking printable documents with a unique codeword that is indiscernible to readers, but can be used to identify the intended recipient of a document just by examination of a recovered document.

The marking techniques Brassil et al.\(^{[37]}\) describes are to be applied to either an image representation of a document or to a document format file, such as PostScript or TEX files. The idea is that a codeword (such as a binary number, for example) is embedded in the document by altering particular textual features. By applying each bit of the codeword to a particular document feature, one can encode the codeword. It is the type of feature that identifies a particular encoding method.

The technique they propose is intended to be used in conjunction with standard security measures. For example, documents should still be encrypted prior to transmission across a network. Primarily, their techniques are intended for use after a document has been decrypted, i.e., once it is readable to all. An added advantage of their system is that it is not prone to distortion by methods such as photocopying and can thus be used to trace paper copies back to their source. Brassil identifies three features, which are described in the following subsections:

**Line-shift coding:** In this method, text lines are vertically shifted to encode the document uniquely. By moving every second line of document either 1/300 of an inch up or down, Brassil et al.\(^{[37]}\) found that line-shift coding worked particularly well and documents could still be completely decoded, even after the tenth photocopy. However, this method is probably the most visible text coding technique to the reader.

**Word-shift coding:** In word-shift coding, codewords are coded into a document by shifting the horizontal locations of words within text lines, while maintaining a natural spacing appearance. This may be achieved, for example, for each text-line, the largest and smallest spaces between words are found. To code a line, the largest spacing is reduced by a certain amount and the smallest is extended by the same amount. This maintains the line length and produces little visible change to the text.
Feature coding: This technique is mainly applied either to the bitmap image of a document, or to a format file. In feature coding, certain text features are altered, or not altered, depending on the codeword. For example, one could encode bits into text by extending or shortening the upward, vertical endlines of letters such as b, d, h, etc. Due to the frequently high number of features in documents that can be altered, feature coding supports a high amount of data encoding. Also, feature encoding is largely indiscernible to the reader.

An alternative application of text steganography suggested by Bender et al. is annotation that is, checking that a document has not been tampered with. Moreover, hidden data in text could even be used by mail servers to check whether documents should be posted or not. When embedding data, Bender et al., reminds us that it is important to remember the following restrictions and features:

- The cover data should not be significantly degraded by the embedded data and the embedded data should be as imperceptible as possible. (This does not mean the embedded data needs to be invisible; it is possible for the data to be hidden while it remains in plain sight.)
- The embedded data should be directly encoded into the media, rather than into a header or wrapper, to maintain data consistency across formats.
- The embedded data should be as immune as possible to modifications from intelligent attacks or anticipated manipulations such as filtering and resampling.
- Some distortion or degradation of the embedded data can be expected when the cover data is modified. To minimise this, error correcting codes should be used.
- The embedded data should be self-clocking or arbitrarily re-entrant. This ensures that the embedded data can still be extracted when only a portion of the cover data is available. For example, if only a part of image is available, the embedded data should still be recoverable.

SUMMARY AND CONCLUSION

This study presented an introductory look at steganography. Historical detail is discussed. Several methods for hiding data in text, images and audio are described, with appropriate introductions to the environments of each medium, as well as the strengths and weaknesses of each method. Most data-hiding systems take advantage of human perceptual weaknesses, but have weaknesses of their own. We conclude that for now, it seems that no system of data-hiding is totally immune to attack.

We have also noted that steganography has its place in data security transmitted electronically over insecure network. It in no way can replace cryptography, but is intended to supplement it. Its application in watermarking and fingerprinting, for use in detection of unauthorized and illegally copying of material is continually being realized and developed.

Also, in places where standard cryptography and encryption is outlawed, steganography can be used for covert data transmission. Steganography, formerly just an interest of the military, is now gaining popularity among the masses. Soon, any computer user will be able to put his own watermark on his artistic creations.

This study also discusses the use of steganography to create a file system, which protects its contents by concealing their very existence. It explains the basic requirements for a steganographic file system and the expected properties of such a system. We have also described StegFS and highlighted some of the ways it addresses and the requirements for its successful implementation. In the future the protection of electronic data will continue to be a priority. A steganographic file system can be a useful addition to the arsenal of tools already in use around the world for secure data communication.

The advantages of a steganographic file system are considerable, considering that an attacker who does not possess the name of a file and the password for accessing it cannot determine whether the file is even present. Even if a skilled attacker has unlimited access to a computer and understands it completely, a steganographic file system allows the data owner to plausibly deny that any files are hidden at all.

It is our impression that the scientific community outside national security agencies and the military world has not yet examined the field of steganography in detail. Many of the above mentioned problems in the design of high quality steganographic systems have not been addressed in the literature and only very few attempts of practical solutions have been published and analyzed so far.

Considering that Steganography can make use of unlimited legal means to use any web site without attracting the attention of anybody, including the owners of the target web sites themselves, it remains to be seen how governments and security agents tackle terrorist abuse thereof.

Finally, because the ability to surreptitiously intercept and decrypt encrypted information has historically been a significant military asset, the U.S.
Government restricts export of cryptographic software, including most software that permits use of symmetric encryption keys longer than 40 bits. (Obviously, the responsibility for obeying the laws in the jurisdiction in which you reside is entirely your own).

REFERENCES