جـــامـعـــة الـــشـــرق الأوســـط
MIDDLE EAST UNIVERSITY

A New Audio Steganography Method Using Bi-LSB Embedding and Secret Message Integrity Validation

طريقة جديدة في علم اخفاء المعلومات الصوتية باستخدام أسلوب التخزين (Bi-LSB) و التحقق من سلامة الرسالة

By

Mahmood Maher Salih

Superviser

Dr. Mudhafar Al-Jarrah

A Thesis Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Science

Department of Computer Science

Faculty of Information Technology

Middle East University

Augest / 2015

بسم الله الرحمن الرحيم

((إِنَّ الَّذِينَ آمَنُوا وَعَمِلُوا الصَّالِحَاتِ وَأَقَامُوا الصَّلَاةَ وَآتَوُا الزَّكَاةَ لَهُمْ أَجْرُهُمْ عِندَ رَبِّهِمْ وَلَا خَوْفٌ عَلَيْهِمْ وَلَا هُمْ يَحْزَنُونَ))

صدق الله العظيم

سورة البقرة / اية (277)

Authorization

I, Mahmood Maher Salih, authorize Middle East University (MEU) to provide libraries, organizations and even individuals with copies of my thesis when required.

Name: , Mahmood Maher Salih

Date: Sat. 18 August, 2015

Signature:

III

Examination Committee Decision

This is to certify that the thesis entitled "**A New Audio Steganography Method Using Bi-LSB Embedding and Secret Message Integrity Validation**" was successfully defended and approved on 18/8/2015.

Examination Committee Members	**Signature**
(superviser) **Dr. Mudhafar Al-Jarrah**	 .....
(Internal Committee Member) **Prof. Dr. Ahmed Kayed**	 .....
(External Committee Member) **Dr. Mohammed Salem Al-Atoum**	 .....

Acknowledgment

I would like to thank Dr. Mudhafar Al-Jarrah for his supervision and continued collaboration in each phase of the thesis, and his instructions that was very useful. Also I would like to thank all Doctors who seek to develop our skills and knowledge.

I am greatly indebted to my father and my mother, brothers, sisters and friends.

I am greatly indebted to Dr Mohammed Salem Mohammed Al-Atoum for his support and explaning the related work which his authored

Dedication

To The Remembrance My Brother Dr. Ahmed

Table of Contents

List of Figures

List of Tables

Abbreviations

A/D	Analog to Digital
AWGN	Additive white Gaussian noise
BAF	Embedding Before All Frames
BF	Embedding Between Frames
Bi-LSB	Bi Least Significant Bit
D/A	Digital to Analog
db	decibels
IEC	International Electro-Technical Commission
ISO	International Standard Organization
LSB	Least Significant Bit
M16M	Mod 16 Method
M4M	Mod 4 Method
MPEG	Moving Picture Expert Group
MSE	Mean Square Error
PBS	Padding Byte Stuffing
PSNR	Peak Signal to Noise Ratio
SO	Stego Object
HAS	human auditory system
STMDF	Steganographic technique based on minimum deviation of fidelity
UHBS	Unused Header Bit Stuffing

A new Audio Steganography Method Using Bi-LSB Embedding and Secret Message Integrity Validation

by

Mahmood Maher Salih

Superviser

Dr. Mudhafar Al-Jarrah

Abstract

Steganography is an advanced data hiding technique that has been widely investigated in the recent years due to its efficiency in protecting the security of information exchanged over the internet. Recently, several steganography methods have been introduced and developed to ensure the transmission of data in a secured way. However, some of those methods, such as the Least Significant Bit (LSB) technique have certain limitations concerning the verification of attacks in the secret messages. Therefore, this work introduces the development of Bi-LSB steganography method using the MATLAB program as a solution for the low security and capacity limitations of the traditional used LSB techniques.

The performance of the developed Bi-LSB technique is then compared with that of three traditional LSB techniques; 1-LSB, 2-LSB and 4-LSB based on using them in embedding the same secret messages in the same cover ones and then computing the Peak Signal to Noise Ratio (PSNR) values after adding a AWGN with different variance values to the stego file before extracting the secret message. In addition, various methods of integrity are used. A comparison is performed among the extracted messages and the original ones with the use of three integrity techniques; checksum, hash function and frequency techniques. The obtained results illustrated that the presented Bi-LSB technique outperforms the traditional LSBs in terms of PSNR values. It can be concluded that the increase in attack variance results in more degradation in the PSNR value of the secret message. In addition, the hash function check method offers the best correlation percentage among the extracted messages and the original ones.

Keywords: Steganography, LSB, Bi-LSB, embedding, extracting, AWGN, integrity

طريقة جديدة في علم اخفاء المعلومات الصوتية باستخدام اسلوب التخزين (Bi-LSB) و التحقق من سلامة الرسالة

الطالب

محمود ماهر صالح علاوي الجبوري

المشرف

الدكتور مظفر الجراح

ملخص الرسالة

إخفاء المعلومات هي تقنية متقدمة لإخفاء البيانات حيث تم البحث فيها على نطاق واسع في السنوات الأخيرة نظرا لكفاءتها في حماية أمن المعلومات المتبادلة عبر شبكة الانترنت. في السنوات الاخيرة الماضية, تم تقديم وتطوير عدة طرق لإخفاء المعلومات لضمان نقل البيانات بطريقة آمنة. ومع ذلك، فإن بعض تلك الطرق مثل تقنية LSB لديها بعض القيود فيما يتعلق بالتحقق من الهجمات على الرسائل السرية. لذلك، يقدم هذا العمل تصميم طريقة Bi-LSB لإخفاء المعلومات باستخدام برنامج الماتلاب كحل للامنية والقدرة المنخفضة لتقنيات ال LSB التقليدية.

في هذا العمل, تم مقارنة اداء طريقة ال Bi-LSB مع اداء ثلاثة تقنيات تقليدية ل LSB وهي: 1-LSB, 2-LSB & 4- LSB بالاعتماد على استخدامهم في اخفاء نفس الرسائل السرية بنفس الرسائل ومن ثم حساب قيم نسبة ذروة الاشارة الى الضوضاء (PSNR) بعد إضافة AWGN مع قيم تباين مختلفة إلى ملف ال stego قبل استخراج الرسالة السرية. تم عمل مقارنة ايضا بين الرسائل المستخرجة والرسائل السرية باستخدام ثلاثة طرق لل integrity وهي checksum, hash function and frequency techniques. أوضحت النتائج ان تقنية Bi-LSB تتفوق على طرق ال LSB التقليدية من حيث قيم PSNR. يمكن تلخيص ان الزيادة في قيمة التباين للهجمات يزيد من الانخفاض في قيمة الPSNR للرسائل السرية. . وبالإضافة إلى ذلك، طريقة ال hash function check تعطي افضل نسبة الارتباط بين الرسائل المستخرجة والرسائل الاصلية

كلمات مفتاحيه: علم الاخفاء , التضمين , التحقق من سلامة الرساله , اخفاء المعلومات الصوتية

Chapter One

Introduction

Chapter One: Introduction

1.1. Introduction

Information is shared globally through the Internet, in digital form (Fricker & Schonlau, 2002). There are issues and challenges regarding the security of information in transit from senders to receivers. The major issue is the protection of digital data against any form of intrusion, penetration, and theft. The major challenge is developing a solution to protect information and ensure their security during transmission (Feruza & Kim, 2007). Three components of information security are confidentiality, integrity, and availability (Feruza & Kim, 2007). Confidentiality ensures that information is kept secret from any unauthorized access. This could be done through information hiding techniques, namely cryptography and steganography (Lenti, 2000).

Cryptography involves the act of encryption and decryption of a digital data. The major weaknesses of such techniques are that even though the message has been encrypted, it still exists. Steganography dwells on concealing any digital data in an innocuous digital carrier, the word steganography is derived from an old Greek word which means covered writing (Katzenbeisser & Petitcolas, 2000).

Steganography has been used in for concealing secret messages during ancient times (Rahim, Bhattacharjee & Aziz, 2014). It was used by Histiaeus, the tyrant of Miletus, who, in 499 BC, tattooed the scalps of his slaves with a hidden message with a command for his men to attack the Persian (Huayin & Chang-Tsun, 2008; Emelia, Sugathan & Ho, 2008). The message became hidden when the slaves' hair grew back. According to researchers, steganography can be described as a study of the means of hiding secondary information within primary information without affecting the size of information nor the cause of any form of distortion which could be perceived (Liu, Sung & Qiao, 2009); (Ganeshkumar & Koggalage, 2009); (Petrovic & Yang, 2009); (Lee, Bell, Huang, Wang & Shyu, 2009);(Jangra & Singh, 2014).

The primary information, known as the carrier or host, was embedded within the secondary information, which is typically hidden and could be in the form of a file or message. The media with the embedded information is called stego signal, file, bit stream or sequence (Matthews, 2003);(Khairullah, 2009); (Alla, Prasad & Siva, 2009); (Changder, Debnath & Ghosh, 2009); (Qi, Ye, & Liu, 2009); (Farouk, 2014).

Steganography is one of the two techniques used for covert communication. However, watermarking is the second technique that can embed watermark into host cover to keep copyright for the hosts. Steganography typically establishes point-to-point data security (Mandala, Kotagiri & Kapala, 2013). The strength of steganographic technique, in keeping the data in the carrier medium against attacks or alteration is weak during transmission, storage or format conversion is weak (Katzenbeisser & Petitcolas, 2000)

The process of embedding information in host media in steganography technique and watermarking is usually done transparently (Manimegalai, Gomathi, Ponniselvi & Santha, 2014). The difference between steganography and watermarking is that while steganography is a technique which hides the information, visible watermarking actually allows the third person to see the message (Cvejic & Seppanen, 2004);(Neeta, Snehal & Jacobs, D, 2006).

Thus, in terms of watermarking visible and invisible, the process needs to ensure robustness so that any intentional attacks would not compromise, remove, or cause destruction of the information in any way in the marked media while at the same time preserving the quality of the signal (Scagliola, Pérez & Guccione, 2009); (Bhattacharyya & Sanyal, 2012). The invisible watermarking technique is the most suitable technique in cases where knowledge of the hidden information could cause possible manipulations (Yusnita & Othman, 2007); (Naji, Zaidan, Zaidan, Shihab & Khalifa, 2009).

1.2. Problem Statement

Audio steganography is an efficient method to secure embedded data and sent it through internet. Unfortintully the integrity message method is not focuses in steganography technique as well as LSB technique is not introduced encrypted method before embedding secret message.

As result this work introduces the development of an advanced Bi Least Significant Bit (Bi-LSB) MP3 audio steganography method to addresses the security problems of LSB. Furthermore, an integrity part is added at the receiver to ensure the integrity messages is recived correctly or not.

1.3. Research Questions

The main research question is: how to develop a new steganography algorithm based on the LSB technique to address the security issues?

Several other questions relating to this:

- How to implement the Bi-LSB technique in MP3 steganography?
- How to implement a new attack to the Bi-LSB technique?
- What is the best evaluation integrity method for the extracted messages?

1.4. Research Objectives

This study dwells on providing Bi Least Significant Bit (Bi-LSB) MP3 audio steganography technique that will circumvent the drawbacks studied in the LSB techniques. The specific objectives are:

- To design algorithm for MP3 audio (Bi Least Significant Bit) steganography systems capable of solving security problems observed in LSB technique.
- To implement an attack in MP3 steganography
- To ensure verifiable extraction of the embedded message.

1.5. Research Scope

This research focuses on Bi LSB technique on MP3 audio files. The unit of analysis is the security of the proposed technique. The following are the highlights:

- Embedding audio into MP3 audio format after compression.
- Using spatial domain.
- Bi LSB algorithm.
- This research does not focus on real time application.
- Applying different methods for Integrity of the message.

1.6. Contribution

The main contributions of the current work are:

- Enhancement the performance of PSNR compared with three traditional LSB techniques.
- Applying different methods for Integrity of the secret message.
- Applying an attacks in stego file before extraction, to check if that message is altred or not.

1.7. Thesis outline

The research report will consist of the following chapters:

Chapter One: Introduction.

The first chapter of this research includes a research background and demonstrates the problem statement, research questions and objectives

Chapter Two: Literature Review.

This chapter reviews some of the related works concerning the development of steganography methods

Chapter Three: Research Methodology.

This chapter explores the description and implementation of the research which mainly lie in implementing the audio steganography method using Bi-LSB embedding and secret message integrity validation

Chapter Four: Result and Discussion.

This chapter introduces a detailed discussion of the study results aided with the required diagrams and graphs.

Chapter Five: Conclusion and Future Works.

This chapter concludes the whole work that has been introduced and summarizes the obtained results

Chapter Two

The Theoretical Literature and Related Studies

Chapter Two: The Theoretical Literature and Related Studies

2.1 Introduction

The information revolution is the key technology in which the information has been gathered, processed and distributed as an interface between users, and many of the offices in this world. The development of communication makes the source of information to be more valuable and content with active speed like the International Network (Internet). The security issue is the main requirement for every system or protocol, which deals with information. To keep something secret, two basic ideas can be used:

Rules can be used to change the object to a form that is unrecognizable in such a way that original object is formed and can only be recognized by people who know these rules. This is referred in Greek term **Cryptography**, meaning secret writing (Polpitiya & Khan, 2001). The object can as well be hidden in a place that is secret where nobody will find it apart from the people familiar about the secret sender. The concerned methodology of this kind of information security is referred to as **Steganography,** Greek term implying covered writing (Polpitiya & Khan, 2001).

2.2 Background

Mandal and Sengupta (2011) proposed a (STMDF) with a minimum fidelity deviation of a data embedding technique where 2 bit/byte had been changed by selecting the location randomly between least significant bits (LSB) up to most significant bit (MSB). Also this method optimized pixel intensity value after operation of embedding by comparing this value with value of original pixel. The technique of STMDF had been compared with existing H.C. Wu method and Wu-Tsai's method, this show that the suggested method has better performance in parts of PSNR and stego images fidelity.

A steganography with a given distortion criteria is called combinatorial steganography by Galand and Kabatiansky (2009), it is equivalent to Hamming spaces coverings or to what called codes of centered correcting of error. Based on whether if an opponent is active or passive. A creation of codes of centered correcting of error depends on algebraic geometry and a Reed-Solomon code is suggested.

He and Luo (2008) found that the most of techniques of hiding data in digital video use (I) structure to implant the confidential information so (P) and (B) structure capacity is wasted. So that they analyze the algorithm of data hiding at first by using phase angle dissimilarity of the vector of motion, on this base a new algorithm of steganography depends on motion vector phase was suggested. The algorithm uses single motion vector phase to insert the confidential data in (P) or (B) structure. In order to enhance the efficiency of embedding they used the method of matrix encoding to obtain a better transaction between the hidden data amounts and the modification rate of the motion vectors.

The system of multimedia surveillance system purposed to provide safety and security of public in a monitored space by Rahman, Hossain, Mouftah, El Saddik and Okamoto (2010). Though, because of the surveillance nature, information of privacy sensitive, like gait, face, and other physical factors depend on the captured medium from several sensors, could be exposed without the worry of the people. So, it is wanted to have such mechanism which can conceal information of privacy sensitive as much as probable, so far supporting efficient supervision tasks.

Ramkumar, Akansu, and Alatan (1999) presented an information theoretic approximation to achieve an approximate of bits number that can be concealed in sequences of compressed image. They showed how adding of the signal of message in an appropriate transform field rather than the domain of special can considerably increase the capacity of data hiding. Also they compared the achievable capacities of data hiding with several block transform and show that the transform selection could based on the needed robustness. Where, it is better to select transform that has a good power compaction characteristic such as (Wavelet, DCT etc). When the needed robustness is low, poorer power compaction characteristic of the transformer (such as Hartley or Hadamard transform) are preferable selection for requirement of higher robustness.

Yan and Ping (2009) implemented and constructed a novel algorithm in Steganography depend on spatial domain to conceal information that have a large amount into BMP colored image. It uses the secret data arrangement to cover distortion that is defined fixed Least Significant Bits (LSBs) substitution techniques. The technique doesn't need referencing of the original image when the hided data are extracted from the stego image. So that their experimental results show the suggested technique can accomplish high capacity with good quality of image. They also found by an improvement and continued research in design of algorithms, steganography of neural based

can be taken such a serious technique to conceal information so that the current work shows that it was more capable than the most familiar algorithm such as Optimal Pixel Adjustment Process (OPAP).

A revolutionary scheme of steganographic was suggested by Kumar, Sasidharan, Karthikha, Sherly, and Avani (2010) for compression of images, genetic and wavelets programming was used. This method of image compression was built by depending on the Discret Wavelet Packet Transform (DWPT) to make the image shorter, and to present better quality degrees at enhanced compression ratios. Evolutions in genetic programs were allowed by quantizing of images in the domain of wavelet, this is achieved without affecting in image quality terms. At this approach diamond encoding is a novel model in data hiding. This encoding (diamond) offers a simple way to create a very appreciable result than other results produced by other embedding techniques. This technique has ability of concealing more confidential data while maintaining the quality of stego image degradation imperceptible. The restoration of image was also done by utilizing "morphological neural network" to provide an enhanced performance. This technique not only maintains stego image of high quality but also hide bigger data amount into cover image for confidential communication. An act of this system shows to be better than the various used systems in parts of Peal Signal to Noise Ratio (PSNR).

2.3 MP3 Audio Structure

2.3.1 Introduction

One of the methods used to compress audio to digital form is MP3; it tries to consume the minimum space possible, and at the same time keeps the quality of the audio with as good as possible. This method in this area is one of the best achievements (Supurovic P, 1998).

