Investigation and Experiment in Digital Steganography

Ioannis L. Kolaxis
Birmingham, June 2002
i.kolaxis@ieee.org

Aristotle University of Thessaloniki
School of Engineering
Faculty of Electrical and Computer Engineering
Informatics Laboratory, General Department

Aston University
School of Engineering and Applied Science
Computer Science Group
**Acknowledgements**

I would like to express my sincere gratitude to the following people, for their support and assistance in completing this project:

Professor George Pangalos (Aristotle University of Thessaloniki, Greece) for offering me the chance to live and study abroad, helping me gain invaluable experience.

Lecturer Aspasia Synefaki (Aristotle University of Thessaloniki, Greece) for being patient, always helping me overcome any difficulties or problems I came along.

Lecturer Bernard Doherty (Aston University, UK) for his guidance and advice throughout the duration of the project.
Table of Contents

ABSTRACT .......................................................................................................................... 4
SUMMARY ........................................................................................................................... 4

1. INTRODUCTION ............................................................................................................. 6
   1.1. HISTORY OF STEGANOGRAPHY ............................................................................. 6
       1.1.1. Technical Steganography ..................................................................................... 6
       1.1.2. Linguistic Steganography ..................................................................................... 7
       1.1.3. Preventing Steganography .................................................................................... 8
   1.2. STEGANOGRAPHY AS AN ALTERNATIVE TO CRYPTOGRAPHY ....................... 9
   1.3. PROJECT OVERVIEW ............................................................................................... 10

2. STEGANOGRAPHIC TECHNIQUES .............................................................................. 12
   2.1. EMBEDDING SECRET MESSAGES .......................................................................... 12
       2.1.1. Substitution Techniques ....................................................................................... 12
           2.1.1.1. Text .................................................................................................................. 12
           2.1.1.2. Images .............................................................................................................. 13
           2.1.1.3. Audio ................................................................................................................ 13
           2.1.1.4. TCP/IP Protocol Suite ...................................................................................... 14
       2.1.2. Transform Domain Techniques ........................................................................... 14
   2.2. DETECTING SECRET MESSAGES ........................................................................... 15

3. LITERATURE REVIEW ................................................................................................. 16

4. IMPLEMENTATION OF A LSB SUBSTITUTION STEGANOGRAPHIC TOOL ............ 19
   4.1. PROGRAM OVERVIEW ............................................................................................. 19
   4.2. COMPRESSING/DECOMPRESSING .......................................................................... 20
   4.3. ENCRYPTING/DECRYPTING ..................................................................................... 20
   4.4. EMBEDDING/EXTRACTING ....................................................................................... 22
       4.4.1. The *.bmp file format .......................................................................................... 23
           4.4.1.1. 8-bit bitmap files .............................................................................................. 23
           4.4.1.2. 24-bit (true colour) bitmap files ...................................................................... 24
       4.4.2. Embedding ........................................................................................................... 24
       4.4.3. Extracting ............................................................................................................. 26
       4.4.4. Pseudocode .......................................................................................................... 27
       4.4.5. Flow Charts .......................................................................................................... 29

5. EVALUATION OF THE STEGANOGRAPHIC TOOL IMPLEMENTED ..................... 33
   5.1. DETECTING HIDDEN INFORMATION ..................................................................... 33
       5.1.1. 24-bit images ........................................................................................................ 34
       5.1.2. 8-bit images .......................................................................................................... 42
       5.1.3. Conclusions .......................................................................................................... 46
   5.2. EXTRACTION HIDDEN INFORMATION .................................................................... 46
   5.3. DISABLING HIDDEN INFORMATION ....................................................................... 47
   5.4. FURTHER WORK ....................................................................................................... 48

CONCLUSIONS .................................................................................................................... 49
REFERENCES ....................................................................................................................... 50
BIBLIOGRAPHY ................................................................................................................... 53
APPENDIX A ....................................................................................................................... 55
APPENDIX B ....................................................................................................................... 57
Abstract

Steganography is the art of passing information in a manner that the very existence of the message is unknown. The goal of steganography is to avoid drawing suspicion to the transmission of a hidden message. If suspicion is raised, then this goal is defeated. Unfortunately, there has not been any significant research into steganography, although it is a science that dates back to ancient times. The primary purpose of this project is to increase the sum of knowledge in the field of steganography. First of all, the use of digital steganography as an alternative to cryptography is suggested, especially in countries where cryptography is outlawed. An investigation into digital steganography is conducted, reviewing possible ways of embedding secret messages into digital media, like images and audio. Moreover, the design and implementation of a steganographic tool which embeds information into digital images is analysed extensively. Next, we experiment on steganography, trying to determine the limitations and flexibility of the steganographic tool implemented. For this purpose, various forms of attacks are applied on the steganographic tool, including the detection, extraction and disabling of a hidden message in a digital image. The results of the experiments are discussed in depth, stating clearly the weaknesses of the implemented steganographic tool.

Summary

Steganography embeds a confidential message into another, more extensive message which serves as a carrier. The goal of steganography is to modify the carrier in an imperceptible way only, so that it reveals nothing: neither the embedding of a message, nor the embedded message itself.

Due to the restrictions imposed on cryptography in many countries all over the world, steganography is proposed as an alternative means of communicating privately. In fact, there are no laws in any country banning the use of steganography. On the other hand, steganography is not a rather researched discipline, although it is a science that dates back to antiquity. Consequently, we must first conduct an investigation into steganography and experiment with the techniques used to embed information into digital media; the goal of this research is to spot possible weaknesses and determine if steganography is secure enough to substitute cryptography. In case it is proved that it is possible for a third party to detect the existence of an embedded, secret message in an innocent-looking media or even worse to extract the secret message, then steganography is not a safe means of communicating privately.

In particular, possible ways of embedding secret messages into digital media, like text, images and audio are examined. Actually, there are two basic categories of steganographic techniques: Substitution and Transform Domain techniques. Although the majority of the currently available steganographic tools implement substitution techniques, there have not been any serious attempts to evaluate them. It is important that we are aware of the security level each technique can provide, as far as the detection, the extraction or the destruction of an embedded message is concerned.
In particular, the LSB substitution method\(^1\) is the most widely used steganographic technique; however, there has not been any significant research, in order to determine its limitations and flexibility. For that reason, we have implemented a steganographic tool that embeds secret messages into bitmap images, using the LSB substitution method. This particular steganographic tool is capable of embedding one, two or four bits of secret message data into each image byte. Moreover, the secret message is being compressed and encrypted, before being embedded into an image. Our purpose is to analyse in-depth the design and implementation of the steganographic tool. The embedding and extracting processes are discussed in detail, for two separate cases: 8-bit (256 colours) and 24-bit (true colour) bitmap images. Moreover, the pseudocode and flow charts of the embedding and extracting functions are provided, so that the reader can easily understand them.

However, the primary goal of this project is to test extensively the steganographic tool implemented, in order to discover all the inherent weaknesses of the LSB substitution technique. The evaluation of the steganographic tool includes embedding secret messages into various images, like digital photos, digital art and clip art images and applying attacks to the resulting images (the ones containing the hidden information). Particularly, the steganographic tool has been tested on the following form of attacks: detecting, extracting and disabling hidden information.

We have concluded that the detection of a hidden message depends primarily on the image used for hiding information; digital art and clip art images are a poor choice, whereas digital photos are ideal carriers. Especially when true colour digital photos are used and only one or two bits of secret message data are embedded into each image byte, the original image and the image that contains hidden information look identical to the human eye. However, if both images are available to an attacker, a comparison can take place, revealing some slight differences between them; those differences may raise suspicion, defeating the goal of steganography. To detect such differences the known-cover and visual attacks are applied to images that could contain hidden information.

In reality, it is possible for someone to extract a hidden message, which has been embedded into an image using our steganographic tool, especially if he is aware of the embedding and extracting algorithms that it implements. A solution to this problem would be to scatter the hidden data throughout an image, instead of embedding the hidden data sequentially in the image’s least significant bits. On the other hand, even if a secret message is extracted, it would be impossible for anyone to read it, since it would be encrypted.

To determine the tolerance of an embedded message to image manipulation, a series of image processing tests have been devised. Unfortunately, it was proved that the slightest image processing was enough to destroy the hidden data. Consequently, the LSB substitution method cannot provide the highest level of security that would be demanded by intelligence agencies or the army, since it is possible for an attacker to extract or destroy the hidden message. On the other hand, the steganographic tool implemented would be suitable for individuals that wish to communicate privately.

\(^1\) Further information on steganographic techniques is available at Chapter 2
1. Introduction

Steganography is the art and science of secret communication. The word steganography comes from the Greek *steganos* (covered or secret) and *-graphy* (writing) and thus means, literally, covered writing. The purpose of steganography is to hide the presence of communication, as opposed to cryptography, which aims to make communication unintelligible to those who do not possess the right keys. In particular, Kuhn (1995) argues that “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”.

1.1. History of Steganography

In this project *digital steganography* is being investigated, which is about embedding *secret messages*\(^1\) into digital media, like images, audio, video, etc. It must be stressed though that steganography is not a new science, but it originates from ancient Greece. There are quite a few examples of steganography throughout history, which can be divided into the following two categories: *Technical* and *Linguistic* steganography.

1.1.1. Technical Steganography

Herodotus tells how Histiaeus managed to send a secret message to his allies, initiating a revolution against the Persians. He shaved the head of his most trusted slave, tattooed the message on the slave’s head, waited until his hair grew back and then sent him along. When the slave reached the allies, he shaved his head, revealing the hidden message.

Furthermore, Herodotus tells how Demeratus, a Greek at the Persian court, warned Sparta that Xerxes, King of Persia was about to invade Greece. Demeratus concealed the message under writing tablets; those were usually two pieces of wood, having each face covered with wax. When someone wanted to sent a message, he wrote it on the wax; after the recipient had read the message, he could melt the wax and reuse the tablets. In fact, Demeratus removed the wax from the writing tablets, wrote his message on the wood underneath and then covered the message with wax. The writing tablets looked as if they were blank, fooling the customs men.

A large number of steganographic techniques have been invented by Aeneas the Tactician, including letters hidden in messengers’ soles or women’s earrings, text written on wood tablets and then whitewashed and notes carried by pigeons. He also proposed hiding text by changing the heights of letterstrokes or by making very small holes above or below letters in a *cover-text*. This latter technique was improved in the 17\(^{th}\) century, using invisible ink to print very small dots, instead of making holes.

\(^1\) A *secret message* is the information embedded into the digital media. We refer to the blank digital media as *cover-media*. Together the *cover-media* and the embedded message create the *stego-media*. 
With invisible ink, a seemingly innocent letter could contain a very different message written between the lines. Invisible inks were originally made of available organic substances (such as milk or urine) or sal armoniack (ammonium chloride NH₄Cl) dissolved in water and developed with heat. Progress in chemistry helped to create more sophisticated combinations of ink and developer by the First World War. Letters written in invisible ink needed special care; water or other liquids could smear the invisible ink and make it impossible to read. Unfortunately, the technology fell into disuse with the invention of universal developers, which could determine which parts of a piece of paper had been wetted, from the effects on the surfaces of the fibers. Moreover, it was not possible to transmit large amounts of data using invisible inks (Petitcolas et al. 1999; Clements Library 2000).

