ÇANKAYA UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

MASTER THESIS

DETECTING LOCATION OF HIDDEN MESSAGES ON DIGITAL IMAGES USING RS STEGANALYSIS METHOD

EFE ÇİFTCİ

JULY 2013

Title of the Thesis : **Detecting Location of Hidden Messages on Digital Images Using RS Steganalysis Method**

Submitted by **Efe ÇİFTCİ**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University

[illegible]

Prof. Dr. Taner ALTUNOK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Assist. Prof. Dr. Murat SARAN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]

Assist. Prof. Dr. Reza HASSANPOUR
Supervisor

Examination Date : 18.07.2013

Examining Committee Members

Assist. Prof. Dr. Reza HASSANPOUR	(Çankaya Univ.)	[illegible]
Assist. Prof. Dr. Nurdan SARAN	(Çankaya Univ.)	[illegible]
Prof. Dr. Mehmet R. TOLUN	(TED Univ.)	[illegible]

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name	: Efe ÇİFTCİ
Signature	: [illegible]
Date	: 18.07.2013

ABSTRACT

DETECTING LOCATION OF HIDDEN MESSAGES
ON DIGITAL IMAGES USING RS STEGANALYSIS METHOD

ÇİFTCİ, Efe
M.Sc., Department of Computer Engineering
Supervisor : Assist. Prof. Dr. Reza HASSANPOUR

July 2013, 41 pages

Steganography is a branch of methods that deal with hiding information in cover media. The result of the hiding process should resist detection by any means. To detect the hidden information, several analysis methods have been developed (named as steganalysis methods). Steganalysis of Regular and Singular Groups (RS) is a method which aims to detect hidden information on digital images. This method is helpful for estimating length of the hidden message but it does not find over which section of the image the hidden message is. The aim of this thesis is to locate in which part(s) of the image the message is hidden by utilizing this steganalysis method and quadtree data structure.

Keywords: Steganography, Steganalysis, Quadtree

ÖZ

RS STEGANALİZ YÖNTEMİ KULLANARAK SAYISAL GÖRÜNTÜLERDEKİ GİZLİ İLETİLERİN KONUMUNUN TESPİT EDİLMESİ

ÇİFTCİ, Efe
Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi : Yrd. Doç. Dr. Reza HASSANPOUR

Temmuz 2013, 41 sayfa

Steganografi, taşıyıcı ortamlar üzerine bilgi gizleme yöntemlerine verilen isimdir. Gizleme işleminin sonucu herhangi bir tespit yöntemine karşı dayanıklı olmalıdır. Gizli bilgiyi tespit edebilmek için steganaliz olarak adlandırılan çeşitli analiz yöntemleri geliştirilmiştir. Bir steganaliz yöntemi olan RS analizi, sayısal görüntüler üzerinde gizli bilgi tespit etmek için geliştirilmiş bir yöntemdir. Bu analiz yöntemi gizli iletinin uzunluğunu hesaplamak için faydalı bir yöntem olmasına rağmen gizli iletinin, görüntü dosyasının hangi kısmı üzerinde bulunduğunu tespit edememektedir. Bu tezin hedefi bu steganaliz yöntemini ve quadtree veri yapısını kullanarak gizli iletinin görüntünün hangi kısım(lar)ında bulunduğunun tespit edilmesidir.

Anahtar Kelimeler: Steganografi, Steganaliz, Quadtree

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM....................iii
ABSTRACT....................iv
ÖZ....................v
TABLE OF CONTENTS....................vi
LIST OF TABLES....................viii
LIST OF FIGURES....................ix
CHAPTERS:
1 INTRODUCTION....................1
1.1 Outline of Thesis....................2
2 STEGANOGRAPHY....................3
2.1 The Prisoners' Problem....................3
2.2 Watermarking....................4
2.3 Encryption....................5
2.4 History and Physical Examples of Steganography....................8
2.5 Digital Steganography....................10
2.5.1 Digital steganography by carrier format....................12
2.5.1.1 Digital audio....................12
2.5.1.2 Digital image....................13
2.5.1.3 Digital video....................15
2.5.1.4 Text and other formats....................16

2.5.2 Techniques for steganography on digital images................................17
2.5.2.1 Techniques on spatial domain...17
2.5.2.2 Techniques on frequency domain..23
3 STEGANALYSIS OF DIGITAL MEDIA...25
3.1 Digital Images...26
3.1.1 Steganalysis methods on spatial domain..26
3.1.1.1 Sample pair analysis...26
3.1.1.2 Chi-Square analysis..27
3.1.1.3 Steganalysis of regular and singular groups................................27
3.1.2 Steganalysis methods on frequency domain.......................................30
3.2 Other Digital Media...30
4 RS STEGANALYSIS WITH QUADTREE DATA STRUCTURE....................32
4.1 Quadtrees On Digital Images...33
4.2 RS Steganalysis With Quadtrees...34
5 EXPERIMENTS AND RESULTS...36
6 CONCLUSION...41
6.1 Future Work...41
REFERENCES..42
APPENDICES:
A. RS STEGANALYSIS CODE WITH QUADTREE DATA STRUCTURE......45
B. RANDOM MESSAGE GENERATION CODE..49
C. CURRICULUM VITAE...50

LIST OF TABLES

TABLES

Table 2.1 Comparison of Steganography, Encryption and Watermarking....................7

Table 5.1 Image Groups...37

Table 5.2 Test 2 Results...39

LIST OF FIGURES

FIGURES

Figure 2.1 The Prisoners' Problem....................4

Figure 2.2 Steganography by Letter Shifting....................10

Figure 2.3 Hiding Messages After EOF....................11

Figure 2.4 RGB Composition....................14

Figure 2.5 Before and After States of 8 Pixels....................18

Figure 2.6 Effects of Flipping Bits....................19

Figure 2.7 Comparison of a Digital Image and Its GIF Version....................21

Figure 2.8 Histogram Comparison for LSB Method....................22

Figure 2.9 Digital Image Embedded in Another Digital Image....................22

Figure 3.1 RS Diagram of an Image....................29

Figure 4.1 Quadtree....................32

Figure 4.2 An Image with Quadtree Segmentation....................33

Figure 5.1 Highest and Lowest Initial Results....................37

Figure 5.2 ucid01049.bmp, Zoomed-in....................37

Figure 5.3 Embedding Process....................38

CHAPTER 1

INTRODUCTION

Since the earliest days of known human history, communication has been an essential aspect of daily life. Voice being the most basic form, human beings have utilized different methods for communication during ages. With each age, communication methods of humanity have evolved from simple drawings on cave walls to carved wax tablets and handwritten letters to digital communication.

There are times when people would not want their communication to be interpreted by others. In such cases, the communication should be protected. This protection can be achieved by many methods. One method of protection is hiding messages in unsuspicious carriers. All techniques involving hiding messages in a carrier medium are called steganography.

With the advancements on digital technologies starting at 20th century and computers getting more popular by each passing day, steganography has gained new fields of application. Today; every information on computers are stored in different formats according to the data they represent. The most basic form of user data on computers is plain text documents, which consist of characters that when read together, they form words and sentences. Digital images are another type of user data; where the bytes that are arranged as two-dimensional matrices represent colors and shapes. Digital audio consist of bytes carrying sound information, which are heard as various forms of sounds such as music or speech. Digital videos are digital images shown one after another and usually accompanied with digital audio.

These are just a few examples; there are many other representations of digital data which are used and interpreted in many different ways. With all these different formats, new methods of steganography have been discovered.

Along with these steganography methods; their counterparts, steganalysis methods have also been developed. Steganalysis is the common name given to methods that are useful for detection of hidden messages in carrier media. Rather than the sender and receiver, these steganalysis methods are employed by other people who are not supposed to be the receivers but who has intercepted and got suspicious of the mentioned media.

The aim of this study is accurately detecting the location of hidden messages on stego carriers by utilising an existing steganalysis method onto different segments of the carrier. Choice of medium is digital images since the mentioned method works on digital images.

1.1 Outline of Thesis

This thesis has been divided into 6 chapters, each focusing on one major topic. **The second chapter** of this thesis contains the history of steganography, explains how messages can be hidden into various types of carrier media. **Third chapter** explains the concept of steganalysis, gives examples on various steganalysis methods and focuses on RS steganalysis method. **Fourth chapter** gives definition about quadtree data structure and explains how it can be combined with RS steganalysis method to further enhance the analysis process. **Fifth chapter** demonstrates the experiments conducted for validation of the method explained in Chapter 4 and presents the results of each test. **Sixth and final chapter** summarizes the work and experiments done, mentions future works and concludes this thesis.

CHAPTER 2

STEGANOGRAPHY

Steganography is the science of hidden communication. The word steganography has been derived from Greek language; *steganos* means *protected* and *graphei* means *writing* [1]. This hidden communication is achieved by embedding the information in unsuspicious carrier media. Any method that hides information in carrier media is named a "steganographic method". Steganographic methods are used when an information should be transferred from a source to a destination but the discovery or visibility of the information is not desired, thus it should remain hidden from everyone else who knows the existance of the information, namely the source and destination.

Steganographic methods should do minimum amount of detectable change to the carrier media because steganography may fail if the carrier media gets altered in a much detectable way and becomes too obvious that it contains extra information.

2.1 The Prisoners' Problem

A famous example for defining steganography is the Prisoners' Problem, defined by Simmons [2]. This problem features three people; of which two are prisoners and one is the warden of the prison. The prisoners have been put in two seperate cells of a prison. In order to devise a plan to escape the prison, the prisoners should communicate but they can't communicate directly, so they should ask the warden to deliver their messages. They should also communicate discreetly; otherwise the warden may destroy the message and prevent their plans if he discovers the prisoners are conspiring. This discreet communication requires steganography. Figure 2.1 demonstrates the process defined by this problem.