"Moving Picture Expert Group (MPEG)" created MP3; it was formed in January 1988 with an aim of creating standards that are applicable internationally, for coded representation of audio, moving pictures and combination of both. To operate the group is under joint direction of "Electro-Technical Commission (IEC) and "International Standard Organization (ISO)". In the video world, initial the aim was come up with a standard that allows playback of the audio to video material from a device with the capacity to deliver at 1.5 million bits/second; in simple terms, the

conventional CD-ROM. The year 1992 marks the period when MP3 was released as a part of MPEG model. The term "MP3" has been used to refer to layer 3 of MPEG-1 mode of compression.

Layer-3 represents the highest complexity mod that has been optimized to ensure that at low bit rate (about 128 kbit/s) there is highest quality (Brandenburg, 1999). By having inherent ability that keeps down the file size without having compromise in quality of audibility, it has become very successful. MP3 has caused online revolution being first audio format, which made the sharing process of audio files on the Internet be feasible. In the past, high quality files required many hours to download. With MP3, time to download has been cut to a tenth without any identifiable change in the quality of "sound" of the audio. The MPEG algorithm achieves perceptually lossless or transparent compression lossless, though the compression of MP3 is considered loss because after compression, some data cannot be recovered. After doing some test, there was a conclusion that expert listeners were not able to distinguish between original and codes audio clips even when using a six to one ratio of compression (Pan, 1995).

When one looks for the reasons why MP3 and no other technology of compression have emerged as main Internet audio delivery tool, the following is realized:

1. **Open standard:** this is the MPEG-1 layer 3. The specifications are made available to anyone who is interested to implement the standards, because no single company owns the standard.
2. **Availability of encoder and decoders:** This is Drove by demand to have professional use, many MP3 decoders and encoders are readily available for their purposeful use. This accelerates and simplifies the MP3 technology adoption.
3. **Supporting Technologies:** Supporting technologies are the main enabling technology of audio compressions. There is widespread Computer sound and computers are getting fast enough to perform functions such as software audio encoding and even decoding. Fast internet access for businesses, universities and the spread of CD-Audio writers and CD-ROM has had great contribution to the ease in distributing MP3 format music via computers. Briefly, MPEG-1/2/3 layer 3 is the right technology that is available at the right and required time (Brandenburg, 1999).

2.3.2 MP3 File Structures

Composition of MP3 files is done with short data frames, and headers being padded. Meta-data tags can also be contained in MP3. The tags are two kinds, the ID3v1 that is the older format, and is post-pended at end of file. The tag always has a length of 128 bytes and has seven fields; they specify the name of the artist, album, song title, and genre and other specifications… Because of lack of flexibility and its static size, the tag type is slowly being replaced by the ID3v2 standard that are more advanced (Supurovic, 1998).

ID3v2 tags are more flexible and the newest and pre-pended to file. There are almost flexible structure similar to the structure of the files of MP3 itself. ID3v2 tags are composed of their own frames; the frames store various bits of information. This consists of the standard character strings like the name of the artist, the title of the song or more advanced information on how the encoding of the file was done. ID3v2 tags are useful in providing hints to the decoder. For instance, ID3v2 tags stores the equalization curves. ID3v2 tags have no set limits so in theory they can grow indefinitely. Tag types can be included in MP3 files in circulation. Either a stenographer has to be prepared to deal with tags of information present after or before the audio data stream as there is no clear preference. It is logical however, to assume that ID3v1 tags will be increasingly rare in future.

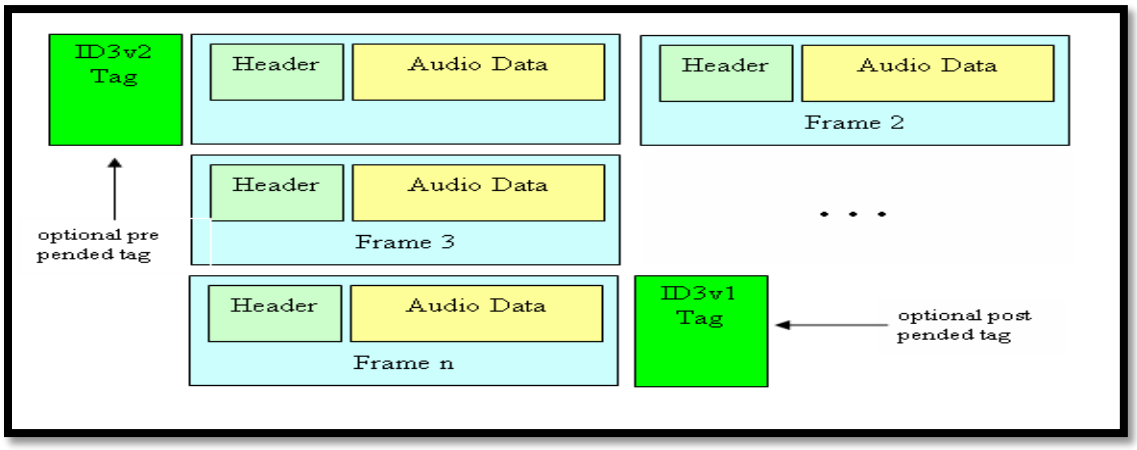


Figure 2.1 Conceptual model of an MP3 file (Maciak, Ponniah & Sharma, 2008).

The ID3v2 tags would be a target that is interesting for embedding information because of their expandability. However, there is no guarantee that they will be present in every MP3 file.

Therefore, the best approach is that the data be embedded into the data frames. It would be appropriate to take a closer look at data frame before stenographic methodology is discussed.

2.3.3 MP3 Frame Headers

The sized of the frame are not obvious and it is therefore necessary to be able to identify the start of a frame and the ends. This is not difficult because it would first appear. There is a frame header pre-pended to each frame. All the headers are identical in Content and structure. Thus, MP3 header identification is just a matter of matching the pattern.

Synchronization (Sync) block are 12-bit block in each header, and they are the ones that start as shown in figure (2.2). The Sync represents a string of one's, whose role is to help the decoder to home in on a header. Hence, to identify a frame one simply requires to detect a 12 consecutive bits initialized to be 1.

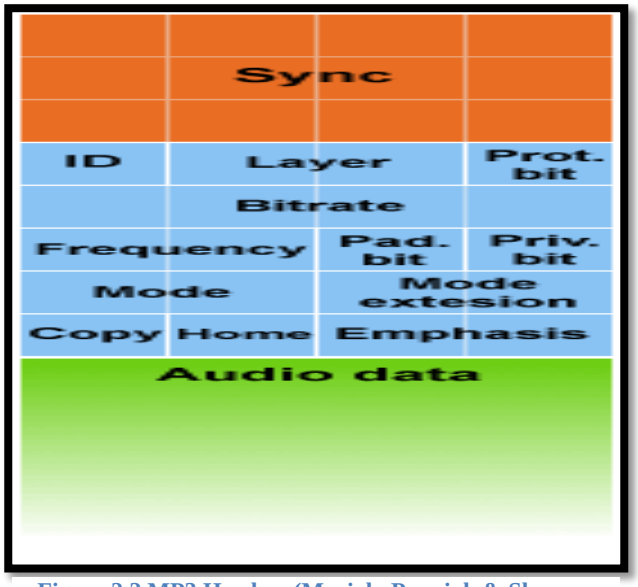


Figure 2.2 MP3 Headers (Maciak, Ponniah & Sharma, 2008).

However, the pattern is not necessarily special to a header. Indeed, the pattern can be easily identified in any data block that is longer. There are other few checks which are performed to check a 4-byte data block as a header:

- The Bit-rate field is not 0000 or 1111.
- The Frequency field is not 11.

- The Layer field is not 00.

A 4-byte block that does not violate the above conditions and starts with the Sync is probably a header (id3, 2014).

Fig (2.3) indicates another view of the MP3 header where characters are used to mark the the fields;

Table (2.1) has a brief explanations of each field.

Figure 2.3 MP3 Headers Fields (Maciak, Ponniah & Sharma, 2008).

Table 2.1 Fields of MP3 Headers (Maciak, Ponniah & Sharma, 2008)

Location	Use	Size
A	The Frame sync	12
B	The Audio Version, MPEG (MPEG-1, 2…)	2
C	The MPEG layer noted as (Layer I, II, III, etc.)	2
D	Protection (checksum following header if on)	1
E	Bitrate index (In order to specify MPEG layer and version, lookup the table used)	4
F	Rate frequency Sampling (44.1kHz…, lookup table determines this)	2
G	Padding bit (off or on, unfilled frames compensator)	1
H	Private bit (off or on, allows for specific triggers applications)	1
I	Channel mode (dual channel, joint stereo, single channel, stereo)	2
J	Mode extension (To conjoin channel data, it is used only with joint sterio)	2
K	Copyright (off or on)	1
L	Original (on if original and if off, copy of the original)	1
M	Emphasis (original recording emphasis; largely obsolete)	2

2.4 Steganography

Steganography is an ancient art that has been reborn in recent years; this art hides the idea that there is communication happening (Provos, 2001). Here the aim is to have a communication channel that is convert between two parties, the two channel existence is to be hidden to a possible attacker (Kivanc, 2002).

Steganography basically, takes single piece of information and then hides the information within another computer file (sounds recordings, images, and texts) containing insignificant or unused areas of data. It takes the advantage of the areas, where it replaces them with information. These files can later be transported or sent without anyone getting to know what really is inside it (Katzenbeisser & Petitcolas, 2000).

Precisely, Steganography in today's connected society has an increasingly important role, as digital copyright protections and covert communications demands continue to rise (Matthews, 2003).

Steganography is concerned with methods of ensuring that secret message is embedded (which can be serial number or a covert communication or a copyright mark) in a cover message (like an audio recording or a video film, or even computer code). Parameterization of the embedding is done by a key; without the knowledge of existence of this key. It is hard for a third party to remove or detect the embedded material, when cover object has material embedded in it, this is called stego object. For instance, we might embed a text in a cover image or a mark in a cover text to give or giving a stego-image or stego text, and so on.

In a stego system that is perfect, the stego image is not being distinguishable from the original cover. A cover can easily detect and then possibly reconstruct the message. In order to avoid accidental reuse, both receiver and sender should destroy all covers they already have used for transfer of information (Johnson, Duricn & Jajodia, 2001).

The Basic Steganography System Model

The basic steganography system model is indicated in the diagram below (Cacciaguerra & Ferretti, 2001).

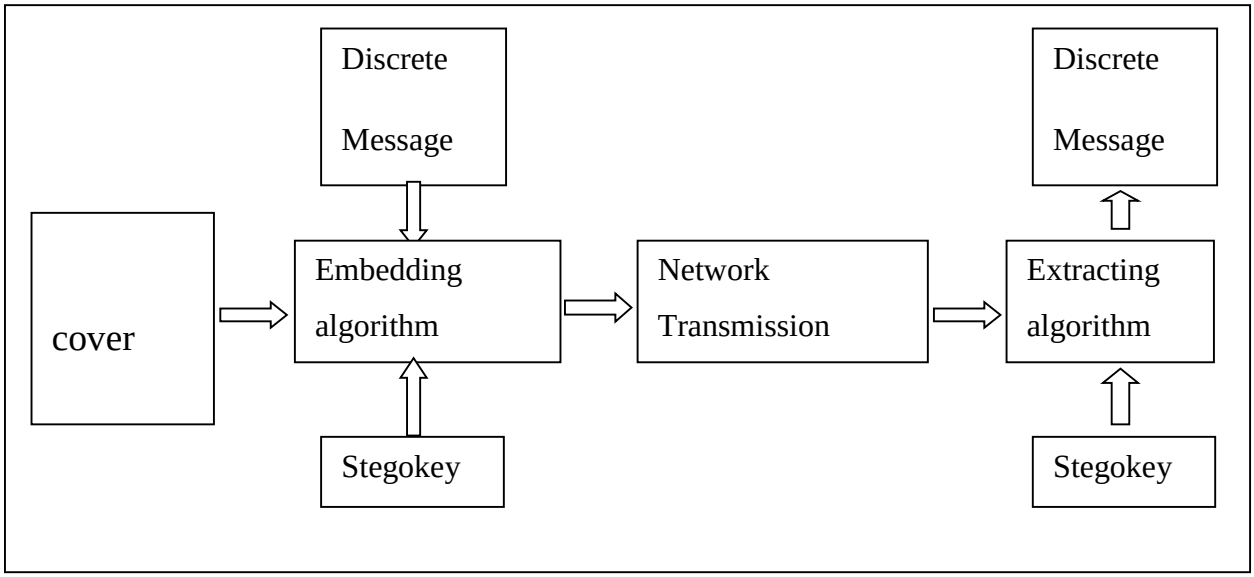


Figure 2.4 The basic steganography Model

Each method of hiding data consists of:

- Embedding algorithm;
- Extracting algorithm;

The use of embedding algorithm is to hide secret messages in a cover/carrier) where as the key protects the embedding process so that only those passing the secret key word can have access to the hidden message.

The application of extracting algorithm is done to a modified returns and carrier of the hidden secret message.

2.4.1 Properties of Steganography

Each technique of data hiding must have particular properties, dictated by the application intended. The most crucial properties of data hiding schemes are capacity, robustness, and undetectability, several definitions for the concepts are described below:

A. Robustness

The embedded data needs to be immune to modification, ranging from intelligent and intentional attempt for removal to any manipulations anticipated (Bender, Gruhl, Morimoto & Lu, 1996). Instances are nonlinear and linear filters (sharpening, blurring, median filtering), recoloring, loose compression, resampling, scaling, noise adding, rotation, cropping, printing / scanning/ copying, Analog to Digital (A/D) conversion and Digital to Analog (D/A) conversions. Robustness does not have attacks on embedding schemes, which are based knowledge of the availability of the detector or on function embedding algorithm or, robustness implying resistance to "blind", or common image operations or non-targeted modifications (Fridrich, 1998).

B. Capacity

Capacity is the amount of information, which can be hidden relative to the cover size (Lin, Delp & Edward, 1998). There are Trade-offs between the degree of host signal degradation and the quantity of the embedded data. A data-hiding method can be able to operate with either high resistance to modification or high-embedded data rate, but not on both. As one goes high, the other must be decrease (Bender, Gruhl, Morimoto & Lu, 1996). A bitplane tool encompasses methods, which apply LSB insertion as well as noise manipulation. The approaches are most common in steganography and are easy to apply in audio and image. With little, if any, perceptible impact to carrier, a surprising amount of information can be hidden (Katzenbeisser & Petitcolas, 2000). However, the approaches are vulnerable to small changes resulting from lossy compression or image processing (Johnson, Duricn & Jajodia, 2001).

C. Undetectability

The property is required in order to secure covert communication. For instance, if a steganography method is using the noise component of digital images to embed a secret message, it should do so while not making statistically significant changes to the noise in the carrier. The undetectability concept is inherently tied to statistical model of the source of image. If an attacker has a model of the source that is more detailed, he can be able to detect the hidden message presence, note that the ability to detect the hidden message presence does not automatically mean the ability to be able to read the message (Fridrich, 1998).

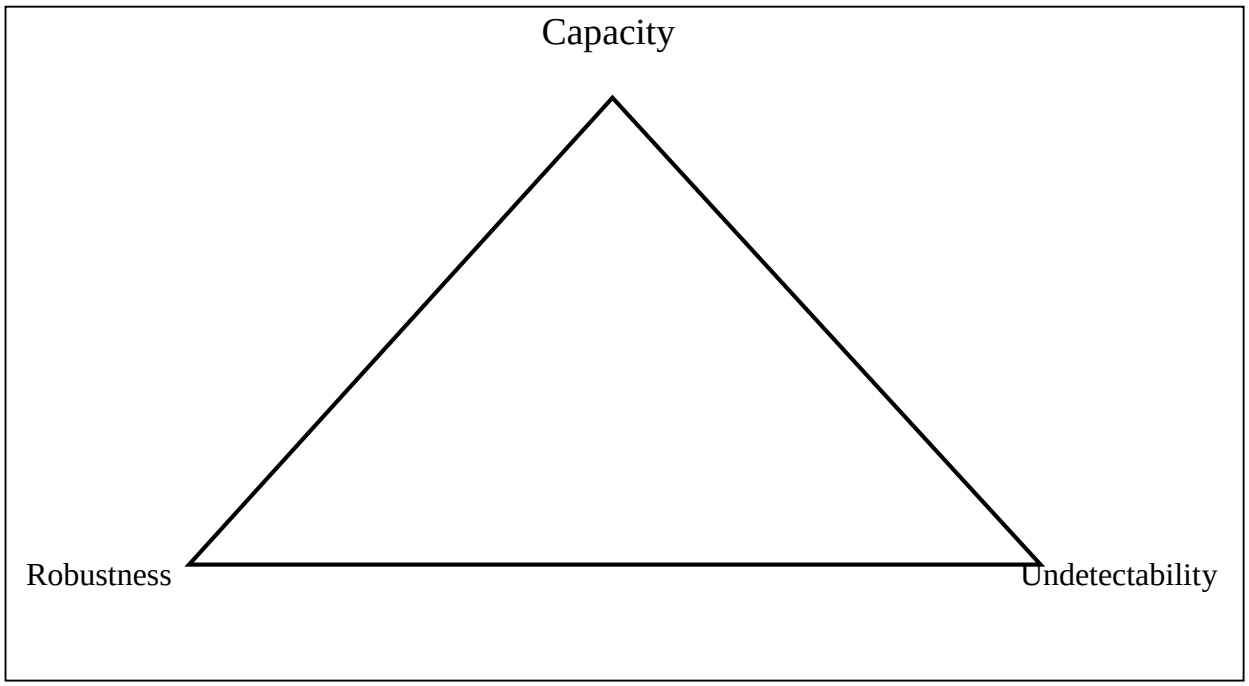


Figure 2.5 Trade-off among undetectability, capacity and robustness

2.4.2 Types of Steganography

There are three types that steganography can be divided into:

1. Pure Steganography

When the system does not need to change secret info it is called a "Pure Steganography", such as stego-key. The process can be represented as the following: mapping E: C×M→ C, Where C represents all possible covers, and M is all the possible messages. While the formula D: C → M is used to extract any secret message exists out of the cover, of course $|C| \geq |M|$ is necessary condition. Algorithms here are not public, only senders or receivers can access extraction process and embedding (Katzenbeisser S., Peticotas F., 2000).