During the Franco-Prussian war of 1870-1871, while Paris was under siege, messages were sent out by pigeon post. The messages were written by hand, in big characters, on large sheets of card, which were pinned side by side and photographically reduced. The prints were on photographic paper and varied in size, but with one side not significantly exceeding 40 mm (Hayhurst 1970). By the First World War, better lenses and photographic film have been developed, allowing photographers to reduce the size of a photograph down to the size of a printed period. Those photographs, which had the size of a printed period, were called microdots and when developed, they could reproduce a standard-sized type-written page with perfect clarity. Microdots were used widely by German spies during the Second World War and they were usually stuck on top of printed periods or commas in innocuous cover material, such as magazines. Aside from being extremely difficult to detect, microdots permitted the transmission of large amounts of printed data, including technical drawings (Kahn 1967).

A well-publicized example of steganography occurred during the height of the Vietnam War, when U.S.A. Commander Jeremiah Denton, a naval aviator who had been shot down and captured by North Vietnamese forces, was paraded in front of the news media as part of a well-staged propaganda event. Denton knew he would be unable to say anything critical about his captors, so as he spoke to the media, he blinked his eyes in Morse code, spelling out T-O-R-T-U-R-E (Denton 2002).

Nowadays, steganography is applied in the field of currency security; special inks are used to write a hidden message on bank notes or other secure documents. These inks provide a unique response to some particular excitation such as a laser light at a particular frequency (Murphy et al 1998).

1.1.2. Linguistic Steganography

A widely used method of linguistic steganography is the acrostic. A famous example of acrostic comes from the Hypnerotomachia Poliphili, published in 1499. This book, written anonymously, reveals the guilty love between a monk and a woman. The first letter of the thirty eight chapters spelled out “Poliam frater Franciscus Columna peramavit”, which is translated as “Brother Francesco Colonna passionately loves Polia”. Colonna was a monk, still alive when the book was published (Katzenbeisser & Petitcolas 2000).
In a security protocol developed in ancient China, the sender and the receiver had copies of a paper mask with a number of holes cut at random locations. The sender would place his mask over a sheet of paper, write the secret message into the holes, remove the mask and then compose a cover message incorporating the code ideograms. The receiver could read the secret message at once by placing his mask over the resulting letter. In the early sixteenth century, Cardano, an Italian mathematician reinvented this method which is now known as the Cardano grille. The Cardano grille has been used extensively by many countries, for their diplomatic correspondence, during the sixteenth century (Davern & Scott 1996).

Another form of linguistic steganography is to mark an object by the presence of errors or stylistic features at predetermined points in the cover-text. An early example was a technique used by Bacon (1561-1626) in his biliterarie alphabet. In this method, each letter is encoded in a 5-bit binary code and embedded in the cover-text by printing the letters in either normal or italic fonts. The variability of the sixteenth-century typography acted as camouflage (Petitcolas et al 1999).

Document text may contain a hidden message through the use of null ciphers, which camouflage the real message in an innocent-sounding letter. In the null cipher, only certain words or letters from the cover-text are significant, constituting the secret message. For example, the following null cipher message was actually sent by a German spy in World War II (Kahn 1967):

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

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

Pershing sails from NY June 1

Unfortunately, letters containing null ciphers usually sound very weird, because of the difficulty in getting the correct order to the words of the secret message.

1.1.3. Preventing Steganography

The threat of microdots, secret tongues, and other forms of steganography during World War II caused the Office of Censorship to introduce a series of bizarre restrictions. Banned in advance were the international mailing of postal chess games, crossword puzzles, newspaper clippings, knitting instructions, children's drawings, and report cards, all of which, it was thought, could serve as a communications medium for spies. It was also illegal to send cables ordering that specific types of flowers be delivered on a specific date, and eventually all international flower orders were banned by the U.S. and British governments. Government censors even went so far as to change or rearrange stamps on envelopes and even to rewrite letters, using different words (Gloeckler 1996).
1.2. Steganography as an Alternative to Cryptography

The use of digital steganography may enhance individual privacy. In countries where cryptography is totally banned or where restrictions apply to the key length, digital steganography can be an efficient way of communicating privately. Moreover, in some countries an order to decrypt encrypted data may be given by the authorities; failure to comply with the order is usually punishable with fines or imprisonment. In particular, Koops (2002) has reported that the following countries have a law demanding decryption:

**Australia:** There is a law that requires release of encryption keys or decryption of encrypted data, upon a magistrate's order. Failure to comply with the order is punishable with up to 6 months imprisonment.

**Belgium:** A refusal of someone to decrypt data is punishable with 6 to 12 months of imprisonment and a fine.

**France:** The domestic use of cryptography was liberalized in January 1999. There is a law that entails a power to require all qualified persons to decrypt or to hand over decryption keys if encrypted data are encountered during an investigation. Someone who fails to comply with a decryption order is punishable with a maximum of 3 years imprisonment and 45,000 Euro, or with 5 years and 75,000 Euro if decryption could have prevented or mitigated the effects of a crime.
India: If someone refuses to decrypt data, he can be punished with imprisonment of up to 7 years.

Ireland: Persons or public bodies who fail or refuse to comply with an order to decrypt data are guilty of a summary offence\(^1\).

Malaysia: A refusal of someone to decrypt data is punishable with 2 to 4 years of imprisonment and a fine.

The Netherlands: Failure to comply with a request to decrypt data is punishable with up to 6 months imprisonment if it was not intentional, and with up to 2 years imprisonment if the not-complying was intentional.

Singapore: Failing to comply with a decryption request is punishable with at most S$10,000 or 3 years of imprisonment.

United Kingdom: The order to decrypt data can be given if decryption is necessary in the interest of national security, crime prevention or detection, or the UK's economic well-being, or if it is necessary for the effective exercise or proper performance of a statutory power or duty, and if requiring decryption is proportionate and the only reasonably practicable means. A person who knowingly fails to comply with the order is punishable with up to 2 years of imprisonment.

Obviously, the citizens of the above countries are in dire need of an alternative to cryptography, so that they communicate without being subjected to their employer’s or government’s monitoring systems. In this particular case, digital steganography is a good solution, as long as the hidden communication is not detected. Furthermore, there are not any countries having imposed restrictions on steganography. Besides that, Franz et al. (1996) have argued that it is impossible to prove steganography to court.

1.3. Project Overview

Due to the restrictions imposed on cryptography in many countries all over the world, we wanted to conduct an investigation into digital steganography and examine if it can really provide an alternative means of communicating privately.

In reality, there has not been any significant research into steganography, although it is a science which dates back to ancient times. In particular, the biggest online bookstore\(^2\) had only 3 books available on steganography. Moreover, we found out that those books concentrate mostly on watermarks, which is a commercial application of steganography for the copyright protection of digital media, like images and audio.

---

\(^1\) For more information please refer to Koops (2002)

\(^2\) [http://www.amazon.com](http://www.amazon.com)
On the other hand, when looking for books on cryptography (at the same bookstore as before) we came up with 505 results. Similarly, we looked for articles from both disciplines (that is steganography and cryptography) at the IEEE\textsuperscript{1} Xplore\textsuperscript{2} digital library; there were only 42 documents on steganography, from a total number of 771,118 documents available in the library. On the contrary, there were 2,365 documents on cryptography. Furthermore, the majority of the available documents on steganography concentrate mainly on digital watermarks. It must be stressed that the requirements for a steganographic system are totally different from the requirements for a watermarking system.

According to Johnson & Jajodia (1998a), the difference between steganography and watermarks is primarily one of intent. Digital steganography conceals information, whereas watermarks extend information and become an attribute of the cover-media. Digital watermarks may include such information as copyright, ownership, or license. In steganography, the object of communication is the hidden message. In digital watermarks, the object of communication is the cover. A brief review of the most important differences between digital steganography and watermarks is available at Katzenbeisser & Petitcolas (2000, pp. 2-3).

It is clear that the field of steganography has not yet been examined in detail by the scientific community. Moreover, the main driving force for research in the field of steganography is concern over protecting copyright. Consequently, research is restricted to digital watermarks, because of the need of music, film and software publishing industries to prevent and control copyright violations, by embedding copyright messages and serial numbers into digital media. Unfortunately, there has not been an overall approach to the field of steganography; most scientific articles concentrate on particular aspects of steganography and not steganography as a whole. Additionally, there have not been published any attempts of implementing and analysing a steganographic system.

The purpose of this project is to conduct an extensive investigation into steganography; thus we wish to study in depth the currently available articles about steganography and then try to present all of those bits of scattered information as a whole, something which is being done in the next chapters. Moreover, we aim to experiment on steganography; thus we will develop a steganographic tool, which will be implementing the LSB substitution\textsuperscript{3} steganographic method. We have chosen to implement this particular method, because it is used by the majority of the steganographic tools freely available on the internet. However, there has not been an extensive analysis of the LSB substitution method by anyone so far, in order to determine its limitations and flexibility. Thus, our target is to apply attacks on the tool we will implement, exposing its weaknesses and suggesting possible improvements. Most of all, the primary goal of the project is to increase the sum of knowledge in the field of steganography.

\textsuperscript{1} IEEE: Institute of Electrical and Electronics Engineers
Further information on IEEE is available at the official website \texttt{http://www.ieee.org}

\textsuperscript{2} IEEE Xplore provides full-text access to the IEEE transactions, journals and conference proceedings published since 1988 and all current IEEE standards. IEEE members can access the digital library at the following URL: \texttt{http://ieeexplore.ieee.org}

\textsuperscript{3} Further information on the LSB substitution steganographic method is available in the next chapter
2. Steganographic Techniques

2.1. Embedding Secret Messages

The general principle underlying most steganographic techniques is to place the secret message in the noise component of a signal. If it is possible to embed the secret message in such a way that it is indistinguishable from true random noise, then an attacker has no chance in detecting the secret communication.

Here, we will classify steganographic techniques according to the cover modifications applied in the embedding process. Actually, there are two basic categories of steganographic techniques: Substitution and Transform Domain techniques.

2.1.1. Substitution Techniques

Substitution techniques embed a secret message by substituting insignificant parts of the cover-media with secret message bits; the receiver can extract the secret message if he has knowledge of the positions where the secret message bits have been embedded. Since only small changes are made in the embedding process, the sender assumes that they will not be noticed by a third person.

2.1.1.1. Text

Unlike images and audio, written text does not contain a noise component, which could be substituted with a secret message. The only form of substitution that can take place is by encoding the secret message directly in the text, exploiting the natural redundancy of languages. Bender et al. (1996) suggest assigning a primary or secondary value to two synonyms; for instance, the word “big” could be considered primary and “large” secondary. When decoding a stego-text, primary words will be read as “1”, secondary words as “0”. Some other synonyms that could also be used are “small-little”, “chilly-cool”, “smart-clever”, etc. Where there are more than two synonyms more than one bit can be encoded per substitution. For example, the choice between “big”, “large”, “huge” and “vast” represents two bits of hidden data. However, there is a problem with the choice of some synonyms, like the synonym pair “chilly” and “cool”; calling someone “cool” is totally different than calling him “chilly”.

Alternatively, many other ways have been proposed to embed a secret message in written text. Infrequent typing or spelling errors could be introduced, commas omitted and spaces added. Most of them are not serious options, as they degrade the text heavily. Additionally, the embedding task requires the interaction of the user, it therefore cannot be automated.
### 2.1.1.2. Images

The most commonly used technique for embedding hidden information in digital images is the *Least Significant Bit Substitution*, where the LSB of each image byte is replaced with a secret message bit. We usually refer to the steganographic tools implementing the particular technique as *bitplane*, *LSB insertion* or *LSB substitution* tools.

The embedding process consists of choosing a subset of cover-image bytes and performs the substitution operation on them, which replaces the LSB of a cover-image byte with a secret message bit. The substitution operation could also be changing more than one bit of the cover, for instance by storing two secret message bits in the two least significant bits of one cover-image byte. In the extraction process, the LSBs of the selected cover-elements are extracted and lined up to reconstruct the secret message.