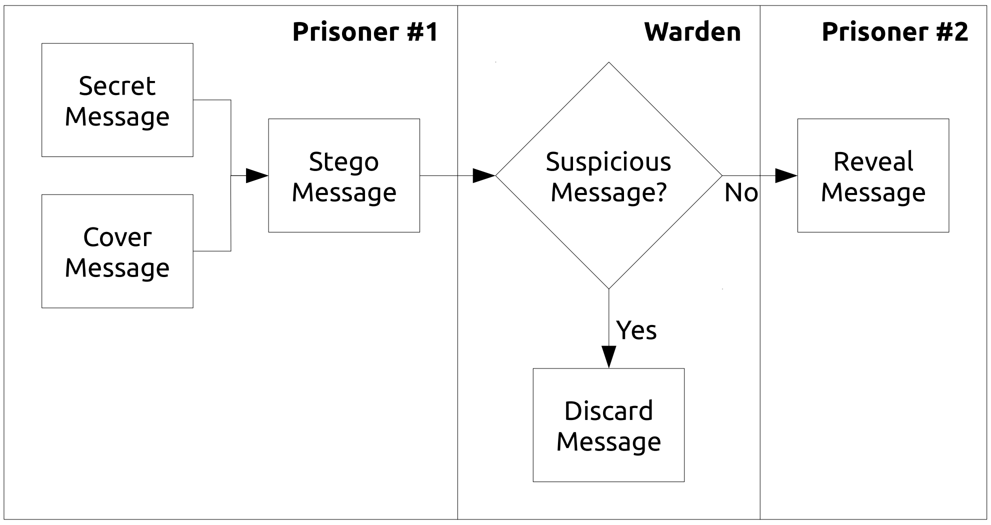


Figure 2.1 The Prisoners' Problem

In order to successfully inform Prisoner #2 of the escape plan without the warden realising it, Prisoner #1 should encode and hide the plans in an innocent cover message so the warden won't get suspicious of the message. When Prisoner #2 receives the message, he can decode the contents and extract the secret message from the innocent cover message.

2.2 Watermarking

A watermark is an addition applied on media to discourage the media from unauthorized uses and to prevent fraud. When a property owned by someone should be made publicly available, the owner would need the property to be protected in a way that the owner can be identified and the property is unique. It is usually a visible text or logo but depending on the medium, different types of watermarks can be used as well.

Some examples for watermarked property are;

- Banknotes: Several hard-to-replicate watermarks are applied onto banknotes for security reasons and to identify which banknotes are authentic and which are not.
- Media such as images or videos owned by a certain individual or organization: These usually have a logo or text embedded on them to identify the owner, so when these media are used elsewhere, the origin of the media can be traced back to the owner.
- Digital watermarking methods such as the ones proposed in [3] and [4] provide robust watermarks for digital media such as images and videos.

Although watermarking and steganography are both used for embedding information into cover media, the goals and results are different. For instance, invisibility of the embedded data is a requirement for steganography but it is not a necessity for watermarking. Watermarking instead focuses on robustness, that the embedded information should resist and be hard to remove from cover.

2.3 Encryption

Encryption is the transformation of readable messages into unreadable ciphertext. This transformation is required when contents of a message should be kept safe from being interpreted by unwanted third parties. Reverse of this transformation process which produces readable messages again is named decryption. Encryption and decryption should be accomplished only by authorized people.

On modern computers and similar devices such as cell phones, encryption is necessary for several tasks such as;

- **Local data storage:** Data of users that are stored on local hard drives can be encrypted to keep them safe from other users of the same computer. While storage encryption can be employed on computers for all purposes, this is especially useful for large data servers which are accessed by hundreds and thousands of users routinely so that no one can access the files of other users without their acknowledgement.
- **Network security:** Information transferred across computers or similar devices are usually encrypted for safekeeping from possible man in the middle attacks. Encryption and decryption occurs at the source and destination devices, respectively. While there are many more, HTTPS (Hypertext Transfer Protocol Secure) and SSH (Secure Shell) are two common examples of network protocols that provide such security measures.

It should be noted that in steganography, the message that is hidden in cover media can be encrypted first in order to make interpreting this message a lot challenging, in case the cover medium has been analysed by attackers and steganography has failed. In such cases, both the sender and the receiver must know how to hide / reveal the message and how to encrypt / decrypt the hidden message respectively.

Differences between steganography, watermarking and encryption has been listed in [5] according to several criteria; these differences are shown in Table 2.1.

Table 2.1 Comparison of Steganography, Encryption and Watermarking [5]

Criteria	**Steganography**	**Watermarking**	**Encryption**
Carrier	Any digital media	Mostly image / audio files	Usually text based, with some extensions to image files
Secret Data	Payload	Watermark	Plain text
Key	Optional		Necessary
Input Files	At least two unless in self-embedding		One
Detection	Blind	Usually informative (i.e., original cover or watermark is needed for recovery)	Blind
Authentication	Full retrieval of data	Usually achieved by cross correlation	Full retrieval of data
Objective	Secret communication	Copyright preserving	Data protection
Result	Stego-file	Watermarked-file	Cipher-text
Concern	Delectability / capacity	Robustness	Robustness
Type of Attacks	Steganalysis	Image processing	Cryptanalysis
Visibility	Never	Sometimes	Always
Fails When	It is detected	It is removed / replaced	Deciphered
Relation to Cover	Not necessarily related to the cover. The message is more important than the cover	Usually becomes an attribute of the cover image. The cover is more important than the message	N/A
Flexibility	Free to choose any suitable cover	Cover choice is restricted	N/A
History	Very ancient except its digital version	Modern era	Modern era

2.4 History and Physical Examples of Steganography

Depending on the age of humanity, there have been various methods of steganography. The earliest known methods of steganography can be traced back to 450 - 400 BC. First examples of steganography covers wax tablets; which was achieved by carving messages on wooden part of tablets and then covering the carved side of the wooden tablet with wax. After this operation, any regular message would be carved on the waxy surface and the tablet would be sent to its destination. Anyone who intercepted these tablets instead of the destination would only notice the text on the top layer (wax) but detection of text on bottom layer (wood) would be difficult unless the wax layer was removed.

In known history, another historical steganographic method was first used in ancient China. This method required usage of paper masks by both the sender and the receiver. These papers had holes at random locations, so that when sender placed his mask over a paper, he could write the secret message into the holes. Then he would remove the mask and write a cover message to the rest of the paper and send it. Receiver could recover the hidden message by putting his own mask paper onto the letter.

One another historical steganographic method was achieved by tattooing a message on the shaved head of a slave. When the hair grows, it covers the tattooed message so it can not be seen. When the slave reached his destination, his head was again shaved to reveal the message. Similar steganographic methods that involve human body have been used recently during World War II as well; by writing the message on the backs of messengers using invisible ink.

Writing messages using invisible ink on different carriers also produces acceptable results for steganography. For example, if a message was written on a paper using invisible ink, that text would not be seen by bare eyes. Then, something totally unrelated and unsuspicious can be written on the paper using ordinary ink. As a result of this process, anyone viewing the paper would only see the information written using regular ink and would not notice the information written with invisible ink. To reveal the invisible message, special chemical solutions should be applied on the paper.

Written text can alone be used for steganography as well. One basic form of steganography on plain text is capitalizing certain non-consecutive letters of a given text so when these capitalized letters are read together, they will form a meaningful message. Although this method is simple to apply, it is also very simple to be detected because randomly capitalized letters can become very suspicious in a standard paragraph where capitalized letters should appear only at abbreviations and at the beginning of sentences and important words.

Another steganographic method that can be applied on printed media that is similar to the previously explained method (but harder to detect) can be accomplished by shifting the positions of certains letters in a text by very small amounts. This shifting operation on letters is very hard to detect by bare eyes at first sight, but with proper information, these letters can be detected and rearranged together to form a meaningful text. Figure 2.2 demonstrates this method.

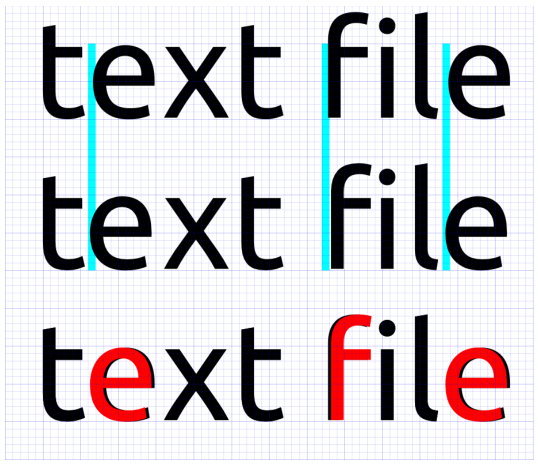


Figure 2.2 Steganography by Letter Shifting

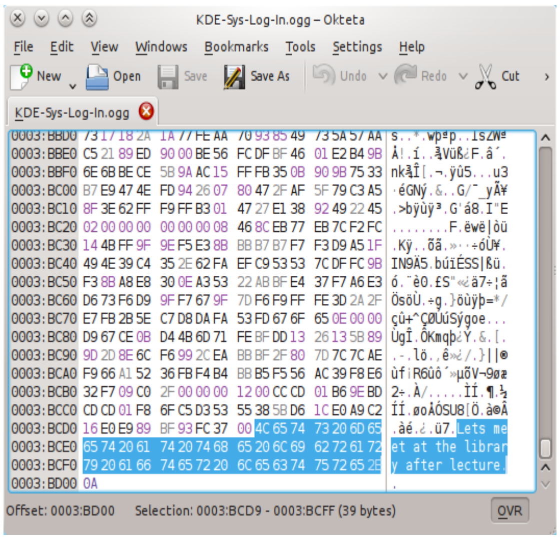
The first line in the figure is the original line. In the second line, three letters were tilted in small amounts to either left or right and third line overlaps both lines to make the differences visible. Cyan lines between the first and second lines have been placed there for the purpose of easier detection of this operation, showing how much these three letters have moved.

This steganographic method can be applied in different ways; such as shifting words themselves, instead of single letters. This "word shifting" steganographic method can be used in different ways. One way is embedding the message by shifting corresponding words in the cover text. These words, when they are read together, will form the hidden message. One another word shifting technique will be mentioned under Section 2.5.1.4.

2.5 Digital Steganography

Computers and similar electronical devices such as embedded systems or mobile devices handle and store user data with different methods according to the data format but basically, all information stored on these devices are represented by bits and bytes. Thus, these devices provide a vast array of steganography methods.