This type provides the least security, because communication using it leads senders and receivers to only depend on assumptions that other parties don't care about secret messages sent by them, like Internet messages (Dunbar B., 2002).

2. Private (Secret) Key Steganography

This type it is needed to change the secret key through communication. The message is embedded into a cover message using a secret key (stego-key). People who know the secret key can get the original message that is inside the cover message and read it. While in pure steganography secret key is more liable to be intercepted, in private steganography, there is a secret channel and stego key exchanges through it. In case private steganography messages intercepted, only people who know the keys can see the message and extract it (Dunbar B., 2002).

3. Public Key Steganography

This type relies on the means of public cryotograghy keys, which are systems that both public and secret key to secure communication channels between people who want to communicate privately. At the sending side, the message is coded by public key, but only the secret key that mathematically corresponds with this public code can decode the message.

Advantages of Public key steganography are clearly found in the more effective implementation public key cryptography, it has different stages of security, that prevents other parties from joining the communication without suspicion of the steganography used, and they must also know how to crack the algorithm used (Dunbar B., 2002).

2.4.3 Steganography Medium

Revolution in computer and internet fields have given steganography a specific importance. Many changes happened to old native carriers due to the use of steganography in computer based technologies. These carriers could be relative to many kinds of data, they may be for text, disks, audio, images, sound, network traffic, or other digital data transmission forms (Johnson N.F., Duricn Z., Jajodia S., 2001). Data hiding techniques are illustrated below:

1. Hiding in Text

It is may be needed to hide some of or the whole text written in documents. Web browsers ignore some formatting like additional line breakers, spaces and tabs, so that HTML files are used to transfer date. These extra line breakers, tabs and so on could not be observed before

reaching the web page source. Various ways could be used to hide text information like (line-shift coding, word-shift coding) (Johnson N.F., Duricn Z., Jajodia S., 2001)**.**

2. Hiding in Image

Most probable images are wanted to be secretly sent. For HVS attributes, many points should be taken into consideration in images hiding, such as luminance effects, edge masking, and contrast. More than one part out of 30 for random patterns may exhibit changes, hence HVS does not have high sensitivity to luminance changes. Another important property for HVS, is the low sensitivity to small spatial frequency, like changes in image brightness. Images are non-causal, so that pixels or blocks could be accessed easily by hiding techniques (Bender W., Gruhl D., Morimoto N., Lu A, 1996).

3. Hiding in Video

Mostly, hiding in videos uses the techniques used for sound and image hiding, because already video forms of images and sounds. Video consists of moving images with sounds, actually this is an advantage, any small distortion will not be noticed by user due to this continuous amount of data (Karen R., 2001).

4. Hiding in Audio

Audio hiding is particularly challenging due to its large frequency range, which is more than 1000 to 1, and also a power more than 1000,000 to 1. Audio signals are also sensitive to random noise. Noise can be detected if it is in the range of one to million in sound files (Bender W., Gruhl D., Morimoto N., Lu A, 1996).

In audio hiding, user should take advantage of of HAS weakness, but should also take care from its high sensitivity (Sellars D., 2003).

"The Magic Triangle" is the name of the Inaudibility strength and capability of the algorithm used in hiding, they are three paradoxical requirements for audio hiding. But HAS is more reliable, loud sounds can cover quiet sounds, due to the limited differential range of HAS. HAS perceives relative phase only, but does not perceive absolute phase. Finally, there are some environmental distortions so common as to be ignored by the listener in most cases. These

"holes" can be exploited by data hiding techniques. One of the drawbacks could be eliminated by data hiding is the ability to ignore environmental disturbances.

To understand the techniques of audio file information hiding, audio signal transmission should be understood first. Also, it is needed to understand the audio Digital representation, and the medium of transmission. The contents of digital representation of an audio file are:

1. Sampling Quantization.
2. Sampling Rate.

Quantization is a linear 16-bit, or 8-bit logarithm. The range of the sampling rate is between 8 kHz to 44.1 kHz (CD quality. Nyquist theorem shows that "the maximum usable sound frequency is bounded by sampling-rate/2". These elements are important in hiding techniques (Sellars D., 2003).

This document takes the secret media MP3 form, and also saved as an MP3.

2.4.4 Methods of Audio Steganography

The best fitting technique can be chosen after understanding what mediums audio signal transfer through:

1. Amplitude Modification

A popular method for amplitude modification is called LSBs insertion, this method could be implemented in steganography or watermarking (Katzenbeisser S., Peticotas F., 2000). LSBs insertion the errors may occur during digitizing of audio signals.

It is obvious in this method data is encoded into LBSs of audio data. For example, let's take a 16-bit sampled file, data hiding can use the least four significant bits. Nevertheless, distortion may happen to the hidden data, also it may be detected while it is hidden. Hidden data could be alerted by resembling or noise. But changes in the LSB may make audible noise (Sellars D., 2003). At risk of data distortion may happen during copying, compression, or A/D, D/A conversion (Yang Y., 2001). LBSs insertion is a simple technique, but it is rarely used, this makes other techniques more likable.

2. Phase Coding

Another method of hiding audio data is phase coding, it refers the phase of the audio signal to a reference phase, which is representing the data. All phases are adjusted to match the relative phase required.

Phase coding is effective for signals coding in order to adjust noise ratio. When the phases between frequency segments change, an obvious dispersion will appear. Anyhow, unacceptable modifications could be detected, in case which involve small modifications on phase. In this case an inaudible coding could be done.

Usually, listeners can not notice changes on audio when the phase coding done with smooth phase shifts, this makes phase coding one of the most effective methods in terms of noise ratio concerns(Yang Y., 2001).

Phase coding has a very low supporting data rate. Actually, this is a major drawback of this method (Polpitiya A.D., Khan W.J., 2001).

3. Spread Spectrum Coding

Another method of hiding information in audio signals is " Spread Spectrum Coding" SSC. Conservation of power and bandwidth is a need in transferring audio data, so the communication channel tends to make frequency region of audio signals as narrow as possible.

In (Bender W., Gruhl D., Morimoto N., Lu A, 1996), a method discussed naming "Direct Sequence Spread Spectrum" (DSSS) encoding. In this method a chip is used to multiply an audio signal in order to spread it, a greater length sequence is modulated at a certain rate. The sampling rate can be chosen as the rate of the chip, because the host signals are already discrete signals. One problem in using DSSS, is how to determine the right the beginning and the end of the chip phase locking quanta, this should be taken into consideration by the discrete signals nature. However, a greater rate of the chip and so a greater rate of associated data is available. Actually, without this, a many algorithms of signal locking could be used, but they are expensive. Unluckily, DSSS results in extra random noise to audio signal, e coding does not (Bender W., Gruhl D., Morimoto N., Lu A, 1996).

A “Frequency Hopped Spread Spectrum” (FHSS) encoding method is also uses spread spectrum, in this technique the carrier signal frequency hops quickly from certain frequency to another when it is alerted in a certain way (Katzenbeisser S., Peticotas F., 2000). The novel audio signal is split into small parts, each part is carried by a special frequency associated with it (Chen P., Hoffman D., 2002).

Using spread spectrum coding is mainly useful due to its resistance to variations. It is difficult to adjust the embedded data without making observable destruction for the cover data. So, the data is spread through the cover information.

4. Echo Data Hiding

In this coding method, an audio signal is considered as a host signal to embed the data in, by providing an echo. Data hiding is done by adjusting three echo parameters: decay rate, initial amplitude, and offset (delay). When the offset between the novel signal and the echo decreases, the two signals combine. In certain point, human can not observe any difference between these two signals. Echo here is introduced as an additional resonance (Bender W., Gruhl D., Morimoto N., Lu A, 1996).

Data could be hidden in the audio file by using dissimilar time delays that separate the novel and echo signals from each other. The novel signal may be divided into several small parts, in order to allow embedding more than one bit, each part of this signal could be echoed to a certain bit. All independent encoded segments are existing in the last cover data (Katzenbeisser S., Peticotas F., 2000).

This technique “Echo Data Hiding”, works specially good to sound files that do not contain added degradation, like the case where there is a time for silence in the file, or when there is losses in encoding or some noise in the signal (Sellars D., 2003).

2.5 Integrity Methods

1. Hash function method

It is a typical integrity method that used widely in various protocols and applications. It maps the strings of various lengths to short constant sizes (Sivathanu.et al, 2005). It has an essential role in the current cryptography. In practice, hash functions depend on taking a message as an input and then offer a specific output that is related to the hashes of that message. The main idea concerning hash functions is that the hashes act as compact delegate image, which known as imprints, digests or digital fingerprints of the input string. Those functions are widely utilized in data integrity in combination with models of digital signature since messages are usually hashed first, where then hashes are signed instead of the original message. In practice, hashes of messages must be uniquely specialized with one input in reality. Generally, hash functions must have of those properties; simplicity to be computed and compression. (Menezes.et al, 1996)

One of the main hash functions classes is the Message Authentication Codes (MACs). It permits recognizing messages using symmetric methods. This technique takes two main distinct inputs; message and secret key to offers an output with steady size with the purpose to offer data integrity, symmetric authentication of data and recognition in symmetric-key models. Another type of hash functions is the Modification Detection Codes (MDCs), which offer hashes of messages. The main purpose of the MDCs is to simplify, in combination with further mechanisms, the integrity of data for various applications. Those codes are subclasses of the un-keyed hash functions as well as they can be classified into two classes; one-way hash functions (OWHFs), where finding a certain input that hashes for a known hash is not an easy issue and collision resistant hash functions (CRHFs), where finding two inputs with the same hash is not an easy issue. (Menezes.et al, 1996)

Another type of hash functions is the Modification Detection 5 (MD5), which offers 16 byte hash value. This function is broadly utilized in cryptographic applications and verification of data integrity. It depends on processing a message with variable length into a output with a fixed length. (Menezes.et al, 1996)

2. Checksum

Checksum is a one of the main methods that used to carry out integrity checks. It can be calculated for disk data as well as it can be saved steadily. It depends on comparing the stored calculated values with the newly ones for each data read. This method is mainly produced with the use of the hash function method. The checksum method assists in the detection of variations in the integrity. However, it could not offer any help in the recovery of data due to the mismatch among the saved and computed checksums values, where this means that one of them is adjusted without offering any information concerning the legitimate one. The saved checksums can be corrupted or modified. Another reason for the recovery problem of the checksum is that it is mainly calculated with the use of one-way hash function, where data then could not be reconstructed to offer values of checksum. (Sivathanu.et al, 2005)

Checksums are used widely to reduce duplicates in data objects, since those duplicate objects have the same checksum value. Those objects can be recognized based on using rationally collision-resistant checksum model based on comparing the checksums of those objects. Another use for the checksums is in the indexing of data. Although the collision-resistant checksums are bigger than traditional integers, they can assist in offering double functionality with minimum costs. Checksums can be used also to name handles, which provides a simple method to receive the related checksums to block, where this in turn helps in enhancing the integrity checking performance. (Sivathanu.et al, 2005)

2.6 Related Studies

Few researches used MP3 as a cover in steganography. Many researchers worked on wave steganography. We will now explain and discuss the current MP3 steganography methods:

1. Embedding in header frame

a. Unused Header Bit Stuffing (UHBS):

Frame headers of the MP3 are made up of fields like the private bit, copyright bit, original bit, and emphasis bit. However, their use is commonly omitted in a number of MP3 players. Such fields are the important aspect of frame, which aids the interpretation of information that is concealed in audio signal. They can be properly applied to embed undisclosed massage where they

replace the undisclosed massage bit stream through the bits in the field. But, if in the process of replacing bit stream with bits in the field there is a fuilure the actual content of the secret message received within the frame is lost and this makes signal recovery to be more challenging (Maciak, Ponniah & Sharma, 2008). The work in Cacciaguerra & Ferretti (2001) highlighted the probability that audio steganography could achieve capacity that is good and low robustness by using of 4 bits in each header frame of the audio signal to the embeded secret messages.

b. Padding Byte Stuffing (PBS):

According to Zaturenskiy, (2009), stuffing of padding byte was recently established as being one of steganography techniques. Its approach is relatively straightforward in terms of implementation. It represents regular and fine storage capability and contains the ability to program 1 byte of information for any frame so long as there is accessibility of padding bytes. MP3 file is a given example of the medium of material that can well utilize the method of padding byte stuffing because it can allow for hundreds of frames in a secret message, specifically when the filling bytes are not able to take any audio information

c. Embedding Before All Frames (BAF):

Atoum, Suleiman, Rababaa, Ibrahim & Ahmed (2011) developed BAF , their approach embeds into MP3 file the embeds text files. The text file is encrypted by use of RSA algorithm in order to increase the undisclosed secret message security. Encrypted information is filled in the first frame. This process is done time and again sequentially until the frame headers are filled. Approximately 15 KB is used when encryption algorithm is utilized; otherwise, it takes around 30 KB for MP3 file. Even with the chances of the secret message to be sniffed, there are many advantages in using this approach; for example, the method of padding and the unused bit even after the frames must have been filled, provides more encoding capability.

d. Embedding Between Frames (BF):

Atoum, Suleiman, Rababaa, Ibrahim & Ahmed (2011) developed technique of Steganography that embeds between frames (BF). It as well embeds text file to MP3 file like the BAF and information is encrypted in bits format by use of RSA algorithm to provide extra protection for that concealed secret messages. There is a difference between BF and BAF that exists in the way

text files are inserted into the frames. This does not start with the first frame seen; but it selects the frame it chooses. Again, in terms of capacity, compared to BAF, BF uses around d40 MB with encryption algorithm, it however requires 80 MB with original format. Although BF provides a higher capacity for embedding text file, it is still prone to attack. Literature review shows that the embedding information method after compression is hard task since the process of embedding is done after the compression and the text file are located in the location of unused bits and not in audio data. The platform provided by this technique is prone to attack as a result of the content of the sent secret message which can easily be deciphered by third party sniffing by use of the communication link. It also provides limited capacity for hiding the message that is secret. However, the problem with capacity would be resolved if the LSB technique used 2, 3 and 4 bit exchange in the audio data (8-bit for sample) to insert speech in MP3 file. While addressing the security problem, the use of key as a lock for the concealed secret message is a viable approach that could achieve maximum security for concealed secret messages.

2. Audio data Embedding

a. M4M Method:

Bhattacharyya, Kundu, Chakraborty & Sanyal (2011) proposed a new method for imperceptible audio data hiding for an audio file with wav or mp3 format. This approach is based on the Mod 16 Method (M16M) designed for image, the Mod 4 Method (M4M), along with Number Sequence Generator Algorithm to avoid embedding data in the consecutive indexes of the audio, which would eventually assist prevent distortion in quality of audio. The input messages exist in any digital form, and are often treated as bit streams. The positions of embedding are selected on the basis of some mathematical function that de-ends on the data value of the digital audio stream. Data embedding is performed by mapping every two-bit of the secret message in each of the seed positions based on the remainder of the intensity value while divided by 4. The extraction process starts with the selection of those seed positions required during embedding. On the receiver's side, a different reverse operation is carried out to the extract of the original information.

b. M16M Method:

Bhattacharyya, Kundu & Sanyal (2011) proposed a new imperceptible method of audio data hiding for audio file with mp3 or wav format. The approach is based on the method of Mod 16 (M16M) designed for image, the Mod 16 Method for audio (M16MA), and used alongside with a Number Sequence Generator Algorithm that help avoid embedding data in consecutive indexes of the audio, this would help prevent distortion of audio quality. Input message can exist in any digital form and is commonly treated as a bit stream. The positions of emending are selected based on some mathematical function that de-ends of the data value of the digital audio stream. The embedding of data is performed by mapping every four-bit of the secret message in each of the seed positions based on the remainder of the intensity value when divided by 16. Extraction process starts by selecting those seed positions required during embedding. On the receiver's side, a different reverse operation is carried out to extract the original information.

c. **New Secure Scheme in Audio Steganography (SSAS)**

Atoum, Suleiman, Rababaa, Ibrahim & Ahmed (2011), developed (SSAS) the most significant and public standard for audio compression on the Internet is the "secret digital music MP3 files". To ascertain high rates of compression and shrunk the main file to the smallest possible size, the MP3 uses a devastating technologies for compression. In the process of protecting the "digital media files", there are many algorithms that used in the data hiding, are known the steganographic algorithms, which defined as covered writing. This algorithm has three types; "public key steganography", the "secret key steganography", and "pure steganography". Applications, which are used in the data hiding, there is a requirement for hiding algorithms on the side of the sender, and detecting algorithms and mechanism on the side of the receiver. The authorized persons just can retrieve the hiding data and messages. The data hiding pivotal parameters are; capacity, invisibility, complexity, security, and reliability. The authors discussed a new method on steganographic based on the algorithm of "novel data-embedding", to embed information between the frames of MP3.

The requirements of the proposed method are; program of "k-lite multi-media player" and MP3 file of CBR type. In the evaluation phase, a three scales have been utilized; the time spent in the embedding process and extracting process, the size of the embedded file, and the audio quality,

the results shows that the size of the embedded file will impact on the size of the file cover, because in the embedding process, the “hidden text” inserts between the frames. BF has an ability of embed 40960 KB within stumpy time and preserve the quality of the sound, where the BAF does not overtake 6 KB. The spent time in the extracting process increases the file size (Atoum, Suleiman, Rababaa, Ibrahim & Ahmed, 2011).

3. Developed method for Audio Steganography with the Use of Message Integrity

Atoum, Ibrahim, Sulong, & Zamani (2013) proposed that basically the designer for Steganography technique used a basic model for embedding and extracting the secret messages. Some of them used a key for embedded message in the host signal and also used the same key to extract secret messages.