The cover-image formats typically used in substitution techniques are *lossless*; thus the image data can be directly manipulated and recovered. Some of the most common cover-image formats are BMP\(^1\) and GIF\(^2\) format. Some steganographic tools that employ the LSB substitution technique to embed information into images are the ones implemented by Arachelian (1996), Hansmann (2001), Hetzl (2000), Maroney (1997) and Wolf (1993).

### 2.1.1.3. Audio

Sound waves, like speech or music, can be transformed into analogous voltages by a microphone. These voltages can be digitised by an Analog/Digital converter. The Analog/Digital converter measures the current amplitude (meaning the value of the voltage) in short periods of time and transforms it into a binary number. This sampling takes place several times per second and thus nearly the original signal can be reconstructed from the established data.

When a sound wave is being converted from analog to digital, quantization takes place, introducing noise. Obviously, this noise can be substituted with hidden data. In particular, the *LSB substitution* technique is employed in order to hide information into digital audio; the LSB of any byte of a file with digital audio (cover-audio) can be set to the value of a bit stream of a secret message file. In fact, the embedding and extracting processes are similar to the ones used in digital images.

The cover-audio file formats that are commonly used are *lossless*, like WAV and AU files. Some steganographic tools that employ the LSB substitution technique to embed information into lossless audio files are the ones implemented by Hetzl (2000), Repp (1996) and Simpson (1999).

---

1 *BMP* is a Microsoft Windows and OS/2 bitmap file. Further information on the particular format is available on Chapter 4, paragraph 4.4.1 and Appendix A.

2 *GIF* stands for the Graphics Interchange Format
Furthermore, it is also possible to embed information into *lossy* audio file formats, like MP3\(^1\) files, applying the LSB substitution technique. Currently there is only one steganographic tool that hides information in MP3 files, by embedding the information into the MP3 bit stream; the tool in question is *MP3-Stego*, which has been implemented by Petitcolas (2002).

2.1.1.4. TCP/IP Protocol Suite

Protocols in the OSI\(^2\) network model have certain characteristics that can be exploited in order to hide information. TCP/IP\(^3\) packets, which are used to transport information across the Internet, have unused space in the packet headers. The TCP packet header has six unused (reserved) bits; similarly the IP packet header has two reserved bits. Obviously, it is possible to modify the TCP/IP packet header before sending it over the internet, substituting the header’s reserved bits with secret message bits. On the other hand, it is fairly easy to detect or destroy a message embedded into the header of a TCP/IP packet; internet routers can be programmed to rewrite TCP/IP packets before transmitting them, destroying a potential hidden message. For further information on transmitting covert data in normal Internet traffic, the reader may refer to Rowland (1996).

2.1.2. Transform Domain Techniques

Transform domain techniques are principally used for watermarks and they embed secret message data in significant areas of the cover-media. Transform domain techniques are generally more robust to cover-media processing than substitution techniques. Moreover, the information hidden in cover-media using transform domain techniques still remains imperceptible to the human eye. A lot of transform domain variations exist. The most popular transform domain technique is the *Discrete Cosine Transform (DCT)*\(^4\). The use of the DCT technique to hide information in digital images is analysed extensively by Koch & Zhao (1995) and Cox *et al.* (1996).

Latham (1999) has developed a steganographic tool that embeds information into JPEG images by applying a transform domain technique. Actually, information is hidden in the way DCT coefficients in the JPEG compression system are rounded.

---

\(^1\) **MP3** is actually the **MPEG Audio Layer III** format.

\(^2\) **OSI** stands for **Open Systems Interconnection**; it is a standard description of 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.

\(^3\) **TCP/IP**: **Transmission Control Protocol/Internet Protocol**; it is the basic communication language or protocol of the internet.

\(^4\) The two-dimensional DCT is the “heart” of the most popular lossy digital image compression system used today: the JPEG system. For further information on DCT and the JPEG compression algorithm, please refer to Wallace (1991).
### 2.2. Detecting Secret Messages

The science of detecting hidden information in digital media, like images and audio is called *Steganalysis*. The techniques employed in *steganalysis* are quite similar to the ones used in *cryptanalysis*\(^1\); just as a *cryptanalyst* applies cryptanalysis in an attempt to decipher encrypted messages, the steganalyst in one who applies steganalysis in an attempt to detect the existence of hidden information. In cryptanalysis, parts of the plaintext and parts of the ciphertext are analysed. In steganalysis, comparisons are made between the cover-media, the stego-media, and possible parts of the embedded message.

In order to define attack techniques used for steganalysis, analogous techniques are considered in cryptanalysis. According to Johnson & Jajodia (1998b), the attacks available to the cryptanalyst are *ciphertext-only*, *known-plaintext*, *chosen-plaintext* and *chosen-ciphertext*. In ciphertext-only attacks, the cryptanalyst knows the ciphertext to be decoded. If the cryptanalyst has the encoded message and part of the decoded message, he may use them together for a known-plaintext attack. When the cryptanalyst has some ciphertext which corresponds to some plaintext chosen by himself, then he applies the chosen-plaintext attack. In case both the encryption algorithm and the ciphertext are available, then the cryptanalyst encrypts plaintext looking for matches in the ciphertext; this chosen-ciphertext attack is applied to find the secret key. Similar attacks are available to a steganalyst:

**Stego-only**: only the stego-media is available for analysis.

**Known-cover**: the cover-media and stego-media are both available.

**Known-message**: in this case, the embedded message is supposed to be known to the steganalyst. The stego-media can be analysed for patterns that correspond to the embedded message; this analysis may prove useful for future attacks against the same stego-system.

**Chosen-stego**: the algorithm used to embed information in a cover-media and the resulting stego-media are known.

**Chosen-message**: the steganalyst creates a stego-media using a particular secret message and steganography tool. The purpose of this attack is to find out obvious and repetitive patterns, which will point to the identification or signature of the steganography tool in question.

**Known-stego**: the steganography tool and stego-media are known.

---

\(^1\) *Cryptanalysis* refers to the study of ciphers, ciphertext or cryptosystems with a view to finding weaknesses in them, that will permit retrieval of the plaintext from the ciphertext, without necessarily knowing the key or the algorithm.
3. Literature Review

It has been relatively difficult to find sufficient articles on steganography, since it is not a much researched discipline. Particularly, as mentioned in the first chapter, there have been published only three books on steganography; moreover, those books focus mainly on watermarking techniques and tools, which have completely different requirements than steganographic tools. Those three books are the ones written by Katzenbeisser & Petitcolas (2000), Johnson et al. (2001) and Wayner (2002). Unfortunately, only the first of the books in question was available at the library of Aston University. Moreover, the particular books were neither available at the University of Birmingham library, nor at the Birmingham Central Library. Consequently, we had to look for articles at the following digital libraries: BIDS, IEEE Xplore, INSPEC, and Science Direct. From all the digital libraries that we used, IEEE Xplore was the best one, having available a satisfactory number of articles on steganography. Furthermore, the Google search engine proved to be a valuable tool for our research. In this chapter, some of the most important articles on steganography will be reviewed.

The best article for introducing someone to the science of steganography is the one written by Johnson & Jajodia (1998a): it mentions everything from the history of steganography, to steganographic and watermarking techniques, including a brief review of currently available steganographic tools. However, it is too general, in the sense that it does not provide an in-depth analysis of steganography. Similarly, the articles written by Davern & Scott (1996), Petitcolas et al. (1999) and Artz (2001) are introductions to steganography, skipping any technical details. In fact, Davern & Scott (1996) examine the history of steganography, reviewing in brief some steganographic tools. However, they do not determine the ability of the particular steganographic tools to successfully hide information, meaning that they do not attempt to spot any vulnerabilities the tools may have. Petitcolas et al. (1999) review information hiding in general, including steganographic and watermarking techniques; they cover a wide area of the information hiding science, including the evolution of steganography and watermarks throughout history. Additionally, Artz (2001) examines digital steganography, specifying some of its possible applications in today’s world.

In the articles mentioned above, authors review steganography in general; they do not attempt to determine the limitations and flexibility of digital steganography. On the contrary, Anderson & Petitcolas (1998) try to define the theoretical limits of steganography by contrasting it with the related disciplines of cryptography and traffic security. However, they do not experiment in digital steganography; it would be wiser to apply theory in practice, just to verify the theoretical limits of steganography.

1 Birmingham Central Library, Chamberlain Square, Birmingham B3 3HQ, Tel: 0121-303 4306
2 http://www.bids.ac.uk
3 http://ieeexplore.ieee.org
4 http://edina.ac.uk/inspec/login.shtml
5 http://www.sciencedirect.com
6 http://www.google.com
On the other hand, Franz et al. (1996) introduce the reader to the art of steganography, taking him a step further; they provide an extensive analysis of their implementation. In particular, Franz et al. (1996) suggest the use of digital steganography as an alternative to cryptography, claiming that it is impossible to prove the use of steganography to court. Furthermore, they investigate the transmission of hidden information through an ISDN telephone line, by embedding data into digitised speech. There is a detailed description of the embedding process, providing plenty of technical facts and figures. Most of all, Franz et al. (1996) conduct a series of extensive tests, so as to evaluate the implemented steganographic system.

In their very interesting article, Lee & Chen (2000) propose a high capacity image steganographic model, which is based on variable-sized LSB substitution. According to the authors, the embedding capacity is over 50% of the cover-image size and the resulting stego-images look identical to the cover-images with a naked eye. Lee & Chen (2000) explain the embedding and extracting procedures going into details. Moreover, there is an epigrammatic evaluation of the implemented steganographic system, which proves to be efficient.

Bender et al. (1996) investigate various ways of embedding hidden information into digital media, like text, images and audio; an excellent, detailed overview of information hiding techniques is provided. However, Bender et al. (1996) focus mostly on watermarking techniques, which is not exactly the subject of our project; the main requirement of the information hiding techniques being discussed is that the embedded information must be immune to modification or removal by a third party. There are many more articles on watermarking techniques and systems; the most representative of them are the ones written by Koch & Zhao (1995) and Cox et al. (1996)

Besides Franz et al. (1996) and Lee & Chen (2000), no other scientists have made an attempt to test and evaluate the steganographic system they implemented. Even in the latter case (Lee & Chen 2000) the tests were not extensive and the evaluation was too brief and general. In this project, we intend to provide an in-depth analysis concerning the design and implementation of the steganographic tool, conducting an extensive series of tests. Our primary goal is to determine the limitations and flexibility of the steganographic tool implemented; we aim to spot its weaknesses and suggest possible changes in the steganographic technique implemented, that could make the stego-system more secure.

In order to evaluate the steganographic tool implemented, we needed to devise a series of tests. Actually, those tests had to be different forms of attacks on the steganographic tool. Unfortunately, there are not many articles on steganalysis; the only articles we were able to find are the ones written by Johnson & Jajodia (1998b), Westfeld & Pfitzmann (2000) and Fridrich et al. (2001a, 2001b).