One basic steganography method on digital media can be achieved by hiding the message in unused bytes of the cover media. These unused bytes can be padding bytes; which are used for defining a fixed length of bytes such as file headers. Or the message can be hidden after the end of file as shown in Figure 2.3. Since the program that is used for opening these files know which bytes to interpret and will stop at EOF marker, the bytes that are used for steganography are ignored by these programs.


KDE-Sys-Log-In.ogg – Okteta
File Edit View Windows Bookmarks Tools Settings Help
New Open Save Save As Undo Redo Cut
KDE-Sys-Log-In.ogg
OVR

```
0003:BBD0 73 17 18 2A  1A 77 FE AA  70 93 85 49  73 5A 57 AA  s..*.wpªp..IsZWª
0003:BBE0 C5 21 89 ED  90 00 BE 56  FC DF BF 46  01 E2 B4 9B  Å!.í..¾Vüß¿F.â´.
0003:BBF0 6E 6B BE CE  5B 9A AC 15  FF FB 35 0B  90 9B 75 33  nk¾Î[.¬.ÿû5...u3
0003:BC00 B7 E9 47 4E  FD 94 26 07  80 47 2F AF  5F 79 C3 A5  ·éGNý.&..G/¯_yÃ¥
0003:BC10 8F 3E 62 FF  F9 FF B3 01  47 27 E1 38  92 49 22 45  .>bÿùÿ³.G'á8.I"E
0003:BC20 02 00 00 00  00 00 00 08  46 8C EB 77  EB 7C F2 FC  ........F.ëwë|òü
0003:BC30 14 4B FF 9F  9E F5 E3 8B  BB B7 B7 F7  F3 D9 A5 1F  .Kÿ..õã.»··÷óÙ¥.
0003:BC40 49 4E 39 C4  35 2E 62 FA  EF C9 53 53  7C DF FC 9B  IN9Ä5.búïÉSS|ßü.
0003:BC50 F3 8B A8 E8  30 0E A3 53  22 AB BF E4  37 F7 A6 E3  ó.¨è0.£S"«¿ä7÷¦ã
0003:BC60 D6 73 F6 D9  9F F7 67 9F  7D F6 F9 FF  FE 3D 2A 2F  ÖsöÙ.÷g.}öùÿþ=*/
0003:BC70 E7 FB 2B 5E  C7 D8 DA FA  53 FD 67 6F  65 0E 00 00  çû+^ÇØÚúSýgoe...
0003:BC80 D9 67 CE 0B  D4 4B 6D 71  FE BF DD 13  26 13 5B 89  ÙgÎ.ÔKmqþ¿Ý.&.[.
0003:BC90 9D 2D 8E 6C  F6 99 2C EA  BB BF 2F 80  7D 7C 7C AE  .-.lö.,ê»¿/.}||®
0003:BCA0 F9 66 A1 52  36 FB F4 B4  BB B5 F5 56  AC 39 F8 E6  ùf¡R6ûô´»µõV¬9øæ
0003:BCB0 32 F7 09 C0  2F 00 00 00  12 00 CC CD  01 B6 9E BD  2÷.À/.....ÌÍ.¶.½
0003:BCC0 CD CD 01 F8  6F C5 D3 53  55 38 5B D6  1C E0 A9 C2  ÍÍ.øoÅÓSU8[Ö.à©Â
0003:BCD0 16 E0 E9 89  BF 93 FC 37  00 4C 65 74  73 20 6D 65  .àé.¿.ü7.Lets me
0003:BCE0 65 74 20 61  74 20 74 68  65 20 6C 69  62 72 61 72  et at the librar
0003:BCF0 79 20 61 66  74 65 72 20  6C 65 63 74  75 72 65 2E  y after lecture.
0003:BD00 0A                                                  .
```

Offset: 0003:BD00   Selection: 0003:BCD9 - 0003:BCFF (39 bytes)

Figure 2.3 Hiding Messages After EOF

As a result, the user notices no difference on how the media is served but viewing the bytes using text editors reveals such messages. While this method provides possibility to hide unlimited lengths of messages, hiding long messages should be avoided as it may produce suspiciously huge files.

2.5.1 Digital steganography by carrier format

Different formats of data require different arrangements of bits and bytes. For example; a plain text file, which consists of mostly human readable information, will not be interpreted as a digital image by computers because bytes stored by the text file would not be organized in the way digital images require them to. Because of these differences, various different steganography methods that are dependent to the specifications of each carrier format have been implemented.

2.5.1.1 Digital audio

Digital audio files store sound information that has been recorded from analog signals and converted to digital representation of these signals. These sound information can either be stored compressed or uncompressed. Compressed audio files usually omit less audible sounds and they have lesser amount of redundant information.

MP3 (MPEG-1 / MPEG-2 Audio Layer III) being the most widely known and popular, there are many both proprietary and free formats such as OGG Vorbis, FLAC, Windows Media Audio and RealAudio that can be used for storing digital audio today.

Differences about how these formats store sound information has enabled researchers to implement steganography on these formats. In [6], it has been proposed that data can be hidden in integer wavelet domain into cofficients of a given cover audio file. [7] and [8] proposes different steganography methods for MP3 files; one by exploiting the rule of window switching during encoding and the other via Huffman tables, respectively.

2.5.1.2 Digital image

Steganography methods that target digital images focus on hiding information in cover images. As a result of this hiding process, a "stego image" is created. Some of the steganography methods are based on modifications on spatial domain and some of them are based on frequency domain. There are also methods such as the one proposed in [9] which utilizes both the spatial and frequency domains of images to achieve steganography. Some other possibilities such as hiding message in unused bytes which has been explained before in Section 2.5 are also present. Methods that focus on spatial and frequency domains will be studied further in Section 2.5.2.1 and Section 2.5.2.2 respectively.

In computer graphics, the smallest element of a digital image is named a "pixel". Each and every single pixel holds a color value. Arranging multiple pixels in rows and columns create a visual display perceived by the human eye as a two-dimensional image. Number of columns and rows of pixel determine the resolution of digital images. Total amount of pixels in digital images is calculated as the multiplication of column and row counts. For example, a 640x480 image has 640 columns and 480 rows of pixels and the total amount of pixels in that image is 307.200.

Color of a pixel is determined by the numerical value of the bits and bytes it occupies. Depending on the image type, pixels may use different lengths of bits. This value is named as bits per pixel (bpp). While there have been many different implementations of color depths during the evolution of computer graphics and display devices, some of these implementations are being used very regularly today.

For example, an image is commonly considered as grayscale today if each pixel of the image uses 8 bits. In such images, every single pixel can hold up to only 256 values (0 – 255) of brightness without any color information; where 0 is black, 255 is white and values between 0 and 255 are shades of gray from black to white.

Another popular implementation is color images that use 24 bits for pixels. Such images define three values of color information for each pixel: red, green and blue; again divided in groups of 8 bits. Such digital images of 24 bit color depth are also referred as "true color" images. Similar to grayscale images, these groups of 8 bits still hold values between 0 and 255 but since now we have three different groups of 8 bits (each for red, green and blue channels) such pixels can now hold up to 16.777.216 different values of color (256 * 256 * 256) due to the composition of the 3 different grayscale channels. This composition operation of three channels is demonstrated in Figure 2.4.

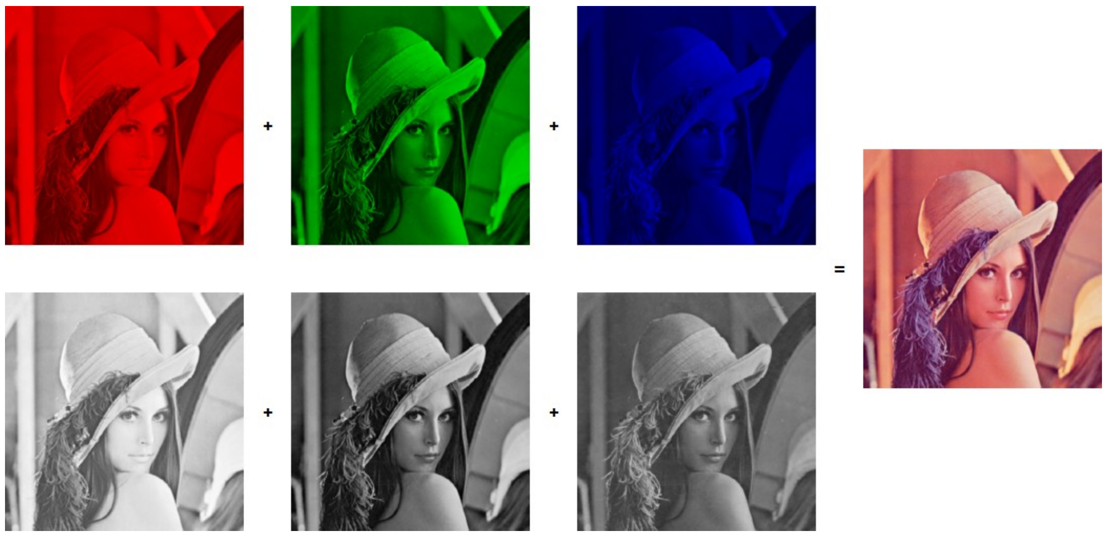


Figure 2.4 RGB Composition

For example; a pixel with 0 red, 0 green and 0 blue is black because values of all three color channels are set to 0, another pixel with 255 red, 0 green and 0 blue is bright red because it has the maximum value of red and no values for green and blue, and an another pixel with 255 red, 255 green and 255 blue is white because it has the maximum value of all three color channels.

An another implementation uses 32 bits per pixel for color images. This kind of color depth is named as RGBA or sometimes ARGB; which stands for red, green, blue and alpha channels. When compared to RGB color depth, RGBA's structure is similar but the extra 8 bits for alpha channel holds opacity value for the pixel. In RGB images, only the color value of pixels can be defined but the pixels themselves should stay 100% opaque. In RGBA images, the extra 8 bits used by opacity channel lets the visibility of the pixels range from 100% transparent (if the opacity value is 0) to 100% opaque (if the opacity value is 255), while all other values between 0 and 255 provide different levels of translucency for the pixel.