The basic model of steganography used real information for the secret message that is embedded in the cover media, and the result stego-object contains the cover and the message. However, the probability for attackers to destroy the secret message and to read it is increased. So the attacked secret message can be modified or removed as it is has been discovered by because the attacker.

The proposed method has given two solutions: firstly, to scramble the secret message before it is embedded. Secondly, the verification process is used after extracting the secret message so as to check the secret message that was attacked, deleted and modified.

Additionally, the proposed method with random selections provides a high level of robustness, where these results are more similar to the results of the original cover than the results of the proposed method with sequential selection. Also, the random selection produces higher security as compared to the sequential selection. This is due the fact that the random embedding of the secret message makes it very difficult for the attacker to extract the whole message that is hidden inside the stego-object.

Evaluation of Audio Steganography performance based on adding AWGN for Messages

"Additive white Gaussian noise (AWGN)" is a fundamental model of noise, which mainly utilized in information theories to imitate the impact of random natural processes. The additive property comes from the fact that this model is added for any noise, which may be inherent for information systems, the white property represents its uniform power all thorugh the information system frequency range, while the Gaussian property represents its normal distribution in time domain with zero average value of time domain (Alam, 2009).

Tomar (2012) developed an efficient data hiding method based on LSB technique and Rivest-Shamir-Adleman (RSA) encryption method to encrypt secret data, transform then into binary sequence bits and hide then in cover pixels based on chaning the cover pixels LSBs. The author studied the effect of AWGN on the extracted messages based on transmitting the steganography image over an AWGN channel with simulating the impacts of this noise. This transmission results in introducing the noise in it, thus, stego images can be corrupted via this noise as well as embedded secret data bits can be infeluenced by this noise. Results demonstrated that the presented method offered low Bit Error Rates (BERs) with high Signal to Noise Ratios (SNR) and small number of hidden data bits corrupted through AWGN.

Shahadi and Razali Jidin (2011) studied the resistance of their developed audio steganography algorithm to AWGN based on both "blocks matching" and "wavelet packet transform". Results demonstrated that the proposed algorithm has capacity more than 35% of the size of the input audio signal with adequate SNR. In addition, results proved that the proposed method is resistance to AWGN to around 25 db.

Shahreza and Shalmani (2008) and Santosa and Bao (2005) proposed that the LSB is the mainly utilized method in embedding data in the disceret wavelet transform domain. However, they proposed that although this methods has better stego-signal capacity and quality, it is very sensitive to AWGN, mainly during embedding secret data in high frequencies components that include low power.

Chapter Three

Research Methodology

Chapter Three: Research Methodology

3.1 Introduction

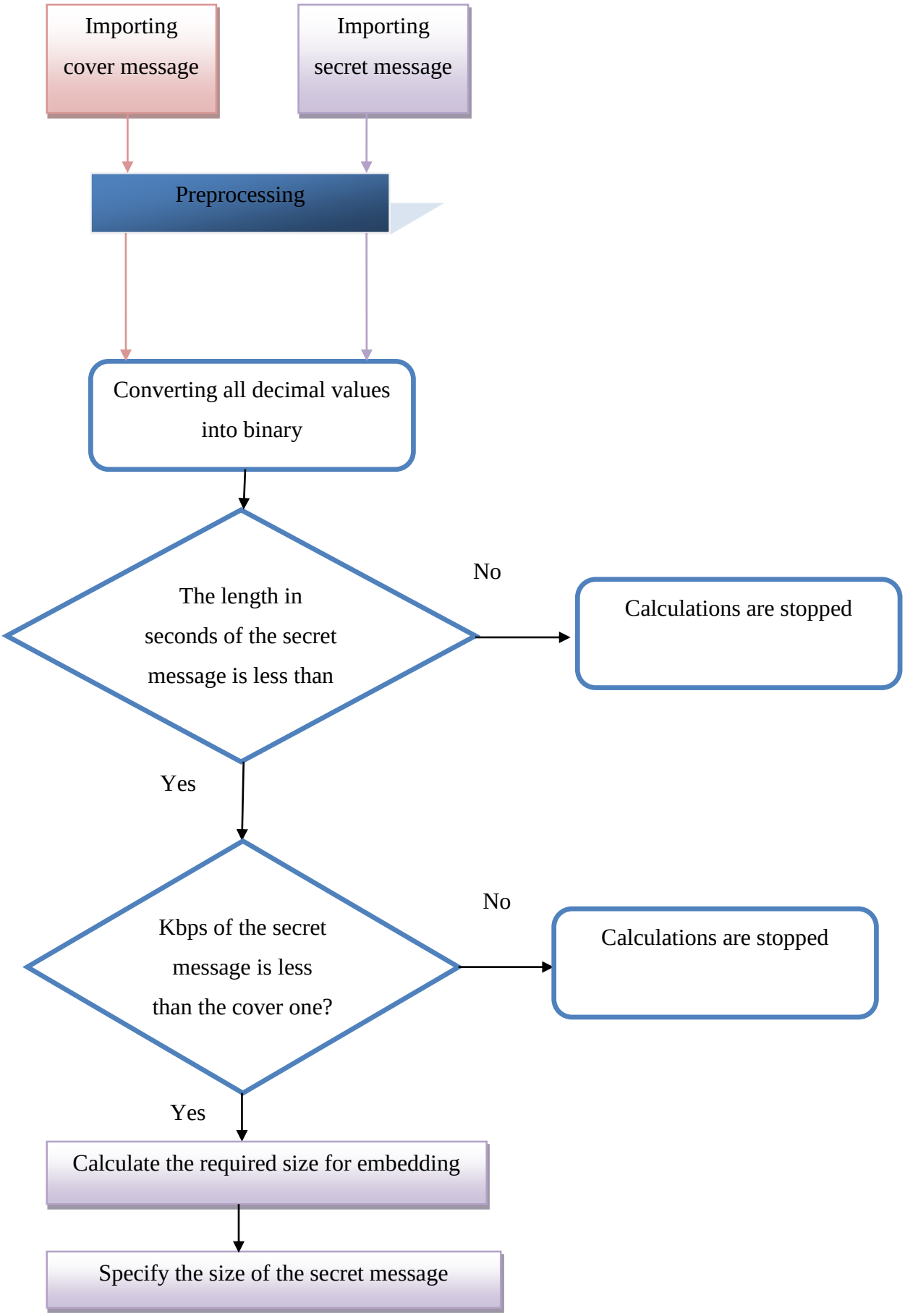
This chapter explores the conducted methodology in this work to offer an advanced Bi Least Significant Bit (Bi-LSB) MP3 audio steganography technique to solve the security problem of traditional LSB techniques and offer an efficient method to hide audio information in more secured way with the use of the MATLAB program version 2014/a. All the conducted procedures and principles in this work to achieve the main purpose of this study are proposed in details.

3.2 Research Framework

In this section, the proposed Bi-LSB MP3 audio steganography technique is explained in details. This method depends on replacing the second right most LSB of each sample in the cover message with a bit of the secret message to enhance the security.

The framework of this research includes three main phases; preprocessing, embedding and extracting and message validation. The preprocessing is applied to enhance the security of messages to be hidden in an MP3 file. The embedding and extracting stage includes designing the proposed algorithm for MP3 files to solve the current security problem of the traditional LSB technique, while the message validation step is applied to develop another method to hide messages to recognize the efficiency of hidden messages in MP3 files.

To evaluate the performance of the presented Bi-LSB method, a AWGN with different variance values is added to the tego file before extracting the secret message, where the PSNR values are then computed and compared with those obtained before adding that noise. The following figure describes the presented method.


Importing cover message
Importing secret message
Preprocessing
Converting all decimal values into binary
The length in seconds of the secret message is less than
No
Calculations are stopped
Yes
Kbps of the secret message is less than the cover one?
No
Calculations are stopped
Yes
Calculate the required size for embedding
Specify the size of the secret message

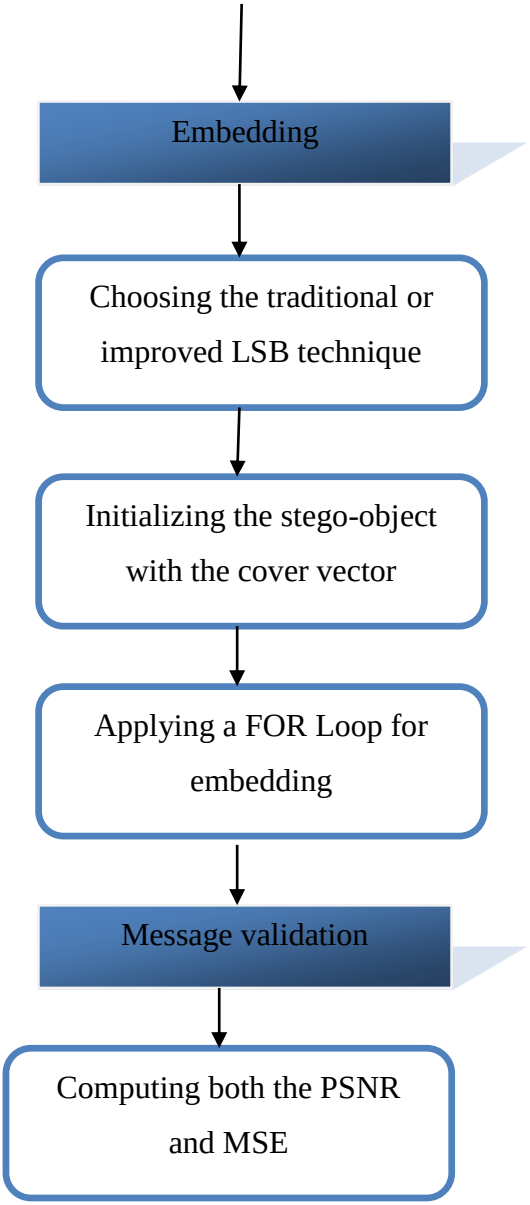


Figure 3. 1 Research framework

3.2.1 Preprocessing Phase

Both the input cover (C) and the Message (M) are MP3 files. In this work, the secret message is validated and preproced based on implementing three methods; checksum, hash function and frequently comparison in order to generate the codebook. The resultant preprocessed C and M files are then used in the second stage.

3.2.2 Embedding and Extracting Phase

This phase includes two main processes; embedding and extracting. In the embedding process, both the preprocessed C and M files are processed to offer a Stego Object (SO). After that, the Bi-LSB method is used to embed the secret message in the cover. In the extracting processes, the SO is then processed to extract the M file.

3.2.3 Message Validation Phase

The traditional steganography methods hide the secret message , but does detect if a message contant is altered during communication. In this work, a contant validation step is added to ensure that the extracted message is exactly the same as the original secret message. This is done by using three contant validation functions (checksum , hash function (MD5) , frequency).

Those methods generate a codebook , which is then used and compared with the hidden message to check if that message is attacked or not.

3.3 Available Datasets

Both the cover and secret messages in this work are MP3 files since they achieve a good data compression when considering the limitations in the human hearing to eliminate information without affecting the sound quality perception. Since there is no efficient number of researchers who used MP3 files as cover messages, there is available standard dataset to apply embedding and extracting algorithms to generate results. This is because most researchers depend on using .wav file, where this results in available standard dataset for it. In this dataset, wav file contains 12 different genres; (Classical, Jazz, Country, and R&B, Rap, Reggae, Pop, Rock, Blues, Hip-hop, Dance and Metal).

The generation of MP3 files depends on using a certain program to convert each genre from wav file to MP3 one, such as the Free Make Audio converter. On the other hand, there are five different bit rate encoding compression methods to compress MP3 files; 320 kbps, 256 kbps, 196 kbps, 128 kbps and 96 kbps, where they differ in their impact on the sound quality. In other words, the increase in the number of bits per sample results in an increase in the quality of sound. The sampling frequency for bit rate for 320, 256 and 192 kbps is 48 KHz, while it is 44.1 KHz for the 128 kbps and 22.050 KHz for the 96 kbps. The following table illustrates the MP3 standard dataset, which generated to be used in this work.

Table 3.1 Cover dataset

Name of genre	Time	Size of file (WAV)	Size under 320kbps	Size under 256kbps	Size under 192kbps	Size under 128kbps	Size under 96kbps
	Minute	(MB)	(MB)	(MB)	(MB)	(MB)	(MB)
Classical	2:54	14.7	6.67	5.33	4	2.66	2
Jazz	3:12	16.2	7.34	5.87	4.4	2.93	2.2
Country	3:42	18.7	8.48	6.78	5.08	3.39	2.54
R&B	3:51	19.4	8.81	7.05	5.29	3.52	2.64
Rap	3:59	20.1	9.14	7.31	5.48	3.65	2.74
Reggae	3:59	20.1	9.14	7.31	5.48	3.65	2.74
Pop	4:00	20.2	9.16	7.33	5.49	3.66	2.75
Rock	4:33	23	10.4	8.35	6.26	4.17	3.13
Blues	4:41	11.8	10.7	8.59	6.44	4.29	3.22
Hip-hop	5:27	27.5	12.4	9.98	7.48	4.99	3.74
Dance	6:12	31.3	14.2	11.3	8.53	5.68	4.26
Metal	6:28	32.6	14.8	11.8	8.88	5.92	4.44

From the table above, it can be concluded that the size of wav file is bigger than that of the MP3 files. Furthermore, the difference in sizes between MP3 file depend on the time of music and the number of bits per sample. The used database is available with Dr. Mohammad Al atoum; moh_atoom1979@yahoo.com

3.4 Designed System

The proposal system is designed using the MATLAB program version 2014/a. The designed system aims to import the audio file in the MATLAB from the MP3 file, prepare both the cover and secret messages, apply the embedding process, validate and evaluate the message and display results.

3.4.1 Importing data from audio files .mat

In this stage, the user is asked to select the cover audio file to be imported to the MATLAB, which must be in the same directory of the MATLAB script. The "mp3read.m" script is used to import the audio file in the MATLAB workspace to generate the following variables:

- Cover : vector containing all the frames of the cover audio file (between -1 and +1)
- FS : sampling rate
- NBITS : number of bits (usually 8 bit).
- OPTS : structure containing different information as shown in the following figure

```
        mpgBitrate: 128000
        mpgVersion: 1
     nAvgBytesPerSec: 16000
      nSamplesPerSec: 44100
           nChannels: 2
         nBlockAlign: 417.9592
      nBitsPerSample: 8
          mpgNFrames: 6697
       mpgCopyright: 'No'
        mpgEmphasis: 'none'
             mpgCRC: 'No'
           mpgLayer: 'III'
        mpgOriginal: 'Yes'
        mpgChanmode: 'joint'
             mpgPad: 'stereo'
  mpgSampsPerFrame: 1152
```

Figure 3.2 OPTS information

3.4.2 Preparing the Cover Message

In this step, the cover file is initially converted from decimal to binary, mp3read and mp3write is used to read cover messages.

3.4.3 Preparing the Secret Message

After preparing the cover audio file, the secret message is then prepared to be embedded in the cover file. In this step, three methods are applied , in each method the codbook is extracted in the preprocessing step, the values of codebook are saved to compared later with extractation step. In this step, the user is asked to select the secret message audio file to be imported to the system using the “uigetfile” and “mp3read” functions, where this file must be in the same directory of the MATLAB script. This in turn creates the following variables:

- Message : vector containing all the frames of the secret message audio file (between -1 and +1)
- FS_Message : sampling rate of the secret message
- NBITS_Message : number of bits of the secret message (usually 8)
- OPTS_Message : structure containing different information of the secret message

After that, the secret message audio values are converted then into positive values with the use of the abs function to be then converted from decimal to binary values. After that, two checks are applied. The first check is applied to check if the length of the secret message is less than the cover one or not in order to recognize if the audio file is a mono or stereo one with the use of the OPTS structure, particularly the OPTS_Message.fmt.mpgPad. When the length is bigger, a warning message is generated and the calculations are stopped immediately. A single column vector is created in the first case, while a double column matrix is generated in the second one; left and right channel, where the average among them is considered in this condition.

The second check is applied to check if the bit rates; kbps of the secret message is less than the cover oner not, the cover messgae should be suitable to the secret message in size format.A warning message is generated and the calculation are stopped immediately. After that, the user is asked if he/she wants to insert the whole audio file or just a part of it, the available choices are 1 for choosing the whole audio file and 2 for choosing a part of it. In the second case, the initial and

the final instants must be offered in terms of seconds. After that, the secret message is converted from decimal to binary.

3.4.4 Applying the Embedding Process

In this stage, the user is asked to verify the desired type of LSB technique; traditional or improved one. After that, the stego-object is initialized with the cover vector. Then, a Generation Randomly (GR) value, which a random number of the cover message is chosen, where then the secret message in binary code is reshaped in one row in order to be used in the LSB algorithm in an ease way based on determining the number of LSBs and initializing two counters; k and w. A For-loop for message embedding is then generated to start at GR and with using a step of mod(i,100) to embed the secret message in the cover audio. For the traditional technieu, in each one of the cycle iterations, the selected byte is modified using the following logic:

- If the 4-LSB is used, then bits from 2^{nd} to 5^{th} bits are substituted with the first available 4 bits in the secret message
- If the 2-LSB is used, then bits 2^{nd} to 3^{rd} are substituted with the first available2 bits in the secret message
- If the 1-LSB is used, only the 2^{nd} bit is substituted with the first bit available in the secret message

For the improved Bi-LSB technique, in each one of the cycle iterations, bits are hidden as follows:

- Hiding 1 bit in the first byte
- Hiding 2 bits in second byte
- Hiding 2 bits in third byte
- Hiding 1 bit in fourth byte and repeat

After that, the modified byte is converted back from binary to decimal to generate the stego object. In each one of the iterations, the counter k is updated depending on the chosen LSB algorithm.