---

1. Fixed-size LSB substitution methods embed the same number of message bits into each pixel of the cover image, whereas variable-sized LSB substitution methods embed a different number of message bits into each pixel.
2. For more information on steganalysis, please refer to Chapter 2, paragraph 2.2
Johnson & Jajodia (1998b) evaluate current steganographic software, by embedding secret messages into cover-images and then applying various forms of attacks on the resulting stego-images. The attacks discussed in this article include: detecting, extracting and disabling hidden information in a stego-image. To detect the existence of hidden information in an image, the chosen-stego\(^1\) and known-cover attacks are applied. Moreover, in order to assess the tolerance of an embedded message to stego-image processing, a series of image processing tests are applied. All steganographic tools examined in this article embed information into 8-bit\(^2\) bitmap cover-images. On the other hand, the steganographic tool we will develop for this project, will be embedding information into 24-bit (true colour) cover-images. Actually, embedding secret message data into 8-bit or 24-bit images is being done in a totally different manner\(^3\). This is because of the main difference between 8-bit and 24-bit bitmap image formats; in 8-bit images, the colour of each pixel is defined by an index to a colour table, whereas in 24-bit images it is defined by a list of RGB (Red, Green, Blue) values, each in the range 0-255. Consequently, the attacks proposed by Johnson & Jajodia (1998b) should also be applied to 24-bit stego-images, created by the steganographic tool that will be implemented; the main goal of this experiment will be to examine if 24-bit stego-images are vulnerable to the same forms of attacks as 8-bit stego-images.

Westfeld & Pfitzmann (2000) propose the use of visual attacks, in order to detect the existence of hidden information in an image. The main concept of a visual attack is to unveil the bits that may constitute a secret message, by removing the image bits that camouflage it. Then, the human eye can distinguish whether those bits constitute a secret message, or genuine image content. In order to evaluate the steganographic tool that will be implemented for this project, we will also make use of the visual attacks.

In their book, Katzenbeisser & Petitcolas (2000) analyse various information hiding techniques, separating steganographic from watermarking techniques. The book in question is actually a compilation of all the articles we reviewed, plus a few others. In fact, S. Katzenbeisser and F.A.P. Petitcolas are the editors of the particular book; besides them, its authors include R.J. Anderson, N.F. Johnson and M.G. Kuhn. In reality, this book does not add an extra piece of information in the field of steganography. However, it may prove really useful to someone who wishes to have all the previously reviewed articles in a more compact form.

---

\(^1\) For more information on the various forms of attacks, please refer to Chapter 2, paragraph 2.2
\(^2\) Further information on bitmap images is available at Chapter 4, paragraph 4.4.1
\(^3\) For a detailed description of the embedding process in 8-bit or 24-bit images, you may refer to Chapter 4, paragraph 4.4.2
4. Implementation of a LSB Substitution Steganographic Tool

4.1. Program Overview

The following figure shows the block diagram of the steganographic tool that has been implemented:

![Block Diagram of Steganographic Tool]

The secret message can be any kind of file, like plain text, image or audio files. That secret message is initially compressed, encrypted and then embedded into the cover-media. The cover-media can only be a *.bmp formatted image; thus the term cover-image will be used when referring to the cover-media. Similarly, the term stego-image will be used when referring to the resulting stego-media. When the embedded message is extracted from the stego-image, it will have to be decrypted and then decompressed, in order to get the plain secret message.

The embedding and extracting routines have been implemented from scratch, whereas some freely available libraries have been used for the compression/decompression and encryption/decryption of the secret message.

The ANSI C programming language has been preferred for the development of the steganographic tool, because it has powerful I/O facilities and bit manipulation functions. Moreover, the GNU Compiler Collection (GCC)\(^1\) has been used, which comes together with the Cygwin Tools. The Cygwin Tools are ports of the popular GNU development tools and utilities for Microsoft Windows. They function through the use of the Cygwin library, which provides the UNIX system calls and environment that these programs require.

---

\(^1\) Further information on Cygwin Tools and the GNU Compiler Collection is available in Appendix A.
4.2. Compressing/Decompressing

For the compression/decompression of the secret message we needed a freely available, widely recognized, lossless data-compression library, which would be written in ANSI C. Furthermore, the compression algorithm implemented by the library had to be capable of achieving high compression ratios. The compression libraries that have been under consideration are the following ones:

\[ \text{LZO}^1, \text{Libbzip2}^2 \text{ and } \text{Zlib}^3 \]

From the above libraries, LZO has the fastest compression and decompression algorithms. However, it favours speed over compression ratio. Similarly, Libbzip2 has fast compression and decompression algorithms, but it does not achieve high compression ratios. Moreover, it is not a widely recognized compression library. On the other hand, Zlib may be slower, requiring more memory during execution, but it achieves the best compression ratios.

From the above libraries, Zlib has been preferred, since one of the major requirements was that of the high compression ratios. Zlib’s compression method is a LZ77 variant called deflation. Actually, LZ77 compression works by finding sequences of data that are repeated. Theoretically, the deflate method is capable of achieving compression factors up to 1.032:1. However, this level of compression is extremely rare and only occurs with really trivial files (e.g. a megabyte of zeros). More typical Zlib compression ratios are on the order of 2:1 to 5:1.

4.3. Encrypting/Decrypting

For the encryption/decryption of the secret message, we needed a symmetric encryption algorithm that would be hard to break with the currently available technology.

The Data Encryption Algorithm (DEA), often called Data Encryption Standard (DES), is one of the best known and widely used symmetric algorithms in the world. DES is commonly used for single-user encryption, such as to store files on a hard disk in encrypted form. However, DES uses a 56-bit key, which is vulnerable to brute-force attacks with today’s rapidly advancing technology. Actually, a DES cracking machine has already been used to recover a DES key in less than 24 hours (Electronic Frontier Foundation 1998).

On the other hand, triple-DES is a variation of the DES. Although triple-DES is three times slower than regular DES, it is \(5 \times 10^{33}\) (5 Billion Trillion Trillion) times harder to break. Consequently, we decided to use the Triple-DES for the encryption of the secret message.

Further information on the compression libraries, which have been under consideration, is available at the following URLs:

3. [http://www.zlib.org](http://www.zlib.org)
The ANSI X9.52 standard defines triple-DES encryption with keys \( k_1, k_2, k_3 \) as:

\[
C = E_{k_3} \left( D_{k_2} \left( E_{k_1} (M) \right) \right)
\]

where \( E_k \) and \( D_k \) denote DES-encryption and DES-decryption, respectively, with the key \( k \). This mode of encryption is sometimes referred to as \( DES-EDE \).

Consequently, three DES computations with three separate keys are being performed. Obviously, the key length is effectively increased to 168-bit. The exact procedure for encryption is the following one: the plaintext is encrypted with the first key, decrypted with the second key and finally encrypted again with the third key. The procedure for encrypting data with the triple-DES is illustrated in the following figure:

The procedure for decrypting data is the same as the procedure for encrypting data, except that it is executed in the reverse order.

For the triple-DES encryption of the secret message, we have chosen to produce the keys \( k_1, k_2, k_3 \) from a password; the user is prompted to enter a password, which gets transformed into the three keys. One of the main requirements is that the keys have to be cryptographically strong; if all three keys, the first and second keys or the second and third keys are the same, then the encryption procedure will actually be the same as regular DES. To ensure that all three keys would be cryptographically strong, a hash function had to be used.

A hash function \( H \) is a transformation that takes an input of an arbitrary-length and returns a fixed-length string, which is called the hash value \( h \). A hash function is said to be one way if it is hard to invert, meaning that given a hash value \( h \), it is computationally infeasible to find some input \( x \) such that \( H(x) = h \). Moreover, if it is computationally infeasible to find any two messages \( x \) and \( y \) such that \( H(x) = H(y) \), then the hash function is said to be strongly collision-free. Actually, the hash value represents concisely the longer message or document from which it was computed; this value is called the message digest.
When considering which hash function to use, one of our main requirements was that it should be secure against brute-force collision and inversion attacks. The hash functions that have been under consideration are the following ones: Message Digest 4 (MD4), Message Digest 5 (MD5), and the Secure Hash Algorithm (SHA-1).

MD4 and MD5 take a message of arbitrary length and produce a 128-bit message digest, whereas the SHA-1 produces a 160-bit message digest. MD4 is the least secure hash algorithm; Dobbertin (1995) has shown how collisions for the full version of MD4 can be found in under a minute, on a typical PC. Moreover, he has shown that a reduced version of MD4 is not one-way. Clearly, MD4 should now be considered broken. MD5 is more secure than MD4; in fact, it has not been broken so far. SHA-1 is slightly slower than MD5, but the larger message digest makes it more secure against brute-force collision and inversion attacks (RSA Laboratories 2000). Consequently, we have chosen to use the Secure Hash Algorithm (SHA-1).

Clearly, we had to look for freely available, written in ANSI C, implementations of the triple-DES and the SHA-1 algorithms. The main requirements were that both of the implementations should have been extensively tested, approved and widely used by others. Thus, we asked for advice from the sci.crypt newsgroup and decided to use the triple-DES implementation by Young (1997) and the SHA-1 implementation by Gillogly (1994).

To sum up, when a secret message is about to be encrypted, the user is prompted to enter a password. Next, the SHA-1 is applied to the password producing a 160-bit output, as described earlier. The keys required for the triple-DES encryption are taken from that hash. Thus, the user must only memorise a password, which gets transformed into the three keys used for the triple-DES encryption. Likewise, when an extracted message is about to be decrypted, the user must provide a password, which gets transformed into the keys required for the decryption.

4.4. Embedding/Extracting

The steganographic tool implemented is an “Image Domain” tool, meaning that it uses some bitwise method, applying Least Significant Bit (LSB) insertion. There are three options available when embedding a secret message into the cover-image: embedding either 1, 2 or 4 secret message bits into each cover-image byte. Thus, the embedding capacity can be approximately equal to 12.5% or 25% or 50% of the cover-image size. However, when 4 secret message bits are embedded into each cover-image byte, the stego-image fidelity degrades significantly. The same options are also available when extracting a secret message from a stego-image. Therefore, it is important to know how many secret message bits were embedded into each cover-image byte, before extracting the secret message from the stego-image.

The number of secret message bits that are embedded into each cover-image byte or extracted from each stego-image byte can be set by modifying the \textit{LSB symbolic constant} in \texttt{general.h}. As discussed above, the LSB symbolic constant can only be set to 1, 2 or 4 bits.
4.4.1. The *.bmp file format

The *.bmp file format is used to define device-independent bitmaps in various colour resolutions. Its main purpose is to allow bitmaps to be moved from one device to another.

The current *.bmp format supports four colour resolutions: 1-bit, 4-bit, 8-bit, and 24-bit. In 1-bit, 4-bit, and 8-bit bitmap files, the pixels are defined by indices into a colour table. The 24-bit version requires no colour table, because it consists of a list of RGB (Red, Green, Blue) values, each in the range 0-255.

4.4.1.1. 8-bit bitmap files

Each 8-bit bitmap file contains a bitmap-file header, a bitmap-information header, a colour table, and an array of bytes that defines the bitmap bits. The file has the following form:

    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;
    RGBQUAD aColours[];
    BYTE aBitmapBits[];

The bitmap-file header contains information about the bitmap file; that information is only about the file, and not about the bitmap itself. The bitmap-information header specifies the dimensions, compression type, and colour format for the bitmap. The bitmap-file header and the bitmap-information header are 14 and 40 bytes long respectively. Thus, the bitmap header is totally 54 bytes long.

The colour table, defined as an array of RGBQUAD structures, contains as many elements as there are colours in the bitmap. The colours in the table appear in order of importance. The RGBQUAD structure has the following definition:

    typedef struct tagRGBQUAD {
        BYTE rgbBlue;
        BYTE rgbGreen;
        BYTE rgbRed;
        BYTE rgbReserved;
    } RGBQUAD;

Obviously, rgbBlue, rgbGreen and rgbRed specify the intensities of blue, green, and red respectively in the colour. Moreover, rgbReserved is always set to zero.