2.5.1.3 Digital video

Digital videos are actually multimedia files that contain multiple data streams such as audio and video tracks. A digital video file must at least contain one video track. These video tracks are sequences of images which are displayed one after another very rapidly so that they are perceived by human visual system as a moving image. Video files can grow very large as the length of the video increases. For example, a 30 minutes long video that displays 25 frames per second stores 45.000 still images. To overcome this problem, various compression methods such as MPEG or H.264 have been implemented.

Depending on their format specifications, it is possible to hide information in both compressed and uncompressed video formats. [10], [11], [12] and [13] are a few examples of proposed methods for video steganography, each focusing on different properties of videos for hiding.

2.5.1.4 Text and other formats

Word shifting method, which has been explained previously under Section 2.4, can also be accomplished through assistance of bits and bytes as well. In this method, the hidden message should first be converted from ascii to binary form. Then according to the binary form, positions of words in the cover text can be altered so that words in their original position will denote bit "0" and shifted words will denote bit "1". To recover the hidden message, binary text should first be extracted from the cover text according to the positions of the words, and then the binary message should be translated into ascii form.

It has been demonstrated in [14] how TeX source files can be used to hide information in generated PDF document. Their method proposes that words in a line should first be grouped such that;

- Each pair of consecutive words carries one bit of secret data,
- Two words before and after a full stop are treated as a single word,
- Last word of a line and first word of next line are treated as a single word.

After this grouping, spaces between words of groups are modified according to whether they will hold bit "0" or bit "1". If 0 is going to be embedded, then the space of current group is left unmodified. If 1 is going to be embedded, then the space of the group will be widened or narrowed in turns to prevent to keep space widths as minimum as possible and increase efficiency of the method.

2.5.2 Techniques for steganography on digital images

Steganographic methods that can be applied on digital images are commonly classified in two groups;

- Steganography in spatial domain,
- Steganography in frequency domain.

2.5.2.1 Techniques on spatial domain

Techniques on spatial domain are mostly focused on altering bits of pixels to carry messages. This alteration must happen in small amounts in order to make the difference caused by steganography indetectable by human visual system. In other words; range of this modification should be kept as narrow as possible, as wider ranges may result in undesirable results such as turning color of a pixel from light gray to dark gray.

LSB steganography (LSB replacement) method focuses on the least significant bits of channels of pixels. The eighth bit of each channel affects the value of the pixel by 1. Likewise, the seventh bit modifies the color by 2 and the sixth bit modifies the color by 4, etc. Because of this fact, any changes on the eighth bit is very insignificant and hard to detect by human visual system. On images with lots of color information (e.g. photos taken by digital cameras), it is almost impossible to detect by human visual system if the LSB's of pixels have been changed or not.

Modifying the LSB's of several pixels enables us to hide information in these bits. For example, binary representation of the character 'e' is "0110 0101". To carry this character, LSB's of eight pixels should be set to ones and zeros accordingly; so that each pixel will carry one bit of this character. Figure 2.5 shows the difference between the original and modified values of eight consecutive pixels of a grayscale digital image. As it can be seen from the figure, original and modified colors of the pixels are almost identical and differences caused by modifying least significant bits are almost impossible to detect with human visual system.

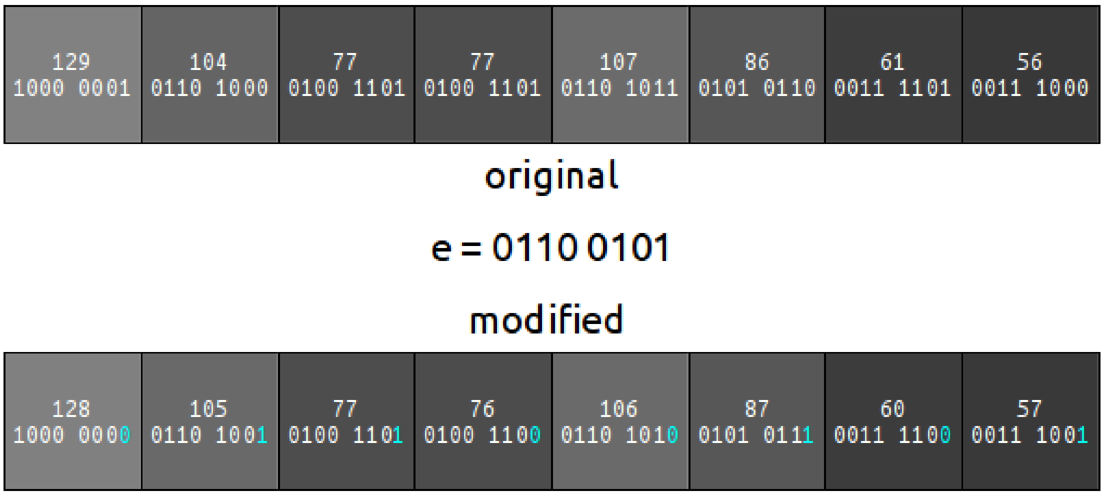


Figure 2.5 Before and After States of 8 Pixels

It is also possible to alter more than one bits per pixel in Least Significant Bit method to increase embedding capacity; but as the amount of bits used for steganography per pixel increases, risk of detection by human visual system also increases. Figure 2.6 demonstrates how the state of each bit affects the value of a randomly chosen color (Red: 139, Green: 0, Blue: 0). From this figure, it can be derived that the visual differences caused by the change of color increase rapidly as the changed bit shifts from the least significant bit to the most significant bit.

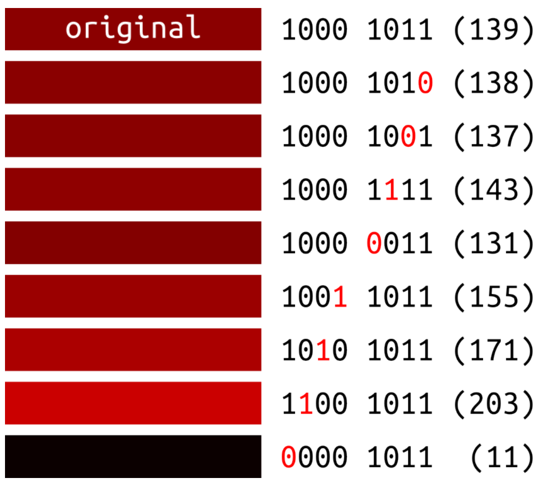


Figure 2.6 Effects of Flipping Bits

Least Significant Bit steganography can be achieved in several ways. One may embed the information in pixels in sequential order. To reveal the message in such images, it is adequate to read LSB's of the image in the same sequential order. To strenghten the method, only a subset of the pixels may be used for steganography. For example, the message can be embedded into every 10th pixel. This way, the message can be restored by extracting LSB's from only the 10th, 20th, 30th, etc. pixels. To further strengthen the method, the order of the pixels used for embedding can be scrambled in a predefined pseudo-random order (that is shared between sender and receiver) as well. Such documents produces meaningless messages if anyone tries to interprets LSB's in sequential order. Thus, it is necessary to interpret LSB's according to the previously defined order.

Embedding message into edges available in images is an another method for LSB steganography. In an image, edges are defined as locations in which the brightness of neighbouring pixels change sharply. This instant change of brightness can be used as an advantage as embedding in these locations will raise less suspicion. Before embedding into edges, edge detection methods such as Sobel, Prewitts, Laplacian, etc. should be used to identify these locations first.

A different approach for LSB steganography is named as **LSB matching**. In this method, LSB's of pixels are modified by randomly adding or subtracting 1 from the pixel if it does not match the bit of the message. Methods proposed in [15], [16], [17] and [18] provide improvements to LSB matching method.

Maximum length of hidden message a digital image can carry depends on;

- Resolution of image (width * height),
- Amount of color channels,
- Amount of bits used per pixel.

As an example; a 512x512 grayscale image can hold messages as long as 32.768 bytes if only 1 bit is used for steganography, or a 384x256 RGB image can hold messages as long as 73.728 bytes if 2 bits are used.

The format of digital images as a carrier media is very important for Least Significant Bit steganographic method. Digital image formats such as JPEG or GIF process the image to decrease the file size. These operations produce lossy results, which means pixels irreversibly lose their original values after the the compression has been applied onto the image. Figure 2.8 demonstrates the loss caused by GIF format by comparing the original image and its GIF version. GIF format commonly supports images that use up to 256 colors, which requires alteration of pixels according to the 256-color palette. In the figure, every pixel is represented by unique colors in the original image on the left; while its GIF counterpart on the right must represent much larger blocks of pixels with the same color because of the lack of available colors.

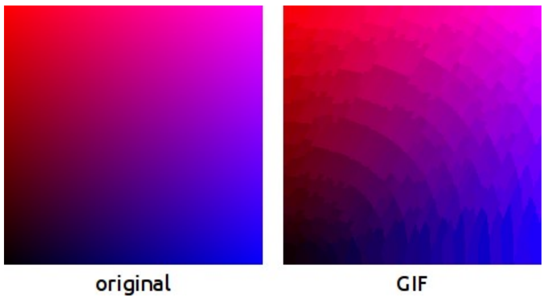


Figure 2.7 Comparison of a Digital Image and Its GIF Version

Because of these and similar facts, a stego image which has been converted to a lossy format can no longer be used for reading the message that was hidden in spatial domain by methods such as Least Significant Bit method; as the bits carrying the information have been modified after embedding. To overcome this problem, digital image formats that provide no compression or lossless compression should be preferred as carrier media for Least Significant Bit method instead. Portable Network Graphics (PNG) or Bitmap image format (BMP) are two examples for this requirement; PNG format provides lossless compression, while BMP files can be saved without any compression at all. Digital image files saved in these formats or any other format that keeps pixel values intact can safely be used as carrier media for Least Significant Bit steganographic method.

In addition to image format, contents of the chosen cover image is also important. Since this method modifies pixels; groups of same colored pixels may turn into noisy regions, which can be detected much easier than regions that are made up of irregular colors. These changes in the image can be observed via histograms. Figure 2.8 compares the histograms of original lenna.bmp and its 100% embedded version according to LSB method. It can be deduced that the embedding operation has resulted a much more noisy image.