3.4.5 Applying the Distortion Evaluation of the Message

In this stage, the Peak Signal to Noise Ratio (PSNR) and Mean Square Error (MSE) are calculated. Both represent the two error metrics used for comparing cover quality. The PSNR and MSE can be calculated using the following formulas, respectively:

$$\mathrm{MSE} = \frac{1}{\mathrm{N}} * \sum_{\mathrm{i}=1}^{\mathrm{N}} ((\mathrm{X(i)} - \mathrm{Y(i)})\text{^}2$$

$$\mathrm{PSNR} = 10 \log 10 \ \frac{(\mathrm{MAX})\text{^}2}{\mathrm{MSE}}$$

Where, X is the original cover, Y is the stego-object, N is the size of cover and MAX is the maximum variation in the input cover.

3.4.6 Applying the Integrity Validation message

In this stage, different intgerity methods are applied, were each method, codebook is saved and compared later with extrcation process. The checksum technique offers integrity certification to simplify the recognition of packets, which carry secret data at the end of received message. The hash function represents the mapping of digital data with random size to data. The offered values by this function are known as hash codes, hash sums, has values or hashes. The frequency technique represents counting the number of ones in the message and comparing it with that number in the extracted message.

3.4.7 Applying Attack

In order to evaluate the performance of the developed method, an AWGN with different variance values is added to the stego file before extracting the secret message. This is performed using the MATLAB program version 2014/a.

3.5 Summary of Designed System Stages

Apply the embedding process, validate and evaluate the message and display results.

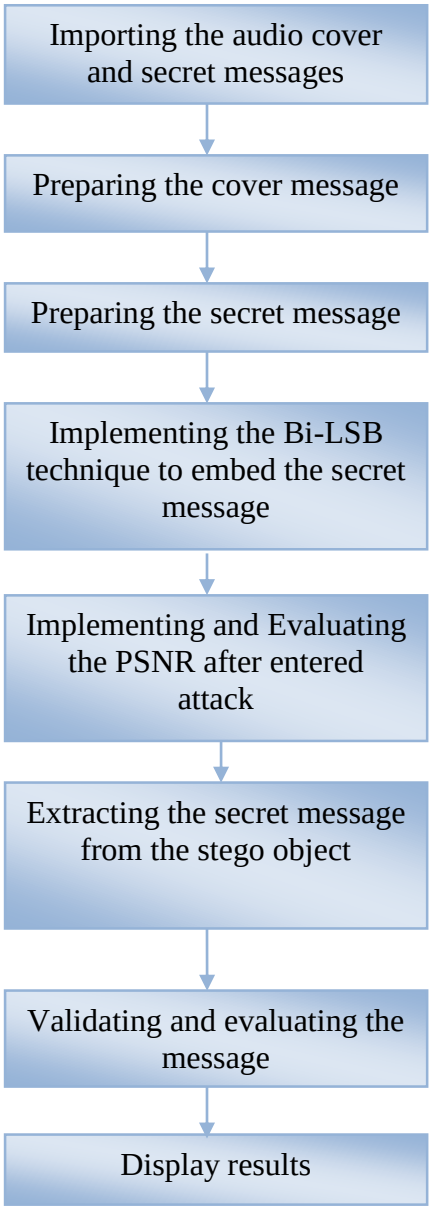


Figure 3. 3 designed system stages

Chapter Four

Results and Discussion

Chapter Four: Results and Discussion

4.1. Introduction

The conducted work in this research aims to design an effective Bi Least Significant Bit (Bi-LSB) MP3 audio steganography technique to find a solution for the security problem of traditional LSB techniques. In addition, it offers an efficient method to hide audio information in more secured way with the use of the MATLAB program, version 2014/a. The main functions of the code are illustarted in Appendix, where the full code is available with the student, you can get it based on contacting him via mail: mms_8921@yahoo.com

In this work, MP3 audio messages are hidden in cover using both the presented Bi-LSB technique and the traditional LSB techniques in order to compare and evaluate the performance of both systems. The audio message is embedded in the cover one using a for-loop as proposed where the selected byte in each cycle iteration is modified using the current Bi-LSB and three traditional LSB techniques; 4-LSB, 2-LSB and 1-LSB. The following chart explores the simulation parts.

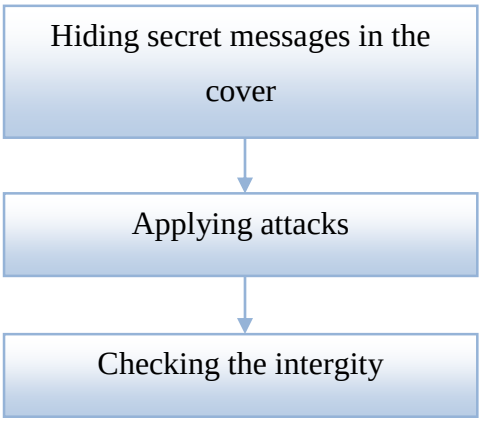


Figure 4.1 Simulation stages

The obtained results of those three parts are illustrated and discussed below in the following sections.

4.2. Embedding phase results

In this part, secret messages are hidden in various cover messages using both the improved and traditional LSB techniques. In this work, two cases are explored for hiding two different secret messages in five cover messages to compare the current technique with the traditional one. Those two cases are explored in the following sub-sections.

4.2.1. Results of Embedding the First Secret Audio Message

The resultant PSNR values after embedding a secret message; "Blues_96_msg1.mp3" with a size around 1MByte , in five different cover messages using the developed Bi-LSB technique in this project and three traditional LSB techniques; 4-LSB, 2-LSB and 1-LSB are shown in the following table and figure.

Table 4.1 PSNR results for methods

Cover messages	Length of cover messages	size of cover messages (MB)	Embedded secret message	4-LSB 2nd to 5th	2-LSB 2nd and 3rd	1-LSB 2nd	Proposed method
Blues_cover1_320.mp3	04.41	10.7	Blues_96_msg1.mp3	44.3862	63.4260	76.2214	95.2852
Classical_cover2_320.mp3	03.12	8.67	Blues_96_msg1.mp3	35.5372	45.8897	57.2895	75.8143
Jazz_cover3_320.mp3	03.12	8.34	Blues_96_msg1.mp3	47.3317	69.7706	83.1974	102.7913
Pop_cover4_320.mp3	04.00	9.16	Blues_96_msg1.mp3	38.7945	50.9703	57.9350	73.4542
R&B_cover5_320.mp3	03.51	8.81	Blues_96_msg1.mp3	46.7376	68.5931	81.9648	101.4737

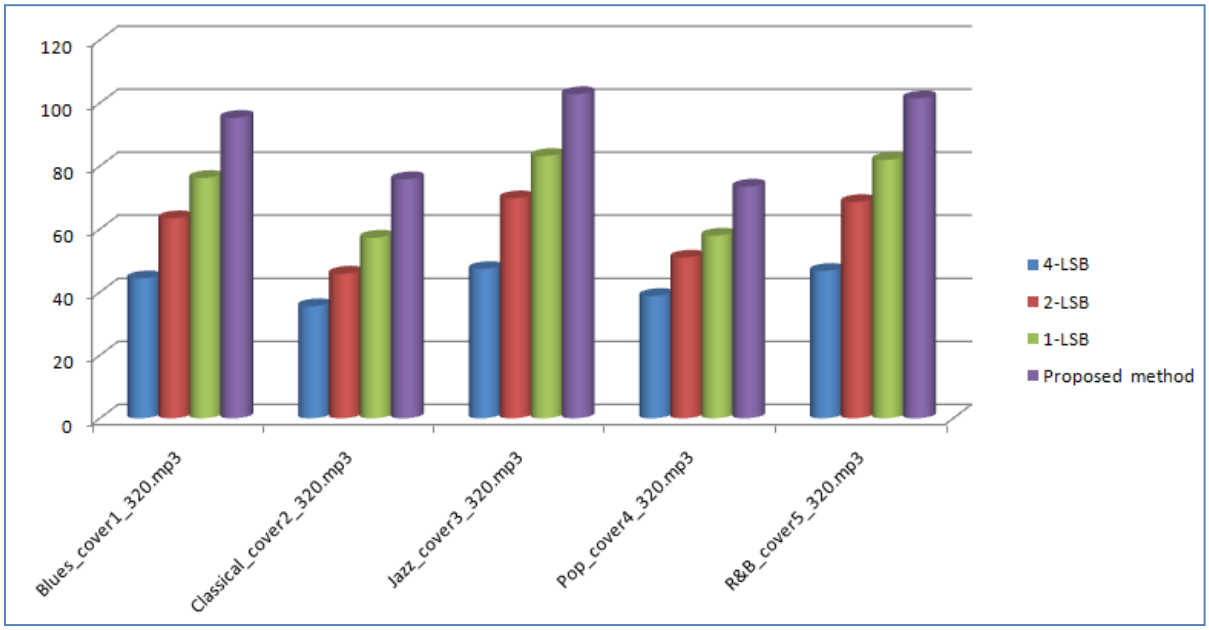


Figure 4.2 PSNR results for methods

It can be clearly seen that the PSNR results of the proposed method are better than those of the traditional LSB for all genre names. As shown above, the Jazz offers the highest PSNR since it is one of the MP3 that includes higher noise than other audio files. Thus, it offers high PSNR. The following table shows the enhancement percentage between the current method and the traditional methods for the five cover messages.

Table 4.2 Enhancement percentage between the current method and the traditional methods for the five cover messages

Cover messages	enhancement percentage between the proposed method and the 4LSB	enhancement percentage between the proposed method and the 2LSB	enhancement percentage between the proposed method and the 1LSB
Blues_cover1_320.mp3	**53.41%**	**33.43%**	**20.00%**
Classical_cover2_320.mp3	**53.12%**	**39.47%**	**24.43%**
Jazz_cover3_320.mp3	**53.95%**	**32.12%**	**19.06%**
Pop_cover4_320.mp3	**47.18%**	**30.60%**	**21.12%**
R&B_cover5_320.mp3	**53.94%**	**32.40%**	**19.22%**

4.2.2. Results of Embedding the Second Secret Audio Message

The obtained PSNR values for embedding a secret message; "Jazz_128_msg2.mp3" with a size around 1MByte, in five different cover messages using the developed Bi-LSB technique and 4-LSB, 2-LSB and 1-LSB traditional techniques are shown in the following table and figure.

Table 4.3 PSNR results for methods

Cover messages	Length of cover messages	size of cover message (MB)	Embedded secret message	4-LSB 2nd to 5th	2-LSB 2nd and 3rd	1-LSB 2nd	Proposed method
Blues_cover1_320.mp3	04.41	10.7	Jazz_128_msg2.mp3	41.7497	63.7926	76.3927	94.5494
Classical_cover2_320.mp3	03.12	8.67	Jazz_128_msg2.mp3	29.0487	50.3877	62.0994	79.9468
Jazz_cover3_320.mp3	03.12	8.34	Jazz_128_msg2.mp3	48.6016	71.6625	84.7373	103.2509
Pop_cover4_320.mp3	04.00	9.16	Jazz_128_msg2.mp3	45.1153	67.4984	80.3707	98.6931
R&B_cover5_320.mp3	03.51	8.81	Jazz_128_msg2.mp3	39.0752	54.7370	61.7298	76.7305

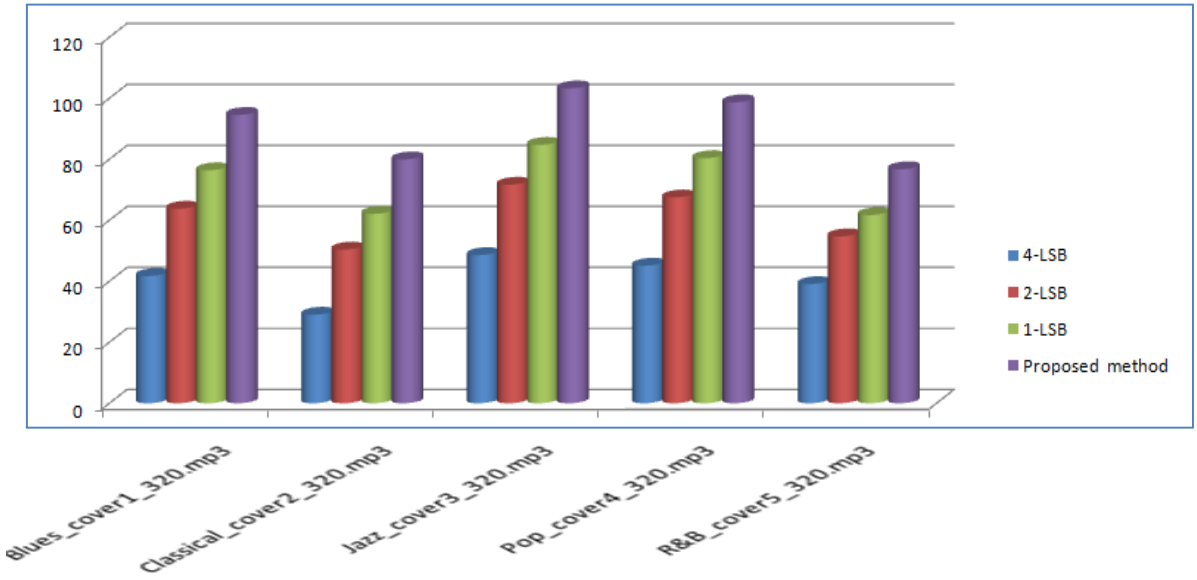


Figure 4.3 PSNR results for methods

The developed technique in this work also outperforms other traditional methods in terms of PSNR values for embedding the second secret message. As shown above, the Jazz offers the highest PSNR since it is one of the MP3 that includes higher noise than other audio files. The enhancement percentage between the current method and the traditional methods for the five cover messages are shown in the table below.

Table 4.4 Enhancement percentage between the current method and the traditional methods for the five cover messages

Cover messages	enhancement percentage between the proposed method and the 4LSB	enhancement percentage between the proposed method and the 2LSB	enhancement percentage between the proposed method and the 1LSB
Blues_cover1_320.mp3	55.84%	32.52%	19.20%
classical_cover2_320.mp3	63.66%	36.97%	22.32%
Jazz_cover3_320.mp3	52.92%	30.59%	17.93%
Pop_cover4_320.mp3	54.28%	31.60%	18.56%
R&B_cover5_320.mp3	49.07%	28.66%	19.54%

4.3. Attack Part

In this part, an attack; Additive White Gaussian noise (AWGN) is added to the stego file before extracting the message in the previous two cases as explored below in order to extract the message and compare it with the original one based on computing the PSNR percentage.

4.3.1. Adding AWGN Attack to the First Secret Audio Message

In this case, an AWGN attack is added to the Blues_96_msg1.mp3 secret message, which embedded in the proposed five cover messages using the current Bi-LSB technique, with several variances. The following table shows the obtained PSNR values for the attack for 0.1 bits/sec/Hz and 0.3 bits/sec/Hz variance, for the all band of the stego file.

Table 4.5 Results of adding AWGN to the first secret message - awgn=0.1

Cover messages	Embedded secret message	PSNR without attack	**Noise 1 variance**	PSNR with attack	Degredation percentage
Blues_cover1_320.mp3	Blues_96_msg1.mp3	95.2852	0.1	92.5227	2.89%
Classical_cover2_320.mp3	Blues_96_msg1.mp3	75.8143	0.1	70.5173	6.98%
Jazz_cover3_320.mp3	Blues_96_msg1.mp3	102.7913	0.1	98.6331	4.04%
Pop_cover4_320.mp3	Blues_96_msg1.mp3	73.4542	0.1	70.5211	3.99%
R&B_cover5_320.mp3	Blues_96_msg1.mp3	101.4737	0.1	97.5227	3.89%

The table below shows the achieved PSNR values for the attack for 0.3 bits/sec/Hz variance, for the all band of the stego file.

Table 4.6 Results of adding AWGN to the first secret message - awgn=0.3

Cover messages	Embedded secret message	PSNR without attack	**Noise 2 variance**	PSNR with attack	Degredation percentage
Blues_cover1_320.mp3	Blues_96_msg1.mp3	95.2852	0.3	90.1174	5.42%
Classical_cover2_320.mp3	Blues_96_msg1.mp3	75.8143	0.3	68.1577	10.09%
Jazz_cover3_320.mp3	Blues_96_msg1.mp3	102.7913	0.3	94.9447	7.63%
Pop_cover4_320.mp3	Blues_96_msg1.mp3	73.4542	0.3	64.5411	12.13%
R&B_cover5_320.mp3	Blues_96_msg1.mp3	101.4737	0.3	93.9547	7.40%

It can be concluded that there is an obvious degradation in PSNR value after adding an AWGN. As shown in the tables above, the degredation in the PSNR value increases with the increase in the noise variance value.

The resultant degradation in PSNR values after adding the AWGN attack with 0.1 and 0.3 variance is shown in the following figures.