The number of unique colours in a 8-bit bitmap can be up to 256 (=2^8); thus, the maximum number of colour table elements is equal to 256. Taking into consideration that 4 bytes are reserved (in the colour table) for each unique colour, the maximum size of the colour table will be equal to 1024 bytes (=256 x 4 bytes).
The bitmap bits, immediately following the colour table, consist of an array of BYTE values representing consecutive rows, or scan lines of the bitmap. Each scan line consists of consecutive bytes representing the pixels in the scan line, in left-to-right order. The number of bytes representing a scan line depends on the colour format and the width (in pixels) of the bitmap. The scan lines in the bitmap are stored from bottom-up. This means that the first byte in the array represents the pixels in the lower-left corner of the bitmap and the last byte represents the pixels in the upper-right corner.

Pixels displayed on the screen

Pixels stored in *.bmp file

4.4.1.2. 24-bit (true colour) bitmap files

Each 24-bit bitmap file contains a bitmap-file header, a bitmap-information header and an array of bytes that defines the bitmap bits. The file has the following form:

\[
\text{BITMAPFILEHEADER bmfh;}
\text{BITMAPINFOHEADER bmih;}
\text{BYTE aBitmapBits[];}
\]

There is no colour table required here; the bitmap data consists of a list of RGB (Red, Green, Blue) values, each in the range 0-255.

➢ Further information on the *.bmp file format is available in Appendix A.

4.4.2. Embedding

When embedding secret message data into a cover-image, the image header cannot be modified; if that happens, the cover-image will be destroyed. Hence, only the bytes that represent the cover-image pixels can be modified, making sure that the header remains intact. Obviously, the number of bytes in question is equal to:

\[
\text{sizeof(cover-image) - sizeof(cover-image header)}
\]

and when the cover-image is a true colour (24-bit) *.bmp file the above expression becomes:

\[
\text{sizeof(cover-image) - 54 bytes}
\]

\[1\text{sizeof(…)} \text{ function returns size in bytes}\]
When referring to the *embedding capacity* of the cover-image, we mean the maximum amount of data that can be embedded into the cover-image. When embedding *only one* secret-message bit into each cover-image byte, the embedding capacity of the cover-image (measured in bytes) is equal to:

\[
\text{CapacityOf}(\text{cover-image}) = \frac{\text{SizeOf}(\text{cover-image}) - 54 \text{ bytes}}{8}
\]

It is clear that, the embedded message’s size will not always be the same, neither will it always be equal to the embedding capacity of the cover-image. If only the secret message was embedded into the cover-image, it would be impossible to extract it later on, being unable to distinguish the stego-image bytes that hide information from the other ones. One way of getting around that problem is by embedding the secret message size and the secret message data together, into the cover-image. In the *ANSI C* programming language, the system call `stat()` takes a filename and returns various information about the file; the file size is represented by a 4-byte variable. Thus, a 4-byte variable is required for storing the secret message size; its value is embedded into the cover-image, right after the bitmap header. The resulting stego-image will have the following structure:

```
<table>
<thead>
<tr>
<th>Secret Message Size</th>
<th>Secret Message Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 45 55 86 87</td>
<td></td>
</tr>
</tbody>
</table>
```

STEGO-IMAGE (24-bit)  

BITMAP HEADER  

BITMAP DATA  

Obviously, the *secret message size* is hidden in the LSBs of the stego-image bytes numbered from 55 to 86 and the *secret message data* is hidden in the LSBs of the following stego-image bytes. Then, the embedding capacity of the cover-image changes into:

\[
\text{CapacityOf}(\text{cover-image}) = \frac{\text{SizeOf}(\text{cover-image}) - 54 \text{ bytes} - 32 \text{ bytes}}{8}
\]

In a similar way, when embedding *two* secret-message bits into each cover-image byte, the embedding capacity of the cover-image becomes equal to:

\[
\text{CapacityOf}(\text{cover-image}) = \frac{\text{SizeOf}(\text{cover-image}) - 54 \text{ bytes} - 16 \text{ bytes}}{4}
\]

Clearly, the generic equation for the embedding capacity of the cover-image is the following one:

\[
\text{CapacityOf}(\text{cover-image}) = \frac{\text{SizeOf}(\text{cover-image}) - 54 \text{ bytes} - \frac{32}{\text{LSB}}} {\frac{8}{\text{LSB}}}
\]

where LSB represents the number of secret message bits embedded into each cover-image byte and can only be equal to 1, 2 or 4.
The steganographic tool implemented can also embed information into 8-bit (256 colours) cover-images. However, there is a slight difference in the way data is being embedded: the indices to the colour table are manipulated in this particular case, instead of the RGB values for the cover-image pixels. Given that the bitmap header is 54 bytes long and the colour table can only be up to 1024 bytes long, the resulting stego-image will have the following structure:

![Stego-image structure diagram](image)

Obviously, the secret message size is hidden in the LSBs of the stego-image bytes numbered from 1079 to 1110 and the secret message data is hidden in the LSBs of the following stego-image bytes.

Note: The symbolic constant HEADER in file *general.h* should be set to 1078, before embedding a secret message into a 8-bit cover-image.

### 4.4.3. Extracting

When we wish to extract an embedded message from a stego-image, the first thing we need to know is the embedded message’s size. Thus, the LSBs from the 32 stego-image bytes following the bitmap header (bytes 55-86) must be extracted, which constitute the embedded message’s size. Next, the LSBs of the remaining stego-image bytes will be extracted, forming the embedded message.

Let us suppose that an image, which does not contain any hidden information, is given as input to the extracting function. In that case, the LSBs from bytes 55-86 will still be extracted, being interpreted as the embedded message’s size. There are two possible cases:

i.) If that size happens to be less than or equal to the image capacity, the extracting function will execute normally. However, the “extracted message” will not make any sense, as it will actually consist of random bits. Consequently, it will be impossible to decrypt or decompress that particular message.

ii.) If the extracted size is greater than the image capacity, the extracting function will continue to execute until it encounters the EOF indicator. Then, the program will terminate, displaying an appropriate error message.
4.4.4. Pseudocode

Pseudocode is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language. It allows the designer to focus on the logic of the algorithm, without being distracted by details of language syntax. Unfortunately, there is no universal standard of pseudocode for the software industry. Here, the pseudocode standard proposed by Dalbey (2001) will be used.

The pseudocode for embedding a secret message into a cover-image is the following one:

GET size of secret message
GET size of cover-image
COMPUTE capacity of cover-image

IF size of secret message > capacity of cover image THEN
    PRINT Error!
    EXIT
ELSE
    COPY header of cover-image
    FOR i=1 TO 32 bits of secret message size
        READ next cover-image byte
        ADJUST LSB of cover-image byte
        WRITE cover-image byte to output
    ENDFOR
    FOR each bit of secret message
        READ next cover-image byte
        ADJUST LSB of cover-image byte
        WRITE cover-image byte to output
    ENDFOR
    COPY the rest of cover image bytes to output
ENDIF

Note: The above pseudocode is valid when embedding only one bit into each cover-image byte. When embedding two or four bits into each cover-image byte, there are minor changes in the FOR loops.
The pseudocode for extracting the embedded message from the stego-image is the following one:

Skip over the stego-image header

\[
\text{FOR } i=1 \text{ TO } 32 \text{ bytes of stego-image} \\
\quad \text{READ next stego-image byte} \\
\quad \text{GET its LSB} \\
\quad \text{WRITE LSB to size_bits[i]} \\
\text{ENDFOR}
\]

\begin{align*}
\text{Extract Size of Embedded Message} \\
\text{FOR } i=1 \text{ TO size of embedded message} \\
\quad \text{FOR } j=1 \text{ TO } 8 \\
\quad \quad \text{READ next stego-image byte} \\
\quad \quad \text{GET its LSB} \\
\quad \quad \text{WRITE LSB to scr_msg_bits[j]} \\
\text{ENDFOR} \\
\text{COMPUTE embedded message byte from scr_msg_bits[j]} \\
\text{WRITE embedded message byte to output}
\end{align*}

\text{ENDFOR}

Note: The above pseudocode is valid when only one secret message bit is embedded into each stego-image byte. For example, when two secret message bits are embedded into each stego-image byte, the FOR loops change into the following ones:

\[
\text{FOR } i=1 \text{ TO } 16 \text{ bytes of stego-image} \\
\quad \text{READ next stego-image byte} \\
\quad \text{GET its two LSBs} \\
\quad \text{WRITE those two bits to size_bits[i]} \\
\text{ENDFOR}
\]

\text{COMPUTE ...}

\[
\text{FOR } i=1 \text{ TO size of embedded message} \\
\quad \text{FOR } j=1 \text{ TO } 4 \\
\quad \quad \text{READ next stego-image byte} \\
\quad \quad \text{GET its two LSBs} \\
\quad \quad \text{WRITE those two bits to scr_msg_bits[j]} \\
\text{ENDFOR} \\
\text{COMPUTE ...} \\
\text{WRITE ...}
\]

\text{ENDFOR}
4.4.5. Flow Charts

A flow chart is a formalized graphic representation of a program logic sequence, work or manufacturing process, organization chart, or similar formalized structure. In computer programming, flow charts are used to describe each processing path in a program. Drawing flow charts involves the use of simple geometric symbols, which are defined in ANSI X3.5 and ISO 1028.

Every flow chart has a START symbol; it represents the beginning of a program.

The PROCESS symbol has only one entry point and one exit point.

The SUBPROCESS symbol is really useful because it permits the modularisation of complex programs.

The DECISION symbol has only one entry point, one TRUE exit point and one FALSE exit point.

Every flow chart has a STOP symbol; it represents the end of a program.
START

Get secret message size

Get cover-image size

Compute cover-image capacity

secret message size > cover-image capacity

FALSE

Copy cover-image header

Embed secret message size

Embed secret message bits

Copy the rest of cover-image bytes

Stego-image

STOP

TRUE

Error!

STOP

Embedding Secret Message
Overall Flow Chart
“Embed secret-message size” subprocess:

```
START
FOR i=1 TO 32 bits of secret message size
Read next cover-image byte
Adjust LSB of cover-image byte
Write cover-image byte to output
STOP
```

“Embed secret-message bits” subprocess:

```
START
FOR each bit of secret message
Read next cover-image byte
Adjust LSB of cover-image byte
Write cover-image byte to output
STOP
```
START

Skip over the stego-image header

FOR i=1 TO 32 bytes of stego-image

Read next stego-image byte

Get its LSB

Write LSB to size_bits[i]

Compute size of embedded message from size_bits[]

FOR i=1 TO size of embedded message

FOR j=1 TO 8

Read next stego-image byte

Get its LSB

Write LSB to scr_msg_bits[]

Compute secret message byte from scr_msg_bits[]

Write byte to output

STOP

Extracting Secret Message
Overall Flow Chart
5. Evaluation of the Steganographic Tool Implemented

The steganographic tool implemented complies with the specifications that have been initially set: it can compress, encrypt and then embed a secret message into a cover-image or extract an embedded message from a stego-image, decrypt it and then decompress it.

Additionally, the steganographic tool has a user-friendly interface, displaying various information when embedding or extracting a secret message. For example, during the embedding process, it displays the size of the secret message before and after compression, the size of the cover-image and its embedding capacity; if the secret message does not fit into the cover-image, it issues an appropriate error message.

However, the major question is whether the steganographic tool achieves the primary goal of steganography. As mentioned in the introduction, the primary goal of steganography is to avoid drawing suspicion to the transmission of a hidden message, so it will not be detected. If suspicion is raised, then this goal is defeated.

In order to determine the limitations and flexibility of the steganographic tool, it has been tested on the following forms of attacks: detecting, extracting and disabling hidden information (Katzenbeisser & Petitcolas 2000, p.80).