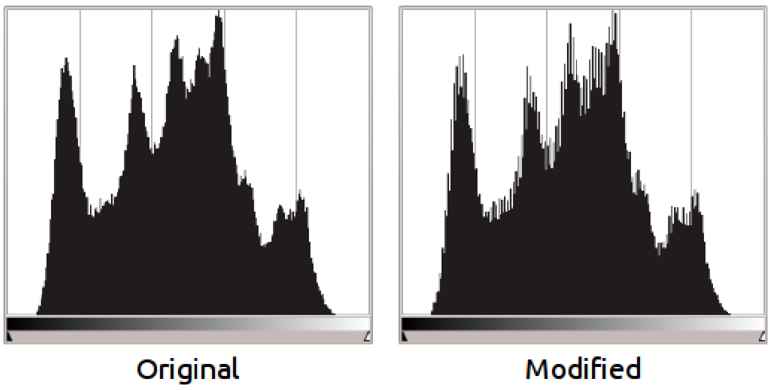


Figure 2.8 Histogram Comparison for LSB Method

Least Significant Bit method does not necessarily require the format of the hidden message to be plain text only. Figure 2.9 [19] is a demonstration of a digital image hidden in another other digital image.

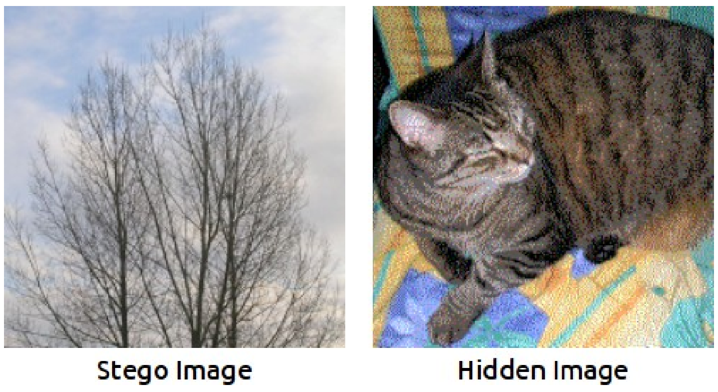


Figure 2.9 Digital Image Embedded In Another Digital Image

Here, the RGB image of cat has first been reduced to 64 colors; which means 4 colors for each color channel. Then these colors have been compressed so that they can be represented using only two bits as Formula 2.1 shows.

$$\begin{bmatrix} 0000\,0000_2 \\ 0101\,0101_2 \\ 1010\,1010_2 \\ 1111\,1111_2 \end{bmatrix} \div 85_{10} = \begin{bmatrix} 00_2 \\ 01_2 \\ 10_2 \\ 11_2 \end{bmatrix} \tag{2.1}$$

As the final step, compressed values of every pixel of the cat image have been embedded on the image of tree. The hidden image reveals itself when these two least significant bits are extracted from stego image and have normalization applied on them to enlarge their color range back as Formula 2.2 shows.

$$\begin{bmatrix} 00_2 \\ 01_2 \\ 10_2 \\ 11_2 \end{bmatrix} * 85_{10} = \begin{bmatrix} 0000\,0000_2 \\ 0101\,0101_2 \\ 1010\,1010_2 \\ 1111\,1111_2 \end{bmatrix} \tag{2.2}$$

This example shows that Least Significant Bit method can be applied on any kind of digital carrier media to hide any kind of information albeit with some losses.

2.5.2.2 Techniques on frequency domain

On computer graphics, two-dimensional transforms are essential for image enhancement operations such as blurring, sharpening and contrast correction. Fourier Transform and Discrete Cosine Transform are two popular frequency domain transforms related to digital image processing. Some digital image formats such as JPEG (Discrete Cosine Transform in this case) utilizes transform operations while encoding as well.

[20] has proposed a steganography method on frequency domain that targets JPEG images as cover media by using 32x32 quantization tables instead of standard 8x8 quantization tables. This method resulted in reduced computation time with increased hidden message capacity while keeping image quality and file size at reasonable levels.

Westfeld's method [21] is a strong steganography method on JPEG files that resists visual and statistical attacks. By utilizing matrix encoding [22], this method requires a small amount of modifications to store hidden messages of same length.

CHAPTER 3

STEGANALYSIS OF DIGITAL MEDIA

As it has been explained in the previous chapter, aim of steganographic methods is to achieve secret communication by hiding messages into cover media indetectably. The intended readers for a stego message are only the sender and the receiver, thus only they should know whether a media contains a hidden message and if it does, how to reveal it. If the chosen steganographic method is strong enough, anyone else who receives the carrier would only notice the carrier itself but not detect the hidden message; which means steganography has succeeded.

Based on the format and properties of cover media, attackers may implement methods that analyse these cover media to determine whether the they carry hidden messages or not. The methods that aim to inspect and reveal information are referred as **steganalysis methods**. If succeeded, steganalysis methods usually produce results such as length of the hidden message or the message itself.

If steganalysis methods were to be applied on media which contains no hidden information, it is possible that the method may produce false positives. This may happen depending on the algorithm and the cover media itself.

Steganography methods on digital media such as digital images, digital audios and others have enabled proper analysis / attack methods to be developed in order to extract information from cover media. The following sections will briefly mention these methods by medium type.

3.1 Digital Images

Steganalysis methods that target digital images usually attack against steganography methods achieved on spatial and frequency domains. As with other media, there are elements that pose risk to accuracy of steganalysis of digital images. Modifications such as resizing, cropping, converting to a lossy format after embedding or other similar operations may cause steganalysis methods to fail or produce faulty results.

3.1.1 Steganalysis methods on spatial domain

Least Significant Bit steganography methods are the earliest examples of steganography on digital images, thus they are the most studied methods. Methods such as Sample Pair Analysis [23], Chi-Square [24], RS Analysis [25], are widely known steganalysis methods that target least significant bit methods. Specialised analysis methods for certain steganography methods have also been implemented. For example; [26], [27], [28] and [29] propose different approaches for attacking against LSB matching steganography, which was previously mentioned under Section 2.5.2.1.

3.1.1.1 Sample pair analysis

Sample Pair analysis is a statistical LSB steganography attack method that reveals hidden message length with high precision. In [23], results obtained by applying this method on several natural images with various message lengths *p* has been reported to be highly accurate.

3.1.1.2 Chi-Square analysis

This method is a statistical analysis method that works on cover images that carry sequentally embedded messages but not on images carrying pseudo-randomly scattered messages.

3.1.1.3 Steganalysis of regular and singular groups

Steganalysis of Regular and Singular Groups (RS analysis), proposed by Fridrich et al [25], can be used against both color and grayscale images and produces an approximate length of the hidden message in the image as result. This analysis method provides a basis for the method which will be explained in Chapter 4 of this thesis.

This method starts the analysis by first dividing an *MxN* image into groups that contain *n* pixels ($x_1, ..., x_n$). Value of these pixels are from set *P*. For example, $P = \{0, 1, ..., 255\}$ for an 8 bit grayscale image. A discrimination function f is defined for checking the pixels in each group *G* for irregularities between them and the differences among the pixels in groups are noted.

$$f(x_{1,}\dots,x_n)=\sum_{i=1}^{n-1}|x_{i+1}-x_i| \tag{3.1}$$

If the values of pixels in group G are close to each other, then the difference value for the group will be a small integer but groups containing noisy pixels will produce a higher value. This operation repeats until all the groups in the image have been traversed.

Then, a flipping operation *F* flips LSB's of each channel of each pixel such that;

$$F_1 = 0 \leftrightarrow 1, 2 \leftrightarrow 3, \dots, 254 \leftrightarrow 255 \tag{3.2}$$

Also, shifted LSB flipping operation F_{-1} has also been defined such that;

$$F_{-1}(x)=F_1(x+1)-1 \tag{3.3}$$

And identity permutation F_0 is defined as;

$$F_0(x)=x \tag{3.4}$$

The discrimination function *f* and flipping operation *F* are used for defining three pixel groups *R*, *S* and *U*. After calculating *f(F(G))* and *f(G)*, both results are compared:

- *f(F(G)) > f(G)* : group is regular
- *f(F(G)) < f(G)* : group is singular
- *f(F(G)) = f(G)* : group is unusable

A mask *M*, which is an n-tuple containing values -1, 0 and 1 is defined to capture assignment of flipping to each pixel. Thus, flipped group *F(G)* can be defined as $(F_{M(1)}(x_1), F_{M(2)}(x_2), ..., F_{M(n)}(x_n))$.

Using mask *M*, $f(F_M(G))$ has been calculated and $f(F_M(G))$ and *f(G)* are compared to calculate number of R, S and U groups. Negative mask *-M* is used for calculating $f(F_{-M}(G))$ and calculating number of negative R, S and U groups by comparing $f(F_{-M}(G))$ and *f(G)*.

Regular groups for mask *M* are denoted as R_M, singular groups for mask *M* are denoted as S_M, regular groups for negative mask *-M* are denoted as R_{-M} and singular groups for negative mask *-M* are denoted as S_{-M}. It has been explained that a typical image should have the following features:

$$R_M+S_M\leqslant 1 \tag{3.5}$$

$$R_{-M} + S_{-M} \leqslant 1 \tag{3.6}$$

$$R_M \approx R_{-M} \tag{3.7}$$

$$S_M \approx S_{-M} \tag{3.8}$$

As a result of their tests with a large image database, they have experimentally verified that as the amount of flipped pixels increase, values of R_M and S_M converge while values of R_{-M} and S_{-M} diverge. At 50%, R_M and S_M curves intersect. This behaviour has been shown in Figure 3.1 [25].