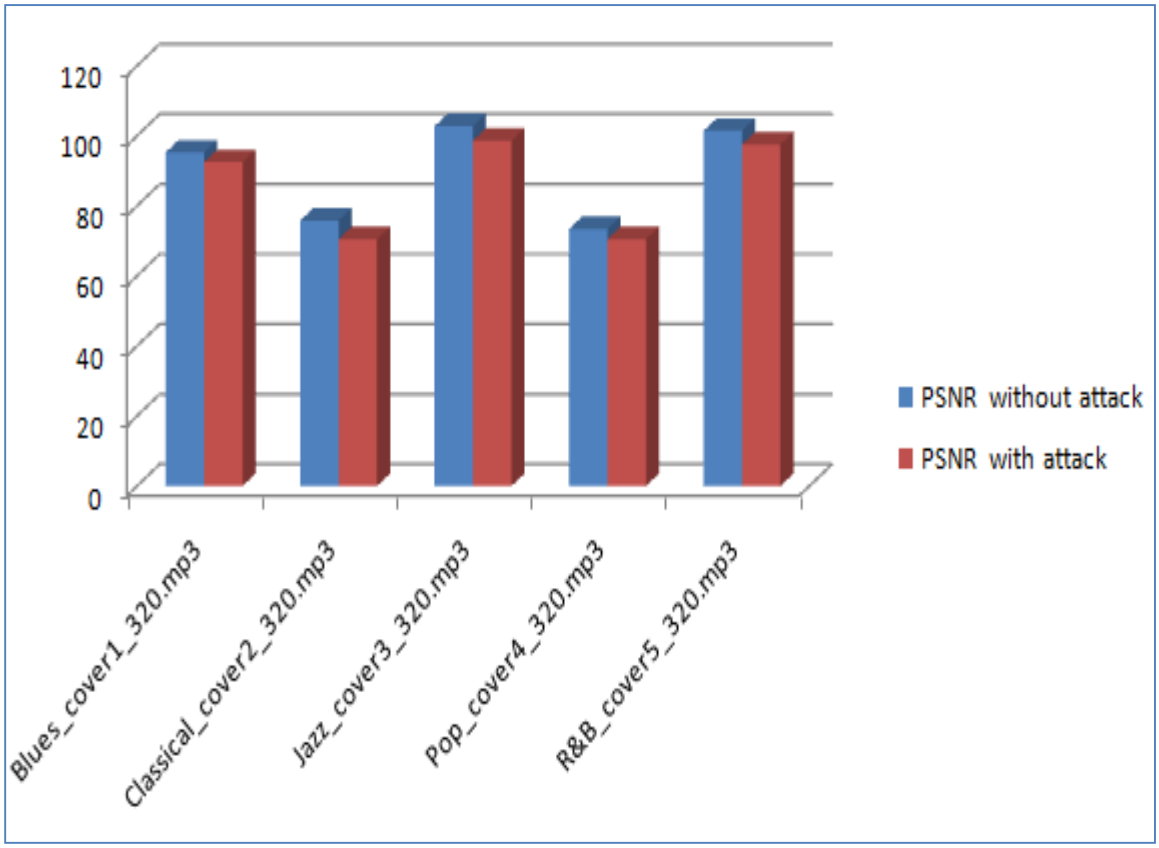


Figure 4.4 Degradation in PSNR values after adding the AWGN attack with 0.1 variance

The above figure illustrates that highest PSNR values occurred at Jazz_cover3, while miumum values of PSNR occurred at Pop_cover4.

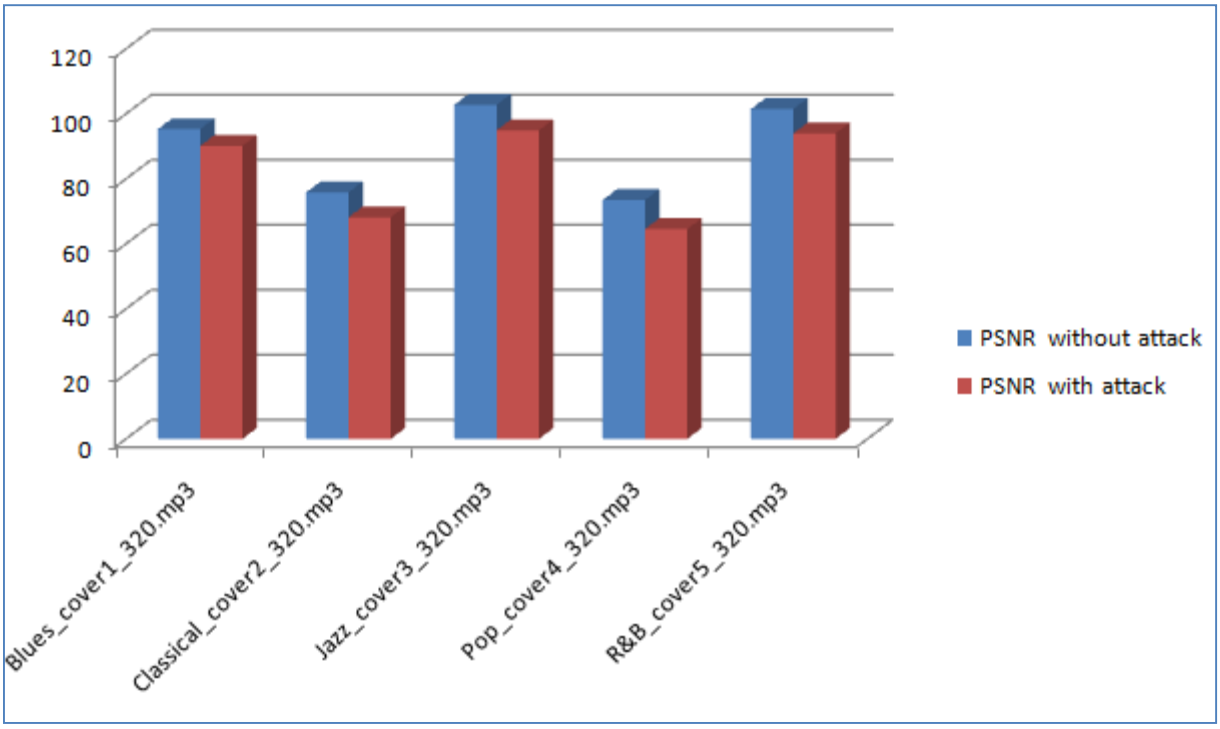


Figure 4. 5 Degradation in PSNR values after adding the AWGN attack with 0.3 variance

The above figure illustrates that highest PSNR values occurred at Jazz_cover3, while miumum values of PSNR occurred at Pop_cover4. while deviation in PSNR in all cases is widley appear at variance 0.3.

4.3.2. Adding AWGN Attack to the Second Secret Audio Message

This section explores the obtained results after adding an AWGN attack to the Jazz_128_msg2.mp3 secret message, which embedded using the current Bi-LSB technique in the proposed five cover messages, with 0.1 and 0.3 bits/sec/Hz variance. The resultant degredation in the PSNR values are shown in the following two tables

Table 4.7 Results of adding AWGN to the second secret message -awgn=0.1

Cover messages	Embedded secret message	PSNR without attack	Noise 1 variance	PSNR with attack 1	Degredation percentage
Blues_cover1_320.mp3	**Jazz_128_msg2.mp3**	**94.5494**	0.1	91.4884	3.23%
Classical_cover2_320.mp3	**Jazz_128_msg2.mp3**	**79.9468**	0.1	75.5149	5.54%
Jazz_cover3_320.mp3	**Jazz_128_msg2.mp3**	**103.2509**	0.1	97.6331	5.44%
Pop_cover4_320.mp3	**Jazz_128_msg2.mp3**	**98.6931**	0.1	94.5227	4.22%
R&B_cover5_320.mp3	**Jazz_128_msg2.mp3**	**76.7305**	0.1	71.5227	6.78%

Table 4.8 Results of adding AWGN to the second secret message awgn=0.3

Cover messages	Embedded secret message	PSNR without attack	Noise 2 variance	PSNR with attack 1	Degredation percentage
Blues_cover1_320.mp3	**Jazz_128_msg2.mp3**	**94.5494**	0.3	88.5144	6.38%
Classical_cover2_320.mp3	**Jazz_128_msg2.mp3**	**79.9468**	0.3	71.7444	10.25%
Jazz_cover3_320.mp3	**Jazz_128_msg2.mp3**	**103.2509**	0.3	94.5144	8.46%
Pop_cover4_320.mp3	**Jazz_128_msg2.mp3**	**98.6931**	0.3	90.7594	8.03%
R&B_cover5_320.mp3	**Jazz_128_msg2.mp3**	**76.7305**	0.3	66.5447	13.27%

The tables above illustrate that there is a clear degradation in the PSNR values after adding the AWGN noise. In addition, it is obvious that the degradation in the PSNR values is directly related to the noise variance value, where it is more for 0.3 variance value than that of the 0.1 value.

The resultant degradation in PSNR values after adding the AWGN attack with 0.1 and 0.3 variance are shown in the following figures.

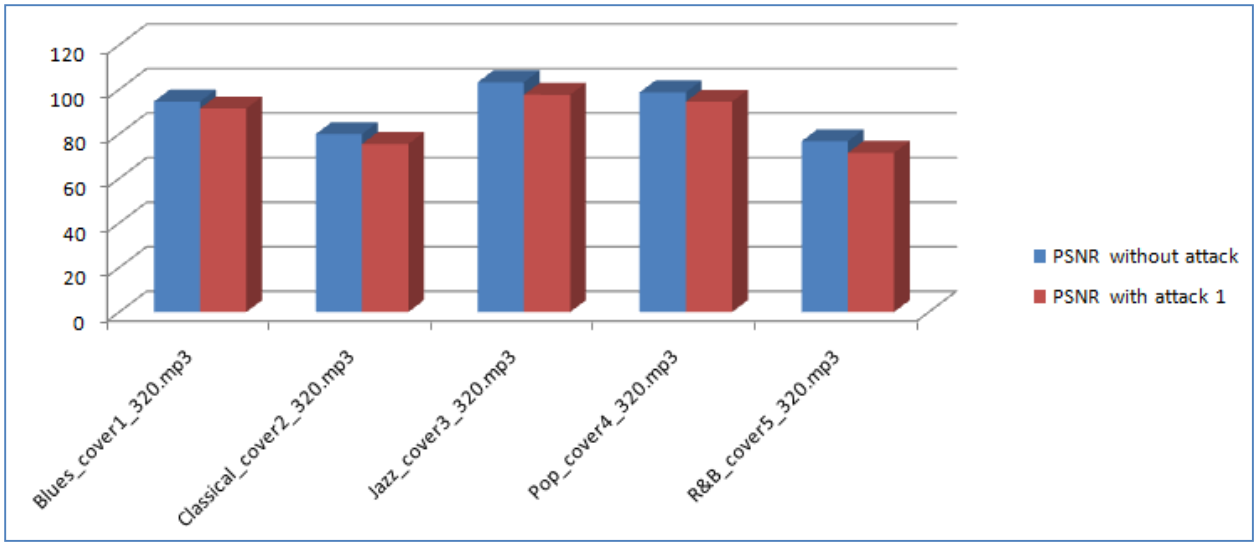


Figure 4.6 Resultant degradation in PSNR values after adding the AWGN attack with 0.1 variance

The above figure illustrates that PSNR values with attack is clearly appear at R&B_cover5 , while its effect is very small at Blues_cover1.

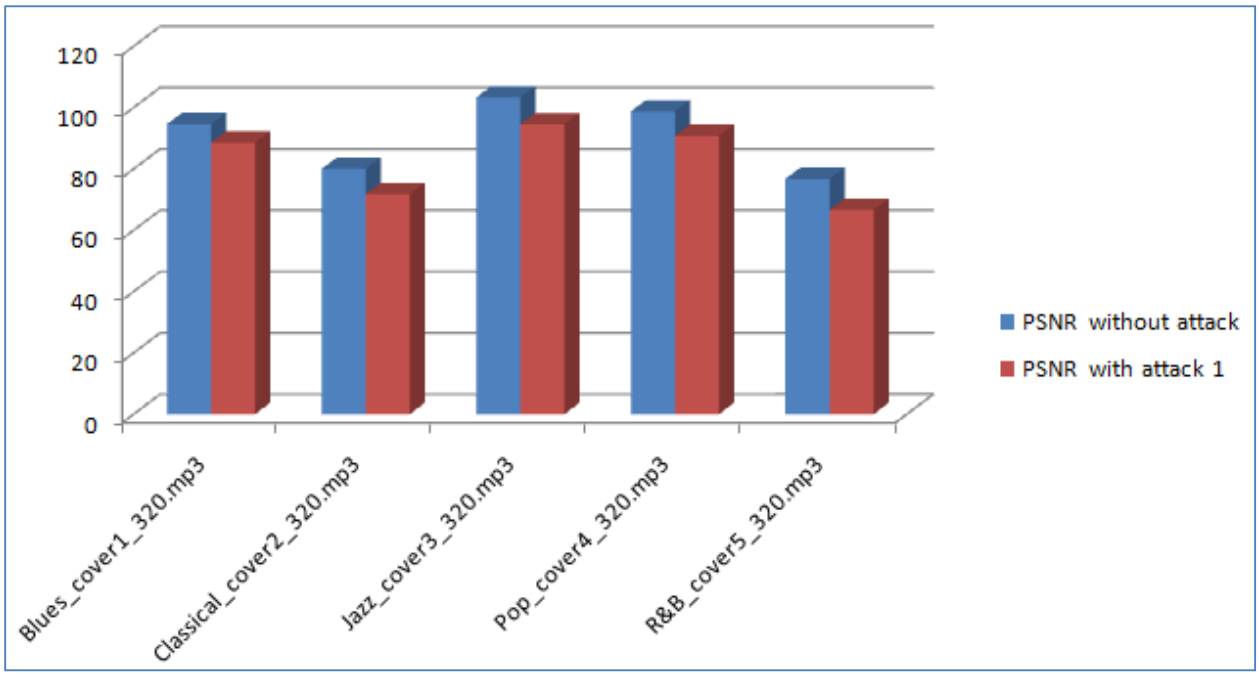


Figure 4. 7 Resultant degradation in PSNR values after adding the AWGN attack with 0.3 variance

The above figure illustrates that PSNR values with attack is clearly appear at Classical_cover2 , while its effect is very small at Blues_cover1.

4.4 Integrity Part

In this step, the hidden secret message in the cover in the presented two cases are extrcated and compared with the original secret message. In this stage, the resulate are obtaned by comaping values of codebook with extraction stage. This is performed using three techniques; checksum, hash function and frequency methods:

- The checksum technique offers steganogram integrity certification to simplify the recognition of packets, which carry secret data at the end of receieved message. This technique counts the number of bits in the extracted message to chek if it is the same as the original one, as shown below

```
% 1) checksum

if (sum(sum(double(dec2bin(message_extracted*2^(NBITS-1),NBITS))))-
size(Mex_extracted,1)*size(Mex_extracted,2)*48)==(sum(sum(double(dec2bin(abs(message_extracte
d_with_white_noise)*2^(NBITS-1),NBITS))))-size(Mex_extracted,1)*size(Mex_extracted,2)*48)
   disp('Checksum: 100 %')
else
   disp(['Checksum: ',num2str((sum(sum(double(dec2bin(message_extracted*2^(NBITS-
1),NBITS))))-
size(Mex_extracted,1)*size(Mex_extracted,2)*48)/(sum(sum(double(dec2bin(abs(message_extracted
_with_white_noise)*2^(NBITS-1),NBITS))))-
size(Mex_extracted,1)*size(Mex_extracted,2)*48)*100),' %'])
end
```

- The hash function represents the mapping of digital data with random size to data. The offered values by this function are known as hash codes, hash sums, has values or hashes. The main properties of this function are that it is a one-way technique, where the digest is produced from the message, not ice-versa and it is a one-to-one function, where there is some probability that the same digest can result from two different messages, as shown below.

```
 % 2) hash function check

if DataHash(message_extracted)==DataHash(message_extracted_with_white_noise)
   disp('Hash function check: 100 %')
else
   disp(['Hash function check:
',num2str(DataHash(message_extracted)/DataHash(message_extracted_with_white_noise)*100),' %'])
end
```

- The frequency technique represents counting the number of ones in the message and comparing it with that number in the extracted message, as shown below

```
    % 3) frequency check

    ones_in_original_mex=length(find(double(dec2bin(abs(message_extracted_with_white_noise)*2^(NB
    ITS-1)))));;
    ones_in_extracted_mex=length(find(double(dec2bin(abs(message_extracted)*2^(NBITS-1)))));;

    if ones_in_original_mex==ones_in_extracted_mex
       disp('Frequency check: 100 %')
    else
       disp(['Frequency check: ',num2str(ones_in_extracted_mex/ones_in_original_mex*100),' %'])
    end
```

4.4.1. Results of Applying Checksum, Hash Function and Frequency Methods to the First Secret Audio Message

The following table represents the achieved correlation percentage among the original first message and the extracted one without adding noise.

Table 4.9 Comparison between different integrity methods

Cover	msg	Checksum	hash function check	frequency check
Blues_cover1_320.mp3	Blues_96_msg1.mp3	100%	100%	100%
Classical_cover2_320.mp3	Blues_96_msg1.mp3	100%	100%	100%
Jazz_cover3_320.mp3	Blues_96_msg1.mp3	100%	100%	100%
Pop_cover4_320.mp3	Blues_96_msg1.mp3	100%	100%	100%
R&B_cover5_320.mp3	Blues_96_msg1.mp3	100%	100%	100%

As shown in the table above, the correlation among the original message and the extrcated one is 100% when there is no added noise for the message.

Adding Addadtive white gaussian noise can be done ashsown below.

```
% Adding white gaussian noise to message extracted
adding_wn=input('Do you want to add white noise to extracted message? (yes 1, no 2): ');

if adding_wn==1
    message_extracted_with_white_noise=awgn(message_extracted,10);
else
    message_extracted_with_white_noise=message_extracted;
end
```

Based on adding an AWGN to the stego file before extrcation, the achieved correlation percentage among the original message and the extracted one are shown in the following tables.

Table 4.10 Adding AWGN with 0.1 variance

Cover	msg	Checksum %	hash function check %	frequency check %
Blues_cover1_320.mp3	Blues_96_msg1.mp3	20.9695 %	86.2071 %	50 %
Classical_cover2_320.mp3	Blues_96_msg1.mp3	21.0825 %	91.4787 %	50 %
Jazz_cover3_320.mp3	Blues_96_msg1.mp3	20.9695 %	86.2071 %	50 %
Pop_cover4_320.mp3	Blues_96_msg1.mp3	20.9869 %	90.6557 %	50 %
R&B_cover5_320.mp3	Blues_96_msg1.mp3	20.9925 %	81.8492 %	50 %

As shown in the table above, the best achieved correlation percentage for the three cover messages is by the has function check method, while the minimum achieved results are for the checksum one. The following table disply those results.