5.1. Detecting Hidden Information

One method for detecting the existence of hidden messages in stego-images is to look for obvious and repetitive patterns, which may point to the identification or signature of a steganography tool or hidden message (Johnson & Jajodia 1998b).

Obviously, distortions or patterns visible to the human eye are the easiest to detect. The approach used to identify such patterns was to compare a large number of cover-images with the respective stego-images and note visible differences (known cover attack\(^1\)).

The cover-images used in these tests include digital photographs, digital art and clip art. The digital photographs are typically 24-bit bitmaps with thousands of colours or 8-bit greyscale. Digital art images are not photographs, but they may also have thousands of colours. Clip art images have relatively few colours and are typically 8-bit bitmaps. Where necessary, the cover-images were converted from 24-bit into 8-bit colour resolution or the opposite, using Paint Shop Pro version 7.0.

For the purposes of this dissertation, only the results that we obtained with three cover-images are being analysed.

\(^1\) For further information on the various forms of attacks, please refer to Chapter 2, paragraph 2.2
5.1.1. 24-bit images

The first cover-image that we used is a 24-bit digital photograph, as shown in Figure 1.1.

![Figure 1.1](Image)

The dimensions of the original cover-image are 712x521 pixels and the file size is 1.112.910 bytes. The embedding capacity of the cover-image is equal to: 139.103 bytes, 278.210 bytes or 556.424 bytes when embedding 1, 2 or 4 secret message bits respectively into each cover-image byte.

It is clear that, the probability of introducing detectable artifacts by the embedding process, becomes smaller when embedding less information. Another important factor is the choice of the cover image; some images may become grossly degraded, even when embedding small amounts of information. To test if the particular image is a good cover, we embedded an Adobe Acrobat Document file, which had a total size of 154.582 bytes. Before the file was embedded into the cover-image it was compressed and encrypted, having a total size of 139.096 bytes. Obviously, the size of the embedded message is approximately the 99,99%, 49,99% or 24,99% of the embedding capacity of the cover-image, when embedding 1, 2 or 4 secret message bits respectively into each cover-image byte.

When hiding information only in the LSB of each cover-image byte, there are no visible distortions in the resulting stego-image. Similarly, when embedding data into the least and second least significant bits of each cover-image byte, there are no detectable artifacts. To the human eye, the resulting stego-images look identical to the cover-image. However, the cover-image contains 136.752 unique colours, whereas the first stego-image (hiding 1 bit of information into each stego-image byte) contains 143.855 unique colours and the second stego-image (hiding 2 bits of information into each stego-image byte) contains 143.298 unique colours. Obviously, if a steganalyst has the cover-image available, he will be able to compare it to the stego-images and notice the variation that we just mentioned. Without the benefit of such a comparison, the variation in the number of unique colours would go unnoticed.
On the other hand, when embedding 4 bits of information into each cover-image byte, distortion occurs at the bottom of the resulting stego-image (Figure 1.2).

![Figure 1.2](image1.png)

In particular, when the stego-image is displayed on a computer screen, the artifacts introduced by the embedding process are easily noticeable. However, when the stego-image is printed, the distortions are not so obvious. For that particular reason, the lower-left corner of the stego-image is shown magnified in Figure 1.3.

![Figure 1.3](image2.png)

Furthermore, the stego-image shown in Figure 1.2 contains 154,316 unique colours, whereas the cover-image contains only 136,752 unique colours. That means 154,316-136,752 = 17,564 unique colours were introduced by the embedding process.

From that variation in unique colours, a rough approximation of the embedded message’s size can be given. Since 17,564 unique colours were introduced and because each colour is represented by three bytes (RGB), at least 17,564x3 = 52,692 cover-image bytes were modified by the embedding process. If a steganalyst makes the assumption that 4 bits of information were embedded into each cover-image byte, then he will conclude that the size of the embedded message is at least 52,692x4 = 210,768 bits or equivalently 26.346 bytes.
The second cover that we used is a 24-bit digital art image, as shown in Figure 1.4.

![SQUEAK ATTACK](image)

Figure 1.4

The dimensions of the original cover-image are 700x542 pixels and the file size is 1.138.254 bytes. The embedding capacity of the cover-image is equal to: 142.271 bytes, 284.546 bytes or 569.096 bytes when embedding 1, 2 or 4 secret message bits respectively into each cover-image byte.

To find out if the particular image is a good cover, we embedded various messages, with each one of them having different size. It has been observed that, when 1 or 2 bits of secret message-data are embedded into each cover-image byte, there are no visible distortions in the resulting stego-image, regardless of the embedded message’s size. Actually, the cover-image and the stego-images look as if they are the same.

On the other hand, when 4 bits of secret-message data are embedded into each cover-image byte, the resulting stego-image contains artifacts. For example, when the Adobe Acrobat Document was embedded (the one mentioned previously), some slight distortions were detected at the bottom of the stego-image. However, those distortions could only be noticed after comparing the stego-image to the cover-image.

Additionally, a variation in the number of unique colours contained in the cover-image and the resulting stego-images has been observed. In particular, the cover-image contains 64.658 unique colours, whereas the first stego-image (1 bit embedded into each cover-image byte) contains 66.288 unique colours, the second stego-image (2 bits embedded into each cover-image byte) contains 67.165 unique colours and the third stego-image contains 71.241 unique colours. That variation in the number of unique colours may raise a steganalist’s suspicion and the purpose of steganography will be defeated.

If the cover-image was not available, it would be impossible to notice the variation in the number of unique colours. Furthermore, the distortion introduced into the stego-image would look like JPEG compression noise, if it was not compared to the cover-image.
The third cover that we used is a 24-bit clip art image, as shown in Figure 1.5. The colour resolution of the particular cover-image was originally 8-bit, but it was converted into 24-bit, for the purposes of this evaluation.

The dimensions of the cover-image are 134x271 pixels and the file size is 151.898 bytes. The embedding capacity of the cover-image is equal to: 18.976 bytes, 37.957 bytes or 75.918 bytes when embedding 1, 2 or 4 secret message bits respectively into each cover-image byte.

As in the two previous cases, there were no visible distortions in the resulting stego-images, when hiding 1 or 2 bits of secret message data into each cover-image byte. Additionally, there were detectable artifacts introduced into the stego-image, when 4 bits of information were embedded into each cover-image byte. However, those artifacts would be hardly detected, if the stego-image was not compared to the cover-image.

The file that we embedded into the cover-image was a JPEG picture, which had a total size of 20.670 bytes. Before the file was embedded into the cover-image it was compressed and encrypted, having a total size of 14.752 bytes. Obviously, the size of the embedded message is approximately the 77.74%, 38.87% or 19.43% of the embedding capacity of the cover-image, when embedding 1, 2 or 4 secret message bits respectively into each cover-image byte.

In that case, there was a huge variation in the number of unique colours contained in the cover-image and the resulting stego-images. In particular, the cover-image contained only 255 unique colours, whereas the first, the second and the third stego-image contained 1.540, 2.999 and 6.757 unique colours respectively. Obviously, there was an increase of 604%, 1.176% and 2.649% in the number of unique colours of the cover-image. If a steganalyst had noticed this variation, he would conclude that the cover-image contains hidden information. Thus, it would be wiser to avoid using a clip art image as a cover.
A more efficient way of detecting hidden information in an image is the visual attack, which has been proposed by Westfeld & Pfitzmann (2000). The main concept of a visual attack is to unveil the bits that may constitute a secret message, by removing the image bits that camouflage it. Then, the human eye can distinguish whether those bits constitute a secret message, or genuine image content. The block diagram of a visual attack is shown in the following figure:

Let us illustrate some visual attacks: some of them on the clip art cover-image (previously examined) and some of them on the resulting stego-images:

Figure 1.6 illustrates the visual attack on the cover-image, where the four most significant bits from each cover-image byte have been removed, assuming that a secret message could have been embedded in the four least significant bits. As we observe in Figure 1.6, the outline of Lucky Luke and his horse remain intact. Consequently, there is no secret message embedded in the four LSBs of the image in question.

On the other hand, when we perform a visual attack on a stego-image, that contains hidden information in the four LSBs of each stego-image byte, we get the image shown in Figure 1.7. The embedded message in the particular stego-image has a total size of 20.670 bytes, which is actually the 19.43% of the embedding capacity of the cover-image. It is clear that, the noise at the bottom of the image (Figure 1.7) is due to the embedded message. Thus, if a steganalyst performs a visual attack on the particular stego-image, he will easily detect the existence of a hidden message.
In a similar way, we can apply a visual attack on the same cover-image, assuming this time that two bits of secret message data are embedded into each cover-image byte. Thus, the six most significant bits from each cover-image byte are removed, when applying this particular visual attack. The resulting image (we will also refer to it as filtered image) is the one shown in Figure 1.8. Obviously, there is no embedded message in the image under visual attack, since the outline of Lucky Luke and his horse remain intact in the filtered image.

![Figure 1.8](image1)

![Figure 1.9](image2)

On the contrary, when we apply a visual attack on the respective stego-image (which contains hidden information in the two LSBs of each stego-image byte) there is some noise noticed at the bottom of the filtered image (Figure 1.9). Obviously, the noise in the filtered image is due to the embedded message.

Moreover, a thorough examination of the filtered image reveals that the noise takes up approximately the 40% of the filtered image. It must be stressed though, that this 40% of the filtered image is not a random number; it is a rough approximation of the amount of hidden information in the stego-image. Indeed, the embedded message in the particular stego-image has a total size of 20.670 bytes, which is actually the 38.87% of the embedding capacity of the cover-image. Consequently, if a steganalyst performs a visual attack on the particular stego-image, not only will he detect the existence of a hidden message, but he will also be able to estimate the size of the embedded message.

In the same way, we can assume that one bit of secret message data is hidden into each stego-image byte and apply the respective visual attack: the seven most significant bits will be removed and only the least significant bit (from each stego-image byte) will be forming each filtered stego-image byte. In fact, the hidden message is easily detected, due to the noise appearing in the filtered image. Unfortunately, the filtered stego-image is too blurry to be printed here; it can only be displayed on the computer screen.
Furthermore, the digital art cover-image has been tested against the visual attacks and it was found to be vulnerable; not only was it easy to detect the hidden information in the stego-images, but it was also possible to estimate the size of the embedded size for each case.

*Figure 1.10* illustrates the filtered cover-image (where the 4 MSBs have been removed) and *Figure 1.11* the respective stego-image. In this case, the embedded message is 139.096 bytes, which is approximately the 24.44% of the embedding capacity of the cover-image.

*Figure 1.12* illustrates the filtered cover-image (where the 6 MSBs have been removed) and *Figure 1.13* the respective stego-image. In this case, the embedded message is approximately the 48.88% of the embedding capacity of the cover-image.

Additionally, when the visual attack was performed on the stego-image hiding only one bit of information into each stego-image byte, it has also proved to be vulnerable. The hidden message was detectable, because of the noise in the filtered image. Unfortunately, the filtered stego-image is too blurry to be printed here; it can only be displayed on the computer screen.

Obviously, the computer art image should be avoided as a cover-image; it does not require an experienced steganalyst to detect the hidden information in the stego-images, but anyone with moderate skills in steganalysis.
The visual attacks have also been applied to the digital photo cover-image, which has proved to be the best cover, when being compared to the previously tested ones.

Figure 1.14 illustrates the filtered cover-image (where the 4 MSBs have been removed) and Figure 1.15 the respective stego-image. Obviously, the noise at the bottom of the filtered image implies the existence of hidden information. Moreover, the amount of hidden information can be estimated. In fact, the embedded message is 139.096 bytes, which is approximately the 24.99% of the embedding capacity of the cover-image.