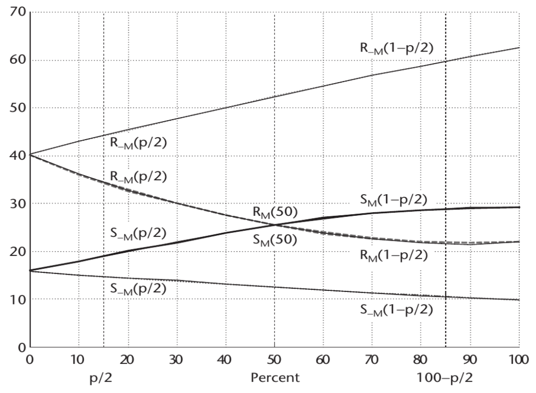


Figure 3.1 RS Diagram of an Image

In light of these information, they have derived the following formula for estimation of message length *p* after rescaling the graph above such that *p/2* becomes 0 and *100-p/2* becomes 1:

$$\begin{gathered} d_0 = R_M(p/2) - S_M(p/2) \\ d_{-0} = R_{-M}(p/2) - S_{-M}(p/2) \\ d_1 = R_M(1-p/2) - S_M(1-p/2) \\ d_{-1} = R_{-M}(1-p/2) - S_{-M}(1-p/2) \\ 2(d_1 + d_0)x^2 + (d_{-0} - d_{-1} - d_1 - 3\mathrm{d}_0)x + d_0 - d_{-0} = 0 \end{gathered} \tag{3.9}$$

Finally, p is calculated as;

$$p = x / (x - 1/2) \tag{3.10}$$

They have deduced the following factors affect the accuracy of RS method:

- **Initial bias:** It has been explained that initial bias can be caused by random variations; for example it has been observed that small images, scans of half-toned images and noisy images generate larger variations.
- **Noise:** Higher amounts of noise alters the difference between regular and singular groups' counts.
- **Message placement:** It has been explained that RS method produces more accurate results on images that carries scattered messages.

3.1.2 Steganalysis methods on frequency domain

The method proposed at [30] aims at detecting hidden messages embedded onto JPEG images. Their experimental results show that this method has high performance of detecting messages embedded via several JPEG steganography methods.

In [31], Fridrich et al has proposed a method for detecting hidden messages embedded with F5 algorithm. Their analysis method focuses on the changes on DCT coefficients caused by F5 algorithm. Their results show that this method can be used to detect even small amounts of modifications on DCT coefficients.

3.2 Other Digital Media

Other than digital images, it is possible to implement steganalysis methods for carrier formats such as digital audio and video as well. For example in [32], a steganalysis method that aims to differentiate stego signals from cover signals of an audio file has been proposed. Another steganalysis method for digital audio files has been proposed in [33] as well. This method can detect hidden messages in WAV files that have been embedded with Steghide [34].

CHAPTER 4

RS STEGANALYSIS WITH QUADTREE DATA STRUCTURE

Quadtree is a tree data structure, which has four children as nodes. Applying quadtree data structure on a dataset results in 4 independent subsets of the original dataset. Figure 4.1 shows a representation of quadtree segmentation. On the left is the representation of a sample data and on the right is the tree-form of this data. Quadtrees can recursively be applied several times on a dataset as long as capacity of the dataset allows further segmentation. Each recursion produces smaller child nodes that hold lesser data than their parent nodes.

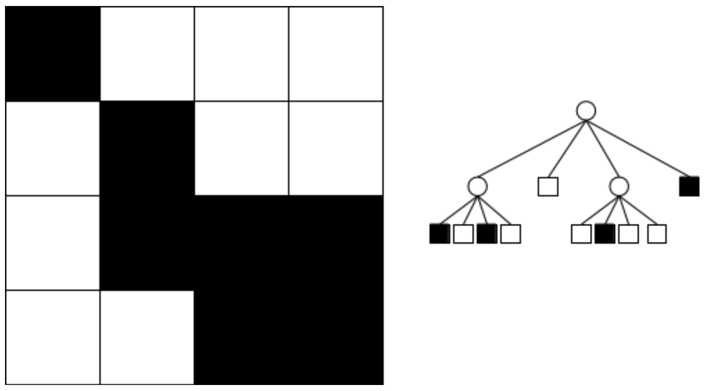


Figure 4.1 Quadtree

[35] and [36] are a few examples of quadtree utilization for different problems. While quadtrees are commonly used with image data, they may be used for segmentation of different data formats as well.

4.1 Quadtrees on Digital Images

Segmentation of a *MxN* digital image with quadtrees results in 4 child nodes, each with width=*M/2* and height=*N/2*; in other words each child node represents one quarter of the original image. Figure 4.2 shows a 512x512 image which has been divided into segments several times with quadtree data structure.

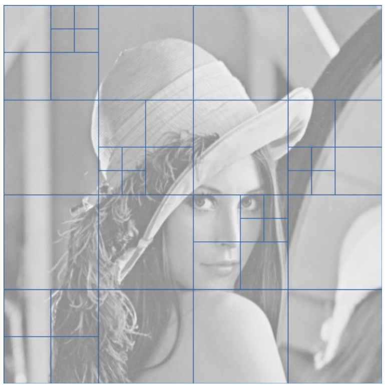


Figure 4.2 An Image with Quadtree Segmentation

As a result of first segmentation, 4 child nodes of resolution 256x256 have been obtained. By repeating this process recursively on these child nodes, further smaller nodes have been obtained as well. The smallest segments represented at this figure have been obtained by applying quadtrees recursively 4 times on the original image. Size of these segments are 32x32 and they each are 1/64th of the original image.

Quadtree's can be applied on digital images for various reasons depending on the purpose. [37] have used quadtrees for partial encryption of images, while [38] have proposed using quadtrees for their image compression technique.

4.2 RS Steganalysis with Quadtrees

In the previous chapter, it was explained that applying RS steganalysis method on to an image *I* resulted in a single value, estimated message length *p*, for a single image. Since this result has been obtained by analysing the whole image, it would not help us identify at which location(s) the message has been embedded at but give us an overall estimation of the length of the message hidden on the image.

But if we instead were to apply quadtree algorithm to an image first, the image would be divided into 4 smaller images. Then, running RS steganalysis method independently on each quadrant would give us 4 independent analysis results. As these results are now responsible for their own quadrants, it can be observed which quadrants carry higher amounts of hidden messages and which quadrants carry less. By recursively repeating this process until a defined depth *D* has been reached, we can determine precise location(s) of hidden messages. It should be noted that *D* must not be a large value because as the size of the image gets smaller, larger variations can be observed; which leads to erroneous analysis results (defined previously under Section 3.1.1.3).

If the message has not been scattered randomly but embedded as blocks instead, this method can help us save time by not trying to extract message from unmodified regions while working with huge images or while working on batch operations with large amounts of images.

To simplify analysis operations with quadtrees, a minimum threshold percentage T_L can be defined; so we can denote which quadrants carry insignificant amounts of hidden message. Defining an upper threshold percentage T_H also proves useful for denoting which quadrants hold maximum amount of hidden message. As a result, the quadrants that don't meet these threshold requirements can be ignored so that further analyses won't be conducted on them.

Algorithm of this proposed method is as follows:

1. Conduct RS analysis on I,
2. If results > T_L and results < T_H then continue, else stop
3. If depth < D, then create quadrants I_{TL}, I_{TR}, I_{BL}, I_{BR} from image I, else stop
4. Repeat from Step 1 for each quadrant I_{TL}, I_{TR}, I_{BL}, I_{BR}.

CHAPTER 5

EXPERIMENTS AND RESULTS

To see how the method which has been explained in Section 4.2 works, several tests were conducted on a large dataset of 1.338 digital images obtained from [39]. These images have been converted to grayscale BMP while keeping their resolution intact.

In order to conduct these tests, algorithm of RS analysis method has been implemented in Python with inspiration from an existing and working implementation of RS analysis written in Java and licensed with General Public License Version 3 (GPLv3) [40]. After the initial port, a few modifications has also been implemented. For imaging operations, Python extension of OpenCV [41] has been used. For quadtree support, a recursive function that calls itself 4 times with different quadrants of the provided image has been implemented. Recursion level can be controlled with a variable in the program. This code is presented at Appendix A.

First of all, the images have been analysed with masks {1, 0, 1, 0} and {0, 1, 0, 1} and their initial results have been recorded. Images that produce the lowest and highest result values are shown in Figure 5.1 with their respective estimated message length.

While the results of a huge majority of the images vary between 0 and 4000, reason of ucid01049.bmp generating such a higher result is the high amounts of variations between neighbouring pixels. Figure 5.2 displays a zoomed-in section of this image. Non-zero results obtained from these initial analyses can be associated with the three factors that affect the accuracy of RS method which have been explained previously under Section 3.1.1.3.

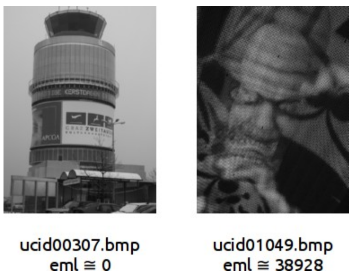


Figure 5.1 Highest and Lowest Initial Results

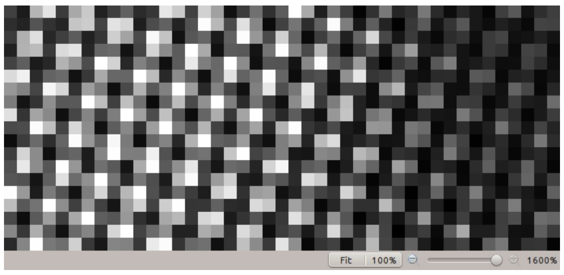


Figure 5.2 ucid01049.bmp, Zoomed-in

Based on these initial results, images have been grouped into three: *smooth*, *normal* and *noisy*. Amount of pictures in each group are given in Table 5.1 with their thresholds.

Table 5.1 Image Groups

Group	Threshold	Amount of Images
Smooth	[0 – 500]	744
Normal	[500 – 4000]	563
Noisy	[4000 – 40000]	31

Every image has also been analysed with 1-level quadtree segmentation and results of each quarter has been recorded as well. After obtaining these initial results, the first test has been conducted. For this purpose, bottom-left quadrant of each image (1/4th) has been extracted and had randomly generated texts of 6.164 and 6.172 bytes long (depending on the resolution of the quadrant) embedded into them. The code used for generating these random messages has been displayed in Appendix A, along with how to use it. After embedding, these extracted stego-quadrants has been merged back with the original images; replacing original bottom-left quadrants. This creation process has been shown in Figure 5.3.