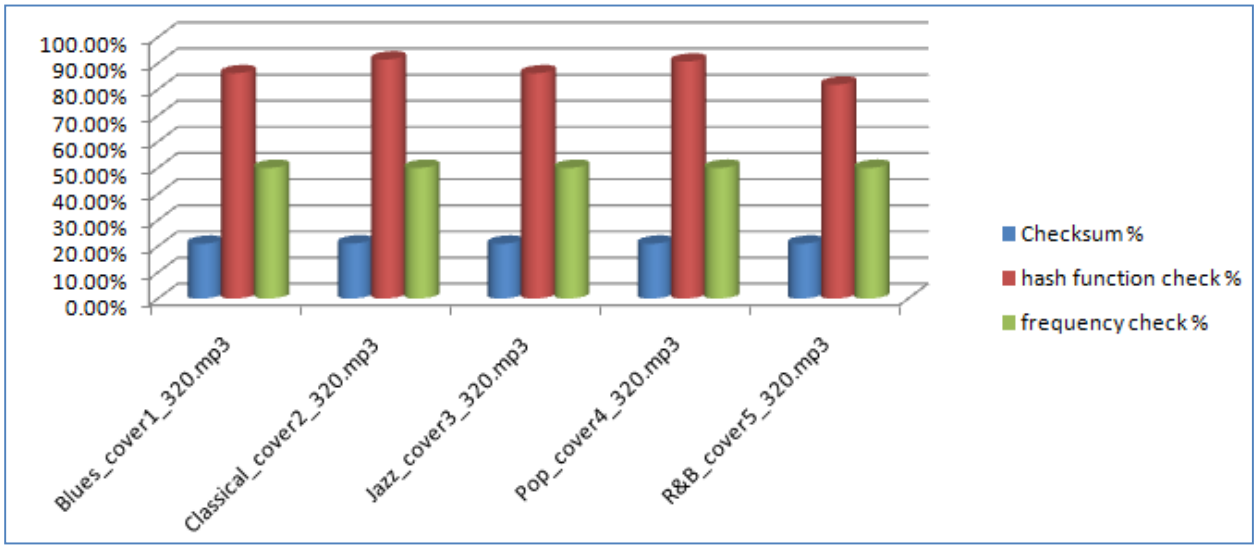


Figure 4.8 Comparison between the three integrity methods after adding an AWGN with 0.1 variance

4.4.2. Results of Applying Checksum, Hash Function and Frequency Methods to the Second Secret Audio Message

The table below disply the achieved correlation percentage among the original second message and the extracted one without adding noise

Table 4.11 Comparison between different integrity methods

Cover	msg	Checksum	hash function check	frequency check
Blues_cover1_320.mp3	Jazz_128_msg2.mp3	100%	100%	100%
Classical_cover2_320.mp3	Jazz_128_msg2.mp3	100%	100%	100%
Jazz_cover3_320.mp3	Jazz_128_msg2.mp3	100%	100%	100%
Pop_cover4_320.mp3	Jazz_128_msg2.mp3	100%	100%	100%
R&B_cover5_320.mp3	Jazz_128_msg2.mp3	100%	100%	100%

As shown in the table above, the correlation among the original message and the extrcated one is 100% when there is no added noise for the second message.

Based on adding an AWGN to the stego file before extracted secret message, the achieved correlation percentage among the original message and the extracted one are shown in the following table.

Table 4.12 Adding AWGN with 0.1 variance

Cover	msg	Checksum %	hash function check %	frequency check %
Blues_cover1_320.mp3	Jazz_128_msg2.mp3	73.4203 %	92.0659 %	87.5 %
Classical_cover2_320.mp3	Jazz_128_msg2.mp3	73.6589 %	91.3738 %	87.5 %
Jazz_cover3_320.mp3	Jazz_128_msg2.mp3	73.6275 %	92.3676 %	87.5 %
Pop_cover4_320.mp3	Jazz_128_msg2.mp3	73.6275 %	91.3676 %	87.5 %
R&B_cover5_320.mp3	Jazz_128_msg2.mp3	73.3995 %	89.5476 %	87.5 %

As shown in the table above, the best achieved correlation percentage for the three cover messages is by the hash function check method, while the minimum achieved results are for the checksum one. The following table disply those results.

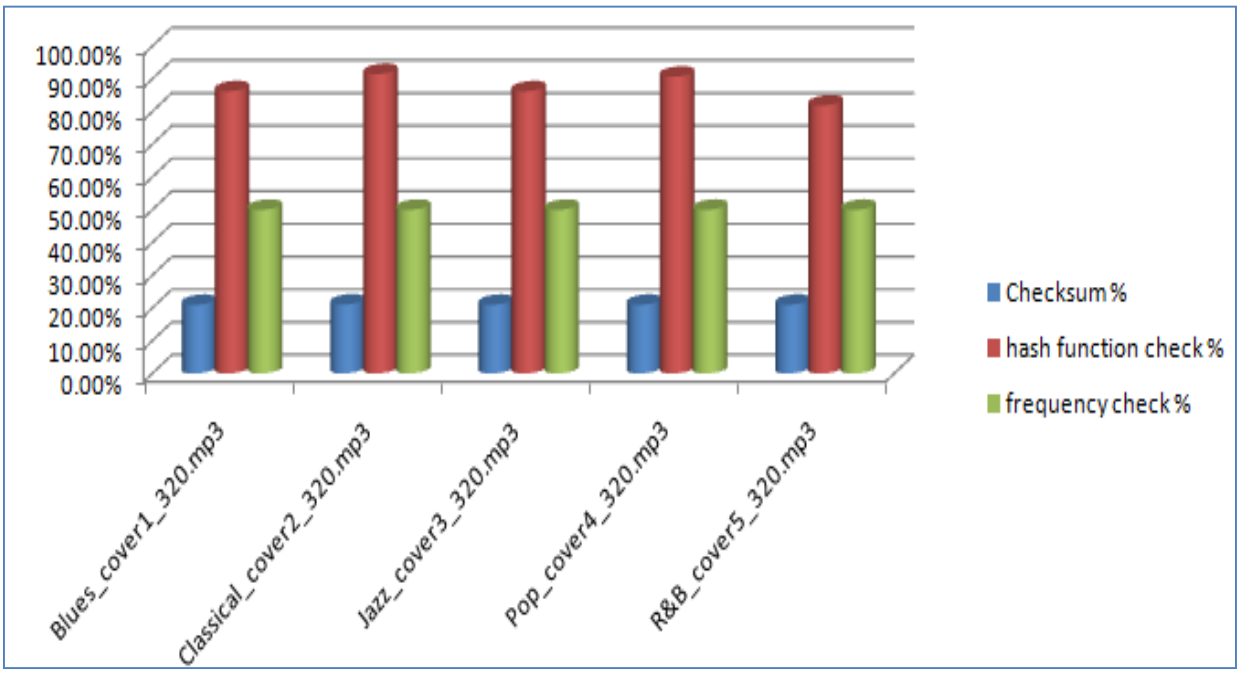


Figure 4.9 Comparison between the three integrity methods after adding an AWGN with 0.1 variance

4.5 Results of Applying Attack and Integrity Methods on 1-LSB

In this case, an AWGN attack is added to the Blues_96_msg1.mp3 secret message, which embedded in the proposed five cover messages using 1-LSB technique. The following table shows the obtained PSNR values for the attack for 0.1 bits/sec/Hz, for the all band of the stego file.

Table 4.13 Results of adding AWGN to the first secret message for 1-LSB technique

Cover messages	Embedded secret message	PSNR without attack	**Noise 1 variance**	PSNR with attack
Blues_cover1_320.mp3	Blues_96_msg1.mp3	76.2214	0.1	65.1109
Classical_cover2_320.mp3	Blues_96_msg1.mp3	57.2895	0.1	48.0195
Jazz_cover3_320.mp3	Blues_96_msg1.mp3	83.1974	0.1	71.9531
Pop_cover4_320.mp3	Blues_96_msg1.mp3	57.9350	0.1	47.1057
R&B_cover5_320.mp3	Blues_96_msg1.mp3	81.9648	0.1	69.9001

The tables above illustrate that there is a clear degradation in the PSNR values after adding the AWGN noise. In addition, it is obvious that the degradation in the PSNR is much worst compared with proposed method (BI-LSB).

The table below shows the correlation percentage among the original message and the extracted one after adding the AWGN to the stego file before extracted secret message for the 1-LSB technique

Table 4. 14Adding AWGN with 0.1 variance for 1-LSB technique

Cover	msg	Checksum %	hash function check %	frequency check %
Blues_cover1_320.mp3	Blues_96_msg1.mp3	18.50 %	83.59 %	44.47 %
Classical_cover2_320.mp3	Blues_96_msg1.mp3	20.45 %	87.43 %	44.47 %
Jazz_cover3_320.mp3	Blues_96_msg1.mp3	17.51 %	82.19 %	44.47 %
Pop_cover4_320.mp3	Blues_96_msg1.mp3	18.11 %	88.95 %	44.47 %
R&B_cover5_320.mp3	Blues_96_msg1.mp3	18.27 %	79.47%	44.47 %

As shown in the table above, the 1-LSB technique offers low correlation percentages for the three integrity methods in comparison with the results of the presented Bi-LSB one.

Chapter Five
Conclusion and Future Works

Chapter Five: Conclusion and Future Works

Conclusions

This work introduces the development of an advanced Least Significant Bit (LSB) technique; Bi-LSB to solve the low security and capacity problems of the traditional used LSB techniques, which do not provide a step for encrypting data, and if secret message is sequentially or randomly embedded and attackers know this pattern of embedding the message, they can obtain the message. In addition, the validation code of those techniques is stored in the stego object. Therefore, the Bi-LSB technique is developed in this work to solve those problems and offer an efficient method to hide audio information in more secured way with the use of the MATLAB program.

The developed technique includes three main steps; preprocessing, embedding and extracting and message validation. In the first stage, the main purpose is to improve the security of messages to be hidden in an MP3 file. In the second stage, the proposed algorithm is designed for MP3 files to solve the current security problem of the traditional LSB technique. In the final stage, another method is implemented to hide messages to recognize the efficiency of hidden messages in MP3 files.

The performance of the developed Bi-LSB technique is compared with that of three traditional LSB techniques; 1-LSB, 2-LSB and 4-LSB based on using them in embedding the same secret messages in the same cover ones. The main simulation parts of both techniques are: hiding, integrity and attack. In the hiding part, two secret messages are hidden in five cover messages using both techniques. The integrity stage represents extracting the hidden secret message in the cover ones and comparing them with the original secret messages using three techniques; checksum, hash function and frequency. In the attack part, an Additive white Gaussian noise (AWGN) is added to the stego file before extrcation with different variances.

Results of the hiding part illustrate that the Peak Signal to Noise Ration (PSNR) results of the proposed method are better than those of the traditional LSB for all genre names and all used secret messages. Results demonstrated that the average enhancement percentage of PSNR between the current method and 4LSB, 2LSB and 1LSB for the first secret message are 52.32%, 33.60% and

20.76%, respectively, while for the second secret message are: 55.15%, 32.06% and 19.51%, respectively. This indicates that the highest enhancement is with the 4LSB method, while the lowest one is with the 1LSB method. On the other hand, the average degredation in the PSNR values after adding an AWGN with 0.1 variance for the first secret message is 4.35%, while it is 8.53% after adding an AWGN with 0.3 variance. For the second secret message, the averages degredation in the PSNR values after adding an AWGN with 0.1 and 0.3 variance are 5.04% and 9.27%, respectively. Thus, the degradation in the PSNR values is directly related to the noise variance value.

In addition, the achieved correlation percentage for the three integrity methods; checksum, hash function and frequency check without noise is 100% for all cover messages, while it decreased with the presence of noise. For the first secret message, the average correlation percentages for the checksum, hash function and frequency check are 21%, 87.28% and 50%, respectively, while for the second secret messages, those values are 73.54%, 91.34% and 87.5%, respectively. Thus, the best correlation percentage was for the hash function check method, while the minimum one was for the checksum method.

Future works

This work can be enhanced in the future based on applying it in hiding more secret messages and adding other types of noises.

Different parameters can be added in order to get high PSNR or to improve message integrity , the following points can be used as future work in this research

- Assign specific packet as reference points between sender and receiver in order to detect any attack in the message.
- Choose different data set in order to compare the work with and without compression of the voice.
- Study the effect of compression ratio on the performance of PSNR.

References

Atoum, M. S., Ibrahimn, S., Sulong, G., Zeki, A., & Abubakar, A. (2013, December). Exploring the Challenges of MP3 Audio Steganography. In *Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on* (pp. 156-161). IEEE.

Alam, S. (2009). Effect of Additive White Gaussian Noise (AWGN) on the Transmitted Data. *Assignment submitted to fulfill the MSc Telecommunications and Computer Networks Engineering*

Alla, K., Prasad, R., & Siva R. (2009). An Evolution of Hindi Text. *Journal of Computer Science*. 9 (5).113-117

Atoum, M. S., Al-Rababah, O. A., & Al-Attili, A. I. (2011)b. New technique for hiding data in audio file. *Journal of Computer Science*, *11*(4), 173-177.

Atoum, M. S., Ibrahim, S., Sulong, G., & Ahmed, A. (2012). MP3 Steganography: Review. *Journal of Computer Science issues*, *9*(6).

Atoum, M. S., Suleiman, M., Rababaa, A., Ibrahim, S., & Ahmed, A. (2011)a. A Steganography Method Based on Hiding secrete data in MPEG / Audio Layer III. *Journal of Computer Science*. 11 (5).184-188

Atoum, M.S., Ibrahim, S., Sulong, G. & Zamani, M. (2013). A New Method for Audio Steganography Using Message Integrity. *Journal of Convergence Information Technology*. 8. 35-44

Bender W., Gruhl D., Morimoto N., & Lu A. (1996). Techniques for Data Hiding", *IBM System Journal*, vol.35, [online]: available at: http://isj.www.media.mit.edu/isj/SectionA/ 313.pdf

Bhattacharyya, S., & Sanyal, G. (2012). Audio Steganalysis of LSB Audio Using Moments and Multiple Regression Model. *International Journal of Advances in Engineering & Technology*. 3(1). 145-160

Bhattacharyya, S., Kundu, A. & Sanyal, G. (2011). a Novel Audio Steganography Technique by M16MA. *International Journal of Computer Application*. 30 (8). 26-34

Bhattacharyya, S., Kundu, A., Chakraborty, K., & Sanyal, G. (2011). Audio Steganography Using Mod 4 Method. *Journal of Computing*. (8).30-38

Brandenburg K. (1999). MP3 and AAC Explained, *Proceeding of AES. 17th International Conference on High Quality Audio Coding*, [online]: available at: http://web.unic.ca/~jinxing/wmp3/3-1.pdf

Cacciaguerra S., & Ferretti S. (2001). Data Hiding: Steganography and Copyright Marking. *Department of Computer Science, University of Bologna, Italy*, [online]: available at: http://www.cs.unibo.it/~scacciag/home-file/teach/datahiding.pdf

Changder, S., Debnath, N. C., & Ghosh, D. (2009). A New Approach to Hindi text Steganography by Shifting Matra. *International Conference on Advances in Recent Technologies in Communication and Computing*. 199-202

Cvejic, N., & Seppanen, T. (2004). Reduced Distortion Bit-Modification For LSB Audio Steganography. *International Conference on Signal Processing*.3. 2318-2321

Dunbar B. (2002). "A detailed look at Steganographic Techniques and their use in an Open Systems Environment", SANS Institute.

Emelia A., Sugathan, S. K., & Ho, A. (2008). Receiver Operating Characteristic (Roc) Graph to Determine the Most Suitable Pairs Analysis Threshold Value. *Advances in Electrical and Electronics Engineering*. 224-230

Farouk, M. H. (2014). Steganography and Security of Speech Signal. In Application of Wavelets in Speech Processing. *Springer International Publishing*. 45-47

Feruza, Y., & Kim, T. (2007). IT Security Review: Privacy, Protection, Access Control, Assurance and System Security. *International Journal of Multimedia and Ubiquitous Engineering*, 2(2), 17-32.

Francia, G. A., & Gomez, T. S. (2006). Steganography Obliterator: An Attack on the Least Significant Bits. *Information Security Curriculum Development Conference*. 85-91

Fricker, R., & Schonlau, M. (2002). Advantages and Disadvantages of Internet Research Surveys: Evidence from the Literature. *Field Methods,* 14(4), 347-367. doi:10.1177/152582202237725

Fridrich J. (1998) . Applications of Data Hiding in Digital Images. Tutorial for the (ISPACS'98) Conference, Melbourne, Australia, URL: http://www.ssie.Binghamton.edu/~jirif.

Galand, F & Kabatiansky, G. A. (2009). Coverings, Centered Codes, and Combinatorial Steganography. *Supported in part by the Russian Foundation for Basic Research*, Moscow.

Ganeshkumar, V., & Koggalage, R. L. W. (2009). Secured Communication using Steganography Framework with Sample RTF Implementation. *4th International Conference on Industrial and Information Systems*.15-21

Groenewald, T. (2004). A Phenomenological Research Design Illustrated

He, X. & Luo, Z. (2008). A Novel Steganographic Algorithm Based on the Motion Vector Phase. *International Conference on Computer Science and Software Engineering*, China.