However, it was not possible to detect the existence of hidden information in the stego-image, when only one or two bits of secret message data were embedded into each image byte. In particular, the visual attack was first applied to the cover-image, having the six most significant bits removed. Nevertheless, the filtered image was plain noise; we could neither distinguish the outlook of the flowers, nor any other pattern resembling the original image. Obviously, when the visual attack was applied to the stego-image, the filtered stego-image looked the same to the filtered cover-image: we could distinguish nothing else but noise. Thus, the detection of the embedded message (using a visual attack) was impossible.

Figure 1.16 illustrates the filtered cover-image (where the 6 MSBs have been removed) and Figure 1.17 the respective stego-image. In this case, the embedded message is approximately the 49.99% of the embedding capacity of the cover-image. Despite that fact, the embedded message could not be detected.
5.1.2. 8-bit images

When embedding data into 8-bit cover-images, the LSBs of the colour indices are modified, without the colour table being altered. Consequently, the colour indices are changed from one entry in the colour table to another, causing colour shifts. If the adjacent colours in the colour table are very similar, there is little or no noticeable change. However, if adjacent colour table entries are not similar, then the noise because of the LSB manipulation is rather obvious.

The first cover-image is the digital photo that has been previously tested (illustrated in Figure 1.1), converted\(^1\) into 8-bit colour resolution, for the purposes of this particular test. The colour table of the 8-bit digital photo is shown in Figure 2.1.

\[\text{Figure 2.1}\]

It is clear that the adjacent colours in the colour table are quite similar; thus, the distortions introduced into the cover-image by the embedding process should not be significant. Indeed, when embedding only one bit of information into each cover-image byte, the distortions in the resulting stego-image are minor and cannot be easily noticed. However, those slight distortions jump out of the stego-image when compared with the cover-image.

On the other hand, when two or four bits of secret message data are embedded into each cover-image byte, there are severe distortions introduced into the resulting stego-image. In fact, those distortions are colour shifts caused by the modification of the colour indices in the cover-image. Nevertheless, the distortions may be reduced if we modify the colour table of the cover-image, before embedding the secret message. Actually, the colour table could be rearranged so as to reduce the occurrence of adjacent colour entries that contrast too much. Furthermore, a new colour table could be made. However, it has been reported (Johnson & Jajodia 1998b) that any changes in the colour table can be detected.

\(^{1}\) *Paint Shop Pro version 7.0* has been used for the conversion of the cover-image from 24-bit to 8-bit.
The stego-image which contains four bits of secret message data into each stego-image byte is illustrated in Figure 2.2. In this case, the size of the embedded message is approximately the 20.87% of the embedding capacity of the cover-image. The artifacts introduced by the embedding process are easily noticeable (at the bottom of the image). Additionally, the lower-left corner of the stego-image is shown magnified in Figure 2.3.

![Figure 2.2](image1.png)

![Figure 2.3](image2.png)

The second cover is the digital art image that has been previously tested (illustrated in Figure 1.4), converted into 8-bit colour resolution. The colour table of the 8-bit digital art cover-image is shown in Figure 2.4.

![Figure 2.4](image3.png)

It is clear that the adjacent colours in the colour table are not contrasting; consequently, there should not be any significant artifacts introduced into the cover-image by the embedding process. In fact, when embedding only one bit of information into each cover-image byte, the artifacts can only be noticed after comparing the cover-image with the resulting stego-image. However, when two or four bits of secret message data are embedded into each cover-image byte, the artifacts in the stego-image are quite obvious.
The stego-image containing four bits of secret message data into each stego-image byte is illustrated in Figure 2.5. In this case, the size of the embedded message is approximately the 20.4% of the embedding capacity of the cover-image. The artifacts introduced by the embedding process are easily detectable (at the bottom of the image). Moreover, the bottom of the stego-image is shown magnified in Figure 2.6.

The third cover is the clip art image that has been previously tested (illustrated in Figure 1.5). The colour table of the image in question is shown in Figure 2.7.

Obviously, the adjacent colours in the colour table are vastly contrasting. Therefore, small shifts in the colour indices will cause dramatic colour changes in the clip art image. Indeed, the colour changes introduced by the embedding process are quite intense, even when only the LSB of each colour index is modified. Consequently, the clip art image is not appropriate for hiding information, because the colour changes in the image would advertise the existence of a hidden message.
The stego-images having one, two or four bits of secret message data embedded into each colour index are illustrated in *Figures 2.8, 2.9 and 2.10* respectively. In this case, the size of the embedded message is approximately the 37.39%, 18.69% or 9.34% of the embedding capacity of the cover-image.

As we can see from the above images, the distortions introduced by the embedding process are so severe, that the original image has actually been ruined. One way of getting around that problem is to convert the clip art image into *greyscale (256 shades of grey)*. In fact, when embedding data into 8-bit greyscale images, the differences between the cover and stego-images are not so obvious. That happens because the greyscale values change gradually from one entry in the colour table to another. The greyscale clip art image and its colour table are illustrated in *Figures 2.11 and 2.12* respectively.
### 5.1.3. Conclusions

We have tested the steganographic tool embedding various messages into different cover-images and have concluded the following:

i.) The less information we embed into the cover image, the smaller the probability of introducing detectable artifacts by the embedding process.

ii.) The choice of the cover-image is very important. The best covers are made from 24-bit images, especially uncompressed scans of photographs or images obtained with a digital camera, containing a high number of colours; digital art and clip art images should be avoided. Moreover, 8-bit images are not a wise choice, as they become significantly degraded with even small amounts of embedded information.

Furthermore, Fridrich et al (2001a) have argued that images stored previously in the JPEG format are a poor choice for cover images. This is because the quantization introduced by JPEG compression can serve as a watermark or unique fingerprint; as a result, even the smallest modifications of the cover-image can be detected, by inspecting the compatibility of the stego-image with the JPEG format.

It is of major importance that the stego-image does not contain any detectable artifacts, due to message embedding. A steganalyst could use such artifacts as an indication that a secret message is present. Once a steganalyst can reliably identify which images contain hidden information, the steganographic tool becomes useless.

### 5.2. Extracting Hidden Information

Once a steganalyst detects the existence of hidden information in a cover-image, he may wish to extract the embedded message. In case he knows the method used by the steganographic tool to embed a secret message in a cover-image, then it is fairly easy to extract it. Moreover, if he lacks knowledge of how the secret message was embedded into the cover-image, he can guess that it was embedded sequentially into the LSBs of the cover-image bytes, since that is the most widely used method by steganographic tools for embedding information.

In reality, if the steganalyst succeeds in extracting the hidden message, he will have to apply cryptanalysis techniques, in order to decrypt the extracted message. As far as the triple-DES algorithm is considered, it is almost impossible to be broken with the currently available technology. The only viable way of decrypting the extracted message would be to apply brute force or dictionary attacks, in order to find the password required for the decryption. In a dictionary attack, the attacker takes a list of some common passwords and applies the hash function (SHA-1) to them, trying to find the three keys required for the triple-DES decryption of the extracted message. Consequently, it is of major importance that we choose a password which is not easy to guess, for the encryption of the secret message.
5.3. Disabling Hidden Information

As discussed above, if the existence of hidden information is detected, then steganography is defeated. However, it may be desirable to let the stego-image pass along the communication channel, but disable the embedded message. Moreover, any image can be manipulated with the purpose of disabling or destroying some hidden information, whether an embedded message exists or not.

To evaluate the tolerance of an embedded message to stego-image manipulation, we devised a series of image processing tests. The 24-bit and 8-bit colour versions of the digital photo, digital art and clip art covers (illustrated in Figures 1.1, 1.4, 1.5) were used for those tests. The cover-images were embedded with known messages and the resulting stego-images were verified for the message contents. Next, the stego-images were manipulated with a number of image processing techniques and checked for the message contents. All tests were conducted using Paint Shop Pro version 7.0 and they included:

- Converting between lossless and lossy formats
- Converting between bit densities (24-bit, 8-bit, greyscale)
- Flipping
- Mirroring
- Rotating
- Blurring
- Sharpening
- Edge enhancement
- Adding noise
- Removing noise

Actually, all of the above tests altered the embedded information, to the point that it could not be retrieved. Simply converting a stego-image to a lossy format (such as JPEG) and then back was enough to render the embedded message useless; although both images seemed identical to the human eye, the processed stego-image no longer contained the hidden information.

Unfortunately, it was proved that it is extremely easy to disable the hidden information contained in a stego-image. In fact, this point has been made a number of times (Johnson & Jajodia 1998a, 1998b; Katzenbeisser & Petitcolas 2000); LSB substitution steganographic tools are vulnerable to even slight stego-image manipulations.
5.4. Further Work

In this chapter, we have tried to determine the limitations and flexibility of the steganographic tool implemented, by testing it extensively on various forms of attacks, like detecting, extracting and disabling hidden information.

The cover-images must be carefully selected, as discussed previously; if a steganalyst is able to detect the existence of a secret message, for instance by applying visual attacks, then the steganographic system should be considered insecure. On the other hand, a steganalyst does not have to detect the existence of a hidden message; he can simply try to extract a potential hidden message from a bitmap image, just as if he were the receiver. Consequently, the steganographic technique implemented does not provide a high level of security. The definition (Katzenbeisser & Petitcolas 2000, p. 9) of a secure stego-system is: “one where an opponent who understands the system, but does not know the key, can obtain no evidence (or even grounds for suspicion) that a communication has taken place”. Unfortunately, the steganographic tool implemented provides security-by-obscurity, meaning that it assumes the enemy will remain ignorant of the system.

One way of dealing with that problem would be to scatter the hidden data throughout the cover-image. Instead of using the LSB of every cover-image byte for information transfer, it is possible to select only some cover-image bytes in a rather random manner according to a secret key and leave the others unchanged. Both Lee & Chen (2000) and Johnson & Jajodia (1998a, p.27) refer to this secret key as a stego-key. Even if the message bits are extracted, they will be useless without the algorithm and stego-key to decode them. The selection of the cover-image bytes that will be hiding information can be done using a pseudorandom number generator. Franz et al. (1996) report a system in which the output of the random number generator is used to spread the sequence of message bits over the cover-image, by determining the number of cover-image bytes which are left unchanged between two bytes used for information transfer.

On the other hand, while scattering helps protect against extracting hidden information, it does not help against disabling hidden information through image processing. A scattered message in the image’s LSBs is still as vulnerable to destruction from lossy compression and image processing as is a message sequentially inserted in the LSBs. Moreover, the embedding capacity of the cover-image reduces, when the hidden data is being scattered throughout the image.

An another disadvantage of the steganographic tool implemented is that it can only hide information in bitmap cover-images; 24-bit (true colour) bitmap images are too big in size to be transmitted over the internet. Moreover, the vast majority of the images available on the internet are GIF or JPEG formatted; thus, a BMP formatted image could raise a steganalyst’s suspicion. Obviously, it would be much better if the steganographic tool could also embed information in images of some other widely used format, like GIF or JPEG.
Conclusions

It is a fact that people have always had the desire to hide certain things of their private life or of business interests. Nowadays, cryptography is the most usual means of communicating privately. However, as discussed in the first chapter, significant restrictions have been imposed on cryptography in many countries around the world. For that reason, we proposed the use of digital steganography as an alternative to cryptography.