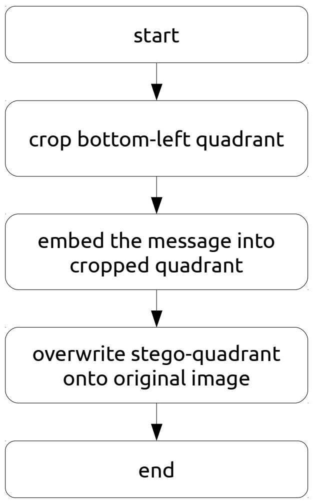

start
crop bottom-left quadrant
embed the message into cropped quadrant
overwrite stego-quadrant onto original image
end

Figure 5.3 Embedding Process

After running the analysis code on the stego images, obtained results have been compared with the original results and the following behaviours have been observed:

- Independent results of bottom-left quadrants of all images from all three sets (smooth, normal, noisy) have increased almost by the length of the embedded message while the independent results of other quadrants stay same.

- Overall results of smooth images have increased almost by the message length as well. But it has been observed that results of some of the most noisy images from noisy set have decreased, rather than increasing. This can be associated with the fact that modifying bits of already noisy sets of pixels may have reduced the variance between them.
- Of 1.338 images; bottom-left quadrant analysis results of 5 images have decreased, while results of 1.333 images has increased.

Next, an another test has been conducted. This time, a depth-2 quadrant (1/16th) of the original image has been cropped. The same steps from the first test has been repeated with a shorter message of length 1.560 bytes. After this test, the following behaviours have been observed:

Table 5.2 Test 2 Results

Noisy Images			
	Overall	**1/4th Quadrants**	**1/16th Quadrants**
Increased	16	22	28
Decreased	15	9	3
Normal Images			
	Overall	**1/4th Quadrants**	**1/16th Quadrants**
Increased	363	524	558
Decreased	200	39	5
Smooth Images			
	Overall	**1/4th Quadrants**	**1/16th Quadrants**
Increased	712	729	740
Decreased	32	15	4

From these results, it is safe to say that hiding a message that is 1/16th long of the cover image;

1. Has caused noticable amount of changes on smooth images but ratio of change gets lower with images in normal and noisy groups,
2. As the depth of analysis increases (or in other words, size of the analysed quadrant gets smaller), amount of quadrants which generate higher values increase.

Based on these results obtained from 1.338 images, the following results have been obtained:

1. Noisier images have a higher chance of deceiving the analyser at initial analyses but further analyses on quadrants can still lead the analyser to locations where the message has been embedded.
2. Smooth images with hidden messages don't deceive the analyser with low values after initial analyses, they let the analyser to know they are carrying hidden messages in the beginning of analyses.

CHAPTER 6

CONCLUSION

The aim of this thesis is locating the positions of hidden messages on a digital image. The chosen steganalysis method provides an overall estimated message length for a given image but does not tell us where this message is located at. This steganalysis method has been combined with quadtree data structure; this has enabled analyzing smaller parts of the image and obtain independent results for each part, thus knowledge about which parts of the image generate higher results and which parts don't has been obtained. Thanks to this method, parts containing higher amounts of messages can be focused and other parts can be skipped to speed-up revealing the message.

To prove this method, a large dataset of images has been used to create stego images carrying sequential blocks of messages. Depending on the noisiness / smoothness of these images, mixed results have been obtained initially but further analyses on smaller quadrants usually proved useful to detect where the messages are located at.

6.1 Future Work

In this thesis, the proposed method has been tested with RS analysis method. Instead of RS steganalysis method, other spatial domain steganalysis methods can be chosen to work with quadtree data structure to observe the success rate of each method for finding location of hidden messages.

REFERENCES

[1] http://en.wikipedia.org/wiki/Steganography

[2] **SIMMONS, G.** (1984), *The Prisoners' Problem and the Subliminal Channel*, Advances in Cryptology, 51-67.

[3] **COX, I.J.** et al (1997), *Secure Spread Spectrum Watermarking for Multimedia*, IEEE Transactions on Image Processing, Vol. 6, Issue 12, 1673-1687

[4] **HSU, C., WU, J.** (1999), *Hidden Digital Watermarks in Images*, IEEE Transactions on Image Processing, Vol. 8, Issue 1, 58-68

[5] **CHEDDAD, A.** et al (2010), *Digital image steganography: Survey and analysis of current methods*, Signal Processing, Vol. 90, Issue 3, 727-752

[6] **DELFOROUZI, A., POOYAN, M.** (2008), *Adaptive Digital Audio Steganography Based on Integer Wavelet Transform*, Circuits, Systems & Signal Processing, Vol. 27, Issue 2, 247-259

[7] **YAN, D.** et al (2012), *Steganography for MP3 Audio By Exploiting the Rule of Window Switching,* Computers & Security, Vol. 31, Issue 5, 704-716

[8] **YAN, D., WANG, R.** (2011), *Huffman Table Swapping-based Steganography for MP3 Audio*, Multimedia Tools and Applications, Vol. 52, Issue 2-3, 291-305

[9] **RAJA, K.B.** et al (2005), *A Secure Image Steganography using LSB, DCT and Compression Techniques on Raw Images*", Third International Conference on Intelligent Sensing and Information Processing, 170-176

[10] **XU, C., PING, X.** (2007), *A Steganographic Algorithm in Uncompressed Video Sequence Based on Difference between Adjacent Frames*, Fourth International Conference on Image and Graphics, 297-302

[11] **XU, C.** et al (2006), *Steganography in Compressed Video Stream*, First International Conference on Innovative Computing, Information and Control, 269-272

[12] **CHAE, J.J., MANJUNATH, B.S.** (1999), *Data Hiding in Video*, International Conference on Image Processing, Vol. 1, 311-315

[13] **BHOLE, A., PATEL, R.** (2012), *Steganography over Video File using Random Byte Hiding and LSB Technique*, IEEE International Conference on Computational Intelligence & Computing Research, 1-6

[14] **CHAO, C.** et al (2006), *Information Hiding in Text Using Typesetting Tools with Stego-Encoding*, First International Conference on Innovative Computing, Information and Control, Vol. 1, 459-462

[15] **MIELIKAINEN, J.** (2006), *LSB Matching Revisited*, IEEE Signal Processing Letters, Vol. 13, Issue 5, 285-287

[16] **LUO, W.** et al (2010), *Edge Adaptive Image Steganography Based on LSB Matching Revisited*, IEEE Transactions on Information Forensics and Security, Vol. 5, Issue 2, 201-214

[17] **XI, L.** et al (2010), *Improved LSB Matching Steganography Resisting Histogram Attacks*, 3rd IEEE International Conference on Computer Science and Information Technology, 203-206

[18] **QIUDONG, Y., LIU, X.** (2009), *A New LSB Matching Steganographic Method Based on Steganographic Information Table*, Second International Conference on Intelligent Networks and Intelligent Systems, 362-365

[19] http://en.wikipedia.org/wiki/Steganography#Digital

[20] **VONGURAI, N., PHIMOLTARES, S.** (2012), *Frequency-Based Steganography Using 32x32 Interpolated Quantization Table and Discrete Cosine Transform*, Fourth International Conference on Computational Intelligence, 249-253

[21] **WESTFELD, A.** (2001), *F5 – A Steganographic Algorithm: High Capacity Despite Better Steganalysis*, 4th International Workshop on Information Hiding, 289-302

[22] **CRANDALL, R.** (1998), *Some Notes on Steganography*, http://www.di.unisa.it/~ads/corso-security/www/CORSO-0203/steganografia/LINKS%20LOCALI/matrix-encoding.pdf

[23] **DUMITRESCU, S.** et al (2003), *Detection of LSB Steganography via Sample Pair Analysis*, IEEE Transactions on Signal Processing, Vol. 51, Issue 7, 1995-2007

[24] **WESTFELD, A., PFITZMANN, A.** (1999), *Attacks on Steganographic Systems*, Information Hiding

[25] **FRIDRICH, J.** et al (2001), *Reliable Detection of LSB Steganography in Color and Grayscale Images*, IEEE Multimedia, Vol. 8, Issue 4, 2001, 22-28.

[26] **HUANG, F.** et al (2007), *Attack LSB Matching Steganography by Counting Alteration Rate of the Number of Neighborhood Gray Levels*, IEEE International Conference on Image Processing, 401-404

[27] **GUI, X.** et al (2012), *Improved Payload Location for LSB Matching Steganography*, 19th IEEE International Conference on Image Processing, 1125-1128

[28] **ZHANG, T.** et al (2010), *Detection of LSB Matching Steganography Based on Distribution of Pixel Differences in Natural Images*, International Conference on Image Analysis and Signal Processing, 548-552

[29] **YU, X., BABAGUCHI, N.** (2008), *An Improved Steganalysis Method of LSB Matching*, International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 557-560

[30] **LIU, Q.** et al (2010), *An Improved Approach to Steganalysis of JPEG Images*, Information Sciences, Vol. 180, Issue 9, 1643-1655

[31] **FRIDRICH, J.** et al (2003), *Steganalysis of JPEG Images: Breaking the F5 Algorithm*, Information Hiding

[32] **AVCIBAŞ, İ.** (2006), *Audio Steganalysis With Content-Independent Distortion Measures*, IEEE Signal Processing Letters, Vol. 13, Issue 2, 92-95

[33] **RU, X.** et al (2005), *Steganalysis of Audio: Attacking the Steghide*, Proceedings of 2005 International Conference on Machine Learning and Cybernetics, Vol. 7, 3937-3942

[34] http://steghide.sourceforge.net/

[35] **KAMBHAMPATI, S., DAVIS, L.S.** (1986), *Multiresolution path planning for mobile robots*, IEEE Journal of Robotics and Automation, Vol. 2, Issue 3, 135-145

[36] **AVIN, C.** (2011), *Geographical Quadtree Routing*, IEEE Symposium on Computers and Communications, 302-308

[37] **CHENG, H., XIAOBO, L.** (2000), *Partial Encryption of Compressed Images and Videos*, IEEE Transactions on Signal Processing, Vol. 48, Issue 8, 2439-2451

[38] **VAISEY, J., GERSHO, A.** (1992), *Image Compression With Variable Block Size Segmentation*, IEEE Transactions on Signal Processing, Vol. 40, Issue 8, 2040-2060

[39] http://homepages.lboro.ac.uk/~cogs/datasets/ucid/ucid.html

[40] http://vsl.sourceforge.net/

[41] http://opencv.willowgarage.com/

APPENDIX A

RS STEGANALYSIS CODE WITH QUADTREE DATA STRUCTURE

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

import cv, sys

class RSAnalysis:
    def __init__(self, image, m, n):
        k = 0
        self.mMask = [[None] * m * n, [None] * m * n]
        for i in range(n):
            for j in range(m):
                if (((j % 2) == 0 and (i % 2) == 0) or
                    ((j % 2) == 1 and (i % 2) == 1)):
                    self.mMask[0][k] = 1
                    self.mMask[1][k] = 0
                else:
                    self.mMask[0][k] = 0
                    self.mMask[1][k] = 1
                k += 1
        self.image = image
        self.imgy = image.height
        self.imgx = image.width
        self.mM = m
        self.mN = n

    def doAnalysis(self):
        startx = 0; starty = 0
        block = [None] * self.mM * self.mN
        numregular = 0; numsingular = 0
        numnegreg = 0; numnegsing = 0
        numunusable = 0; numnegunusable = 0

        while startx < self.imgx and starty < self.imgy:
            for m in range(2):
                k = 0
                for i in range(mN):
                    for j in range(mM):
                        block[k] = int(self.image[starty + i, startx + j][0])
                        k += 1

                variationB = self.getVariation(block)
                block = self.flipBlock(block, self.mMask[m])
                variationP = self.getVariation(block)
                block = self.flipBlock(block, self.mMask[m])
                self.mMask[m] = self.invertMask(self.mMask[m])
                variationN = self.getNegativeVariation(block, self.mMask[m])
                self.mMask[m] = self.invertMask(self.mMask[m])
                if variationP > variationB:
                    numregular += 1
                elif variationP < variationB:
                    numsingular += 1
                else:
                    numunusable += 1

                if variationN > variationB:
                    numnegreg += 1
                elif variationN < variationB:
                    numnegsing += 1
                else:
                    numnegunusable += 1

            startx += 1
            if startx >= self.imgx - self.mM:
                startx = 0
                starty += 1
            if starty >= self.imgy - self.mN:
                break

        totalgroups = numregular + numsingular + numunusable
        allpixels = self.getAllPixelFlips()
        x = self.getX(numregular, numnegreg, allpixels[0], allpixels[2],
                      numsingular, numnegsing, allpixels[1], allpixels[3])

        if 2 * (x - 1) == 0:
            epf = 0
        else:
            epf = abs(x / (2 * (x - 1)))

        if x - 0.5 == 0:
            ml = 0
        else:
            ml = abs(x / (x - 0.5))

        results = [None, None, None, None]
        results[0] = totalgroups
        results[1] = epf
        results[2] = ml
        results[3] = (self.imgx * self.imgy * ml) / 8
        return results

    def getAllPixelFlips(self):
        allmask = [1] * self.mM * self.mN
        block = [None] * self.mM * self.mN
        startx = 0; starty = 0
        numregular = 0; numsingular = 0
        numnegreg = 0; numnegsing = 0
        numunusable = 0; numnegunusable = 0

        while startx < self.imgx and starty < self.imgy:
            for m in range(2):
                k = 0
                for i in range(self.mN):
                    for j in range(self.mM):
                        block[k] = int(self.image[starty + i, startx + j][0])
                        k += 1

                block = self.flipBlock(block, allmask)
                variationB = self.getVariation(block)
                block = self.flipBlock(block, self.mMask[m])
                variationP = self.getVariation(block)
                block = self.flipBlock(block, self.mMask[m])
                self.mMask[m] = self.invertMask(self.mMask[m])
                variationN = self.getNegativeVariation(block, self.mMask[m])
                self.mMask[m] = self.invertMask(self.mMask[m])

                if variationP > variationB:
                    numregular += 1
                elif variationP < variationB:
                    numsingular += 1
                else:
                    numunusable += 1

                if variationN > variationB:
                    numnegreg += 1
                elif variationN < variationB:
                    numnegsing += 1
                else:
                    numnegunusable += 1
            startx += 1
            if startx >= self.imgx - self.mM:
                startx = 0
                starty += 1
            if starty >= self.imgy - self.mN:
                break

        results = [None] * 4
        results[0] = numregular
        results[1] = numsingular
        results[2] = numnegreg
        results[3] = numnegsing
        return results

    def getX(self, r, rm, r1, rm1, s, sm, s1, sm1):
        dzero = r - s
        dminuszero = rm - sm
        done = r1 - s1
        dminusone = rm1 - sm1

        a = 2 * (done + dzero)
        b = dminuszero - dminusone - done - (3 * dzero)
        c = dzero - dminuszero

        if a == 0:
            x = c / b

        discriminant = (b ** 2) - (4 * a * c)

        if discriminant >= 0:
            rootpos = ((-1 * b) + (discriminant ** .5)) / (2 * a)
            rootneg = ((-1 * b) - (discriminant ** .5)) / (2 * a)

            if abs(rootpos) <= abs(rootneg):
                x = rootpos
            else:
                x = rootneg
        else:
            cr = (rm - r) / (r1 - r + rm - rm1)
            cs = (sm - s) / (s1 - s + sm - sm1)
            x = (cr + cs) / 2

        if x == 0:
            a_r = ((rm1 - r1 + r - rm) + (rm - r) / x) / (x - 1)
            a_s = ((sm1 - s1 + s - sm) + (sm - s) / x) / (x - 1)
            if a_s > 0 or a_r < 0:
                cr = (rm - r) / (r1 - r + rm - rm1)
                cs = (sm - s) / (s1 - s + sm - sm1)
                x = (cr + cs) / 2
        return x

    def getVariation(self, block):
        var = 0
        for i in range(len(block) - 1):
            var += abs(block[i] - block[i + 1])
        return var

    def getNegativeVariation(self, block, mask):
        var = 0
        for i in range(len(block) - 1):
            colour1 = block[i]; colour2 = block[i + 1]
            if mask[i] == -1:
                colour1 = self.invertLSB(colour1)
            if mask[i + 1] == -1:
                colour2 = self.invertLSB(colour2)
            var += abs(colour1 - colour2)
        return var

    def flipBlock(self, block, mask):
        for i in range(len(mask)):
            if mask[i] == 1:
                block[i] = self.negateLSB(block[i])
            elif mask[i] == -1:
                block[i] = self.invertLSB(block[i])
        return block
    def invertLSB(self, abyte):
        if abyte == 255:
            return 256
        if abyte == 256:
            return 255
        return self.negateLSB(abyte + 1) - 1

    def negateLSB(self, abyte):
        temp = abyte & 0xfe
        if temp == abyte:
            return abyte | 0x1
        else:
            return temp

    def invertMask(self, mask):
        for i in range(len(mask)):
            mask[i] *= -1
        return mask

def recursion(image):
    if image.width >= minwidth and image.height >= minheight:
        hw = image.width / 2
        hh = image.height / 2
        try:
            print int(RSAnalysis(image, mM, mN).doAnalysis()[3]),
        except:
            print sys.exc_info()
            pass

        recursion(cv.GetSubRect(image, (0, 0, hw, hh)))   # top left
        recursion(cv.GetSubRect(image, (hw, 0, hw, hh)))  # top right
        recursion(cv.GetSubRect(image, (0, hh, hw, hh)))  # bottom left
        recursion(cv.GetSubRect(image, (hw, hh, hw, hh))) # bottom right

if len(sys.argv) != 2:
    print 'You must specify an image file!'
    sys.exit()
else:
    path = sys.argv[1]

image = cv.LoadImage(path)
width = image.width
height = image.height

DEPTH = 1
mM = 4
mN = 1
minwidth = width / (2 ** DEPTH)
minheight = height / (2 ** DEPTH)
recursion(image)
```

APPENDIX B

RANDOM MESSAGE GENERATION CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char** argv)
{
    int i;
    char lower[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
                     'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
                     'y', 'z', ' ' };
    char upper[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
                     'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
                     'Y', 'Z', ' ' };
    char numer[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ' };

    if (argc != 2) {
        printf("Usage: %s, <msg size>\n", argv[0]);
        return 0;
    }

    srand(time(0));

    for (i = 0; i < atoi(argv[1]); i++) {
        switch (rand() % 3) {
        case 0:
            printf("%c", lower[rand() % 27]);
            break;
        case 1:
            printf("%c", upper[rand() % 27]);
            break;
        case 2:
            printf("%c", numer[rand() % 11]);
            break;
        }
    }

    return 0;
}
```

Compilation, sample run and sample output of the program:

```
$ gcc randstrgen.c -o randstrgen
$ ./randstrgen 20 > msg20
$ cat msg20
UcGHoX6cO0 AZ174f vp
```

Desired length of the message should be provided as a command line parameter. In this example, a 20 bytes long message has been generated and stored in a file named "msg20". This file can be used for embedding operation at a later time.

APPENDIX C

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Çiftci, Efe
Nationality: Turkish (TC)
Date and Place of Birth: 21 May 1986, Samsun
Marital Status: Single
E-mail: efeciftci@gmail.com

EDUCATION

Degree	Instution	Year of Graduation
BS	Çankaya Univ. Dept. of Computer Engineering	2008

WORK EXPERIENCE

Year	Place	Enrollment
2009 - Present	Çankaya Univ. Dept. of Computer Engineering	Specialist
2008 - 2009	Avian Software	Software Developer

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. Medeni, İ. T., Çiftci, E., "A Decision Support System Proposal for the Engineering Students' Specialty Area Selection", WCIT 2011
2. Yıldırım, A. A., Çiftci, E., Özdoğan, C., "Geniş Veri Kümeleri Üzerinde Paralel Veri Madenciliği Yaklaşımları: Wavecluster Yöntemi ile Öbekleme Uygulaması", 3. Mühendislik ve Teknoloji Sempozyumu
3. Tokmak, V., Şengez, Y., Çiftci, E., Güngören, B., Aluftekin, N., Aktaş, Z., "Web Tabanlı İnsan Kaynakları Yönetim Sistemi", Akademik Bilişim 2008

HOBBIES

Reading books, listening to music, watching movies, swimming.