As we have seen, steganography is not a rather researched discipline, although it dates back to ancient times. We have reviewed briefly the steganographic techniques used to embed information into digital media, concluding that the \textit{LSB substitution} technique is widely used, being implemented by the majority of steganographic programs. However, nobody has ever made an attempt both to implement and test a LSB substitution steganographic tool. The primary goals of this project were to develop a steganographic tool that implements the LSB substitution technique and test it extensively; only this way could we find out any inherent weaknesses of the particular technique.

This project succeeded in implementing a steganographic tool that compresses, encrypts and then embeds a secret message into a bitmap cover-image, by implementing the LSB substitution technique. Likewise, the steganographic tool successfully extracts an embedded message from a stego-image, decrypts and decompresses it. The design and implementation of the steganographic tool have been examined in detail, giving emphasis to the embedding and extracting functions. Furthermore, the steganographic tool has been extensively tested, by applying various forms of attacks, such as detecting, extracting and destroying an embedded message. We have concluded that the detection of a hidden message depends primarily on the image used for hiding information. Additionally, we have found out that the cover and the stego images look identical to the human eye, when true colour bitmaps are used as covers and only one or two bits of secret message data are embedded into each cover-image byte. On the other hand, we have realised that if someone is aware of the embedding and extracting algorithms implemented by our steganographic tool, he may easily extract an embedded message from a stego-image. Nevertheless, this particular problem can be solved by scattering the hidden data throughout the cover-image. Furthermore, we have tested various stego-images against a series of image processing tests, proving that even the slightest image processing is enough to render the embedded message useless.

The steganographic tool implemented was found to be vulnerable, as far as the message extraction and destruction are concerned. Although it is possible to make it harder for a third party to extract an embedded message, it is impossible to make the steganographic tool more robust to stego-image manipulation. Steganographic tools that implement the LSB substitution technique, including our implementation, do not provide the highest level of security that would be demanded, for instance by intelligence agencies or the army. On the other hand, the particular tools are suitable for individuals that do not require the ultimate level of security, simply wishing to communicate privately.
References

Books


**Webpages**


Dalbey, J. (2001), “Pseudocode Standard” ([http://www.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html](http://www.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html); email: jdalbey@calpoly.edu)


(http://141.59.43.36/rz/www/stego.htm; email: gloeckler@fh-ulm.de)


(http://www.cix.co.uk/~mhayhurst/jdhayhurst/pigeon/pigeon.html)


(http://rechten.kub.nl/koops/cryptolaw/index.htm; email: e.j.koops@kub.nl)

(http://www.jjtc.com/Steganography/steglist.htm; email: mskuhn@cip.informatik.uni-erlangen.de)

Latham, A. (1999), “JPHS” (http://linux01.gwdg.de/~alatham/stego.html; email: alatham@flexsys-group.com)

(http://www.rugeley.demon.co.uk/security/hdsk50.zip)

(http://www.cl.cam.ac.uk/~fapp2/steganography/mp3stego/; email: fapp2@cl.cam.ac.uk)

Repp, H. (1996), “Hide4PGP” (http://www.heinz-repp.onlinehome.de/Hide4PGP.htm; email: heinz.repp@online.de)

(http://www.psionic.com/papers/whitep03.html; email: crowland@psionic.com)


Bibliography

Books


Webpages


Doherty, B.S. (2001), “Notes For Projects” (http://www.aston.ac.uk/~dohertbs/project_notes.htm; email: b.s.doherty@aston.ac.uk)


Petitcolas, F.A.P (2002), “Information Hiding Homepage” (http://www.cl.cam.ac.uk/~fapp2/steganography/; email: fapp2@cl.cam.ac.uk)


Appendix A

Cygwin: Library and Tools

Cygwin is a UNIX environment, developed by Red Hat, for Microsoft Windows. It consists of two parts:

i.) a Dynamic Link Library (cygwin1.dll) which acts as a UNIX emulation layer, providing substantial UNIX API (Application Programming Interface) functionality.

ii.) a collection of tools, ported from UNIX, for Microsoft Windows.

The Cygwin Dynamic Link Library (DLL) works with all modern 32-bit versions of Microsoft Windows, except Windows CE. The Cygwin tools are ports of the popular GNU development tools and utilities for Microsoft Windows. They function through the use of the Cygwin library, which provides the UNIX system calls and environment that these programs require.

With the Cygwin tools installed, programmers may write Win32 console or GUI applications that make use of the standard Microsoft Win32 API and/or the Cygwin API. Consequently, it is possible to easily port many significant UNIX programs, without the need for extensive changes to the source code; this includes configuring and building most of the available GNU software. The Cygwin tools include many standard UNIX utilities and compiler tools; all of them can be used both from the bash shell provided or from the command.com.

It is remarkable that, for the development and testing of the steganographic tool, Cygwin DLL version 1.3.10 has been used. The steganographic tool implemented uses the Cygwin API. Therefore, when executing the steganographic tool from the command.com, the Cygwin DLL (cygwin1.dll) should be included in the same directory as steg.exe.

→ Further information on Cygwin is available at the official website:

http://www.cygwin.com

GNU Compiler Collection (GCC)

Initially GCC stood for GNU C Compiler; nowadays, GCC stands for GNU Compiler Collection and it contains front ends for C, C++, Objective C, Fortran, Java and Ada, as well as libraries for these languages. An open development environment is being used for the development of the GCC, supporting various platforms, to attract a large team of developers, to ensure that GCC works on multiple architectures and diverse environments and to thoroughly test and extend the features of GCC.

→ Further information on GCC is available at the official website:

http://gcc.gnu.org/
The *.bmp format:

Each bitmap file contains a bitmap-file header and a bitmap-information header. The bitmap-file header contains information about the bitmap file; that information is only about the file, and not about the bitmap itself. The bitmap-information header specifies the dimensions, compression type, and colour format for the bitmap. The bitmap-file header and the bitmap-information header are 14 and 40 bytes long respectively. Thus, the bitmap header is totally 54 bytes long. The structures of the bitmap-file and bitmap-information headers are shown at the following tables:

BITMAP-FILE HEADER

<table>
<thead>
<tr>
<th>Start</th>
<th>Size (in bytes)</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>bfType</td>
<td>defines the file type; it must always be 'BM'</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>bfSize</td>
<td>specifies the size of the file in bytes</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>bfReserved1</td>
<td>must always be set to zero</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>bfReserved2</td>
<td>must always be set to zero</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>bfOffBits</td>
<td>Specifies the offset from the beginning of the file to the bitmap data</td>
</tr>
</tbody>
</table>

BITMAP-INFORMATION HEADER

<table>
<thead>
<tr>
<th>Start</th>
<th>Size (in bytes)</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>4</td>
<td>biSize</td>
<td>specifies the size of the BITMAPINFOHEADER structure (in bytes)</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>biWidth</td>
<td>specifies the width of the image (in pixels)</td>
</tr>
<tr>
<td>23</td>
<td>4</td>
<td>biHeight</td>
<td>Specifies the height of the image (in pixels)</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>biPlanes</td>
<td>must always be set to one</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>biBitCount</td>
<td>defines the color resolution (in bits per pixel) of the bitmap-only four values are valid: 1, 4, 8, 24</td>
</tr>
<tr>
<td>31</td>
<td>4</td>
<td>biCompression</td>
<td>specifies the type of compression-it is usually set to zero (meaning no compression)</td>
</tr>
<tr>
<td>35</td>
<td>4</td>
<td>biSizeImage</td>
<td>contains the size of the image data in bytes-it can also be set to zero (no compression)</td>
</tr>
<tr>
<td>39</td>
<td>4</td>
<td>biXPelsPerMeter</td>
<td>contains application-specified values for the desirable dimensions of the bitmap (usually set to zero)</td>
</tr>
<tr>
<td>43</td>
<td>4</td>
<td>biYPelsPerMeter</td>
<td>contains application-specified values for the desirable dimensions of the bitmap (usually set to zero)</td>
</tr>
<tr>
<td>47</td>
<td>4</td>
<td>biClrUsed</td>
<td>specifies the number of colors used in the bitmap</td>
</tr>
<tr>
<td>51</td>
<td>4</td>
<td>biClrImportant</td>
<td>specifies that the first x colors of the color table are important to the bitmap-if set to zero, all colors are important</td>
</tr>
</tbody>
</table>
Appendix B

Compilation of the Steganographic Tool:

In order to compile the steganographic tool, you will have to type `make steg`. Additionally, you will have to type `make test` in order to test the compiled steganographic tool. It is essential that Cygwin\(^1\) and GCC\(^2\) are installed on your computer and that the file `Makefile` is in the same directory as the source (*.c) and header (*.h) files. Particularly, the following files are required for the program to be compiled:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>main.c</code></td>
<td>This is the actually the main program</td>
</tr>
<tr>
<td><code>embed.c</code></td>
<td>Functions used to <em>Embed</em> Secret Message into Cover-Image</td>
</tr>
<tr>
<td><code>embed.h</code></td>
<td></td>
</tr>
<tr>
<td><code>extract.c</code></td>
<td>Functions used to <em>Extract</em> Secret Message from Stego-Image</td>
</tr>
<tr>
<td><code>extract.h</code></td>
<td></td>
</tr>
<tr>
<td><code>compdec.c</code></td>
<td>Functions used to <em>Compress</em> or <em>Decompress</em> Secret Message. They make use of the zlib 1.1.4 library</td>
</tr>
<tr>
<td><code>compdec.h</code></td>
<td></td>
</tr>
<tr>
<td><code>encdecr.c</code></td>
<td>Functions used to <em>Encrypt</em> or <em>Decrypt</em> Secret Message. They make use of the 3DES and SHA algorithms</td>
</tr>
<tr>
<td><code>encdecr.h</code></td>
<td></td>
</tr>
<tr>
<td><code>general.c</code></td>
<td>General purpose header file. Includes an error handling function</td>
</tr>
<tr>
<td><code>general.h</code></td>
<td></td>
</tr>
<tr>
<td><code>libz.a</code></td>
<td>Zlib version 1.1.4</td>
</tr>
<tr>
<td><code>zlib.h</code></td>
<td>Library and header file</td>
</tr>
<tr>
<td><code>lbdes.a</code></td>
<td>The implementation of the 3DES encryption algorithm (Young 1997)</td>
</tr>
<tr>
<td><code>des.h</code></td>
<td></td>
</tr>
<tr>
<td><code>sha.c</code></td>
<td>The implementation of the Secure Hash Algorithm (Gillogly 1994)</td>
</tr>
<tr>
<td><code>sha.h</code></td>
<td></td>
</tr>
<tr>
<td><code>Makefile</code></td>
<td>UNIX makefile for the steganographic tool</td>
</tr>
</tbody>
</table>

Additionally, the following files are required in order to test the compiled program:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>coverimg.bmp</code></td>
<td>True colour (24-bit) cover-image</td>
</tr>
<tr>
<td><code>ieee.pdf</code></td>
<td>Secret message</td>
</tr>
</tbody>
</table>

Furthermore, you may delete all the object (*.o) files, after having compiled the steganographic tool, by typing `make clean`.

---

\(^1\) Further information on Cygwin library and tools is available in Appendix A.

\(^2\) Further information on GCC (GNU Compiler Collection) is available in Appendix A.
**Embedding a Secret Message:**

The syntax used to embed a secret message into a cover-image is the following one:

```
steg [cover-image] [secret-message] [stego-image]
```

for example:

```
steg coverimg.bmp ieee.pdf stegoimg.bmp
```

**Extracting a Secret Message:**

The syntax used to extract an embedded message from a stego-image is the following one:

```
steg [stego-image] [output-file]
```

for example:

```
steg stegoimg.bmp test.pdf
```

**Help Using the Steganographic Tool:**

You may ask for help on using the steganographic tool at any time, by simply typing `steg`.