Huayin, S & Chang-Tsun, L. (2008). Maintaining Information Security in E-Government through Steganography. *Department of Computer Science, University of Warwick*, UK

id3. (2014). The Private Life of MP3 Frames. [online]: available at: www.id3.org/MP3frame.html

Jangra, T., & Singh, D. (2014). Message Guided Random Audio Steganography Using Modified LSB Technique. *International Journal of Computers & Technology*. 12 (5)5. 3464-3469

Johnson N.F., Duricn Z., & Jajodia S. (2001). Information Hiding: Steganography and Watermarking Attack and Counter measurements. Kluwer Academic Publishers, USA

Johnson N.F., Duricn Z., Jajodia S. (2001). "Information Hiding: Steganography and Watermarking Attack and Countermeasurments", Kluwer Academic Publishers, USA.

Karen R. (2001) "Steganography and Steganalysis", URL:www.krenn.n1/univ/cry/ steg/article.pdf.

Katzenbeisser S. & Petitcolas F. (2000). Information Hiding Techniques for Steganography and Digital Watermarking. *Artech House Inc*

Khairullah, M. (2009). A Novel Text Steganography System Using Font Color of the Invisible Characters in Microsoft Word Documents. *International Conference on Computer and Electrical Engineering*. 1. 482-484

Kivanc, M. (2002). Information Hiding Codes and their Applications to Images and Audio, PhD Thesis, University of Illinois at Urbana-Champaign

Kumar, G., Sasidharan, S., Karthikha, N., Sherly, A., & Avani, Y. (2010). An Efficient Embedding and Restoration Steganographic Scheme for Secure Multimedia Communication. *International Conference on Advances in Computer Engineering*, India.

Lee, Y.K., Bell, G., Huang, S.Y., Wang, R.Z. & Shyu, S.J. (2009). An Advanced Least-Significant-Bit Embedding Scheme for Steganographic Encoding. *Advances in Image and Video Technology*. Berlin / Heidelberg: Springer. 349-360

Lenti J. (2000). Steganographic Methods. *Department Of Control Engineering and Information Technology, Budapest University. Periodica Poltechnica Ser. El. Eng*. 44, 249–258

Lin T., Delp J. & Edward. (1998). A Review of Data Hiding in Digital Images. [online]: available at: http://www.ece.purdue.edu/~ace

Liu, Q., Sung, A. H., & Qiao, M. (2009). Novel Stream Mining for Audio Steganalysis. *ACM Multimedia Conference*. 95-104

Liu, Q., Sung, A. H., & Qiao, M. (2009). Spectrum steganalysis of WAV audio Streams. *In Machine Learning and Data Mining in Pattern Recognition*. 582-593

Maciak .L, Ponniah. M., & Sharma. R. (2008). MP3 STEGANOGRAPHY

Mandal, J. K. & Sengupta. M. (2011). Steganographic Technique Based on Minimum Deviation of Fidelity (STMDF). *Second International Conference on Emerging Applications of Information Technology*. India.

Mandala, J., Kotagiri, S., & Kapala, K. (2013). Watermarking Scheme for Color Images". *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS).* 2 (5). 179-182

Manimegalai, P., Gomathi, K. S., Ponniselvi, D., & Santha, M. (2014). The Image Steganography and Steganalysis Based On Peak-Shaped Technique for MP3 Audio and Video. *International Journal of Computer Science and Mobile Computing*. 3 (1). 300-308

Matthews, C. (2003).Behind The Music: Principles of Audio Steganography

Menezes, A., van Oorschot, P. and Vanstone, S. (1996). Hash Functions and Data Integrity, Handbook of Applied Cryptography. By CRC Press

Naji A.W., Zaidan A. A., Zaidan B.B., Shihab A., & Khalifa, O. (2009). Novel Approach for Secure Cover File of Hidden Data in the Unused Area Within exe File Using Computation between Cryptography and Steganography. *International Journal of Computer Science and Network security*. 9. 294-300

Neeta, D., Snehal, K., & Jacobs, D. (2006). Implementation of LSB Steganography and Its Evaluation for Various Bits. *International Conference on Digital Information Management*. 173-178

Pan, D. (1995). A Tutorial on MPEG / Audio Compression. *IEEE Multimedia*. 60 – 74

Petrovic, R., & Yang, D. T. (2009). Audio Watermarking in Compressed Domain. *International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services*. 395-401

Polpitiya A.D., & Khan W.J. (2001). Information Hiding in Audio Files with Encryption", Data Security Project, Washington University

Provos N. (2001). Probabilistic Method for Improving Information Hiding. *CITI Technical Report* 01-1

Qi, Y., Ye, L., & Liu, C. (2009). Wavelet Domain Audio Steganalysis for Multiplicative Embedding Model. *International Conference on Wavelet Analysis and Pattern Recognition*. 429-432

Rahim, L. B. A., Bhattacharjee, S., & Aziz, I. B. (2014). An Audio Steganography Technique to Maximize Data Hiding Capacity along with Least Modification of Host. *In Proceedings of the*

First International Conference on Advanced Data and Information Engineering (DaEng-2013) Springer Singapore. 277-289

Rahman, S.M.M., Hossain, M.A., Mouftah, H., El Saddik, A. & Okamoto, E. (2010). Real Time Privacy Sensitive Data Hiding Approach Based on Chaos Cryptography, *IEEE*, Canada.

Ramkumar, M., Akansu, A. & Alatan, A. (1999). On The Choice Of Transforms For Data Hiding In Compressed Video. *IEEE*, USA.

Santosa, R. and Bao, P. (2005). Audio to image wavelet transform based audio steganography. *Proceeding of 47th International Symposium, ELMAR*, pp. 209- 212

Scagliola, M., Pérez, F., & Guccione, P. (2009). An Extended Analysis of Discrete Fourier Transform - Rational Dither Modulation For Non-White Hosts", *1st IEEE International Workshop On Information Forensics and Security*. 146-150

Sellars D., (2003) "An Introduction to Steganography", University of Camberge.

Shahadi, H., Jidin, R. (2011). High Capacity and Resistance to Additive Noise Audio Steganography Algorithm. *IJCSI International Journal of Computer Science Issues*. **8** (2). pp. 176-184

Shahreza, S. and Shalmani, M. (2008). High capacity error free wavelet Domain Speech Steganography. *IEEE International conference on acoustics, speech, and signal processing*, pp. 1729 - 1732.

Sivathanu, G. Wright, C. P and Zadok, E. (2005). Ensuring Data Integrity in Storage: Techniques and Applications. *a report submitted to Stony Brook University*

Supurovic P. (1998). MPEG Audio Compression Basics. [online]: available at: http://www.chested. chalmers.se\~kf96svgu

Supurovic. P. (1998). MPEG Audio Frame Header. *Data Voyage*, [online]: available at: www.dv.co.yu/mpgscript/mpeghdr.htm#MPEGTAG

Tomar, G. (2012). Effect of Noise on image steganography based on LSB insertion and RSA encryption. *IOSR Journal of Engineering*. **2**(3) pp: 473-477

Yan, W. & Ping, L. (2009). A New Steganography Algorithm Based on Spatial Domain. *Second International Symposium on Information Science and Engineering*, China.

Yang Y. (2001). "Digital Watermarking Technology", Faculty of Computer Science, Dalhousie University, URL: http://www.cs.dal.ca/~yyang/6505/6605.pdf.

Yusnita Y. & Othman O. K. (2007). Digital Watermarking for Digital Images using Wavelet Transform. *14th IEEE International Conference On Telecommunications*. Penang, Malaysia

Zaturenskiy. M. (2009). Behind The Music: MP3 steganography. [online]: available at: http://www.cpd.iit.edu/netsecure09/MIKHAIL_ZATURENSKIY.pdf

Appendix

The full code is available with the student, you can get it based on contacting him via mail:

mms_8921@yahoo.com

```
function [Y,FS,NBITS,OPTS] = mp3read(FILE,N,MONO,DOWNSAMP,DELAY)
% MP3READ   Read MP3 audio file via use of external binaries.
%   Y = MP3READ(FILE) reads an mp3-encoded audio file into the
%     vector Y just like wavread reads a wav-encoded file (one channel
%     per column).  Extension ".mp3" is added if FILE has none.
%     Also accepts other formats of wavread, such as
%   Y = MP3READ(FILE,N) to read just the first N sample frames (N
%     scalar), or the frames from N(1) to N(2) if N is a two-element vector.
%   Y = MP3READ(FILE,FMT) or Y = mp3read(FILE,N,FMT)
%     with FMT as 'native' returns int16 samples instead of doubles;
%     FMT can be 'double' for default behavior (to exactly mirror the
%     syntax of wavread).
%
%   [Y,FS,NBITS,OPTS] = MP3READ(FILE...) returns extra information:
%     FS is the sampling rate,  NBITS is the bit depth (always 16),
%     OPTS.fmt is a format info string; OPTS has multiple other
%     fields, see WAVREAD.
%
%   SIZ = MP3READ(FILE,'size') returns the size of the audio data contained
%     in the file in place of the actual audio data, returning the
%     2-element vector SIZ=[samples channels].
%
%   [Y...] = MP3READ(FILE,N,MONO,DOWNSAMP,DELAY) extends the
%     WAVREAD syntax to allow access to special features of the
%     mpg123 engine:  MONO = 1 forces output to be mono (by
%     averaging stereo channels); DOWNSAMP = 2 or 4 downsamples by
%     a factor of 2 or 4 (thus FS returns as 22050 or 11025
%     respectively for a 44 kHz mp3 file);
%     To accommodate a bug in mpg123-0.59, DELAY controls how many
%     "warm up" samples to drop at the start of the file; the
%     default value of 2257 makes an mp3write/mp3read loop for a 44
%     kHz mp3 file be as close as possible to being temporally
%     aligned; specify as 0 to prevent discard of initial samples.
%     For later versions of mpg123 (e.g. 1.9.0) this is not needed;
%     a flag in mp3read.m makes the default DELAY zero in this case.
%
%   [Y...] = MP3READ(URL...)  uses the built-in network
%     functionality of mpg123 to read an MP3 file across the
%     network.  URL must be of the form 'http://...' or
%     'ftp://...'.  'size' and OPTS are not available in this mode.
%
%   Example:
%   To read an mp3 file as doubles at its original width and sampling rate:
%     [Y,FS] = mp3read('piano.mp3');
%   To read the first 1 second of the same file, downsampled by a
%   factor of 4, cast to mono, using the default filename
%   extension:
%     [Y,FS4] = mp3read('piano', FS/4, 1, 4);
%
```

```
%   Note: Because the mp3 format encodes samples in blocks of 26 ms (at
%   44 kHz), and because of the "warm up" period of the encoder,
%   the file length may not be exactly what you expect, depending
%   on your version of mpg123 (recent versions fix warmup).
%
%   Note: requires external binaries mpg123 and mp3info; you
%   can find binaries for several platforms at:
%     http://labrosa.ee.columbia.edu/matlab/mp3read.html
%
%   See also mp3write, wavread.

% $Header: /Users/dpwe/matlab/columbiafns/RCS/mp3read.m,v 1.6 2009/12/08
16:35:23 dpwe Exp dpwe $

% 2003-07-20 dpwe@ee.columbia.edu  This version calls mpg123.
% 2004-08-31 Fixed to read whole files correctly
% 2004-09-08 Uses mp3info to get info about mp3 files too
% 2004-09-18 Reports all mp3info fields in OPTS.fmt; handles MPG2LSF sizes
%            + added MONO, DOWNSAMP flags, changed default behavior.
% 2005-09-28 Fixed bug reading full-rate stereo as 1ch (thx bjoerns@vjk.dk)
% 2006-09-17 Chop off initial 2257 sample delay (for 44.1 kHz mp3)
%            so read-write loop doesn't get progressively delayed.
%            You can suppress this with a 5th argument of 0.
% 2007-02-04 Added support for FMT argument to match wavread
%            Added automatic selection of binary etc. to allow it
%            to work cross-platform without editing prior to
%            submitting to Matlab File Exchange
% 2007-07-23 Tweaks to 'size' mode so it exactly agrees with read data.
% 2009-03-15 Added fixes so 'http://...' file URLs will work.
% 2009-03-26 Added filename length check to http: test (thx fabricio guzman)

% find our baseline directory
path = fileparts(which('mp3read'));

% %%%%% Directory for temporary file (if needed)
%------------------------------
tmpdir = 'E:\sounds\tmp\';  % don't forget trailing slash
rmcmd = 'del';
%------------------------------
% % Try to read from environment, or use /tmp if it exists, or use CWD
tmpdir = getenv('TMPDIR');
if isempty(tmpdir) || exist(tmpdir,'file')==0
  tmpdir = '/tmp';
end
if exist(tmpdir,'file')==0
  tmpdir = '';
end
% ensure it exists
%if length(tmpdir) > 0 && exist(tmpdir,'file')==0
%  mkdir(tmpdir);
%end

%%%%%% Command to delete temporary file (if needed)
rmcmd = 'rm';

%%%%%% Location of the binaries - attempt to choose automatically
```

```
%%%%%% (or edit to be hard-coded for your installation)
%--------------------------------------
mpg123 = 'E:\sounds\mpg123.exe';
mp3info = 'E:\sounds\mp3info.exe';
ame = 'E:\sounds\lame.exe';
%--------------------------------------
ext = lower(computer);
if ispc
  ext = 'exe';
  rmcmd = 'del';
end
% mpg123-0.59 inserts silence at the start of decoded files, which
% we compensate.  However, this is fixed in mpg123-1.9.0, so
% make this flag 1 only if you have mpg123-0.5.9
MPG123059 = 0;
mpg123 = fullfile(path,['mpg123.',ext]);
mp3info = fullfile(path,['mp3info.',ext]);

%%%%% Check for network mode
if length(FILE) > 6 && (strcmp(lower(FILE(1:7)),'http://') == 1 ...
      || strcmp(lower(FILE(1:6)),'ftp://'))
  % mp3info not available over network
  OVERNET = 1;
else
  OVERNET = 0;
end



%%%%% Process input arguments
if nargin < 2
  N = 0;
end

% Check for FMT spec (per wavread)
FMT = 'double';
if ischar(N)
  FMT = lower(N);
  N = 0;
end

if length(N) == 1
  % Specified N was upper limit
  N = [1 N];
end
if nargin < 3
  forcemono = 0;
else
  % Check for 3rd arg as FMT
  if ischar(MONO)
    FMT = lower(MONO);
    MONO = 0;
  end
  forcemono = (MONO ~= 0);
end
if nargin < 4
  downsamp = 1;
```

```
else
  downsamp = DOWNSAMP;
end
if downsamp ~= 1 && downsamp ~= 2 && downsamp ~= 4
  error('DOWNSAMP can only be 1, 2, or 4');
end

% process DELAY option (nargin 5) after we've read the SR

if strcmp(FMT,'native') == 0 && strcmp(FMT,'double') == 0 && ...
      strcmp(FMT,'size') == 0
  error(['FMT must be ''native'' or ''double'' (or ''size''), not
''',FMT,'''']);
end


%%%%%% Constants
NBITS=8;

%%%%% add extension if none (like wavread)
[path,file,ext] = fileparts(FILE);
if isempty(ext)
  FILE = [FILE, '.mp3'];
end

if ~OVERNET
  %%%%%% Probe file to find format, size, etc. using "mp3info" utility
  cmd = ['"',mp3info, '" -r m -p "%Q %u %b %r %v * %C %e %E %L %O %o %p" "',
FILE,'"'];
  % Q = samprate, u = #frames, b = #badframes (needed to get right answer
from %u)
  % r = bitrate, v = mpeg version (1/2/2.5)
  % C = Copyright, e = emph, E = CRC, L = layer, O = orig, o = mono, p = pad
  w = mysystem(cmd);
  % Break into numerical and ascii parts by finding the delimiter we put in
  starpos = findstr(w,'*');
  nums = str2num(w(1:(starpos - 2)));
  strs = tokenize(w((starpos+2):end));

  SR = nums(1);
  nframes = nums(2);
  nchans = 2 - strcmp(strs{6}, 'mono');
  layer = length(strs{4});
  bitrate = nums(4)*1000;
  mpgv = nums(5);
  % Figure samples per frame, after
  % http://board.mp3-tech.org/view.php3?bn=agora_mp3techorg&key=1019510889
  if layer == 1
    smpspfrm = 384;
  elseif SR < 32000 && layer ==3
    smpspfrm = 576;
    if mpgv == 1
      error('SR < 32000 but mpeg version = 1');
    end
  else
```

```
    smpspfrm = 1152;
  end

  OPTS.fmt.mpgBitrate = bitrate;
  OPTS.fmt.mpgVersion = mpgv;
  % fields from wavread's OPTS
  OPTS.fmt.nAvgBytesPerSec = bitrate/8;
  OPTS.fmt.nSamplesPerSec = SR;
  OPTS.fmt.nChannels = nchans;
  OPTS.fmt.nBlockAlign = smpspfrm/SR*bitrate/8;
  OPTS.fmt.nBitsPerSample = NBITS;
  OPTS.fmt.mpgNFrames = nframes;
  OPTS.fmt.mpgCopyright = strs{1};
  OPTS.fmt.mpgEmphasis = strs{2};
  OPTS.fmt.mpgCRC = strs{3};
  OPTS.fmt.mpgLayer = strs{4};
  OPTS.fmt.mpgOriginal = strs{5};
  OPTS.fmt.mpgChanmode = strs{6};
  OPTS.fmt.mpgPad = strs{7};
  OPTS.fmt.mpgSampsPerFrame = smpspfrm;
else
  % OVERNET mode
  OPTS = [];
  % guesses
  smpspfrm = 1152;
  SR = 44100;
  nframes = 0;
end

if SR == 16000 && downsamp == 4
  error('mpg123 will not downsample 16 kHz files by 4 (only 2)');
end

% process or set delay
if nargin < 5

  if MPG123059
    mpg123delay44kHz = 2257;  % empirical delay of lame/mpg123 loop
    mpg123delay16kHz = 1105;  % empirical delay of lame/mpg123 loop for 16
kHz sampling
    if SR == 16000
      rawdelay = mpg123delay16kHz;
    else
      rawdelay = mpg123delay44kHz;  % until we know better
    end
    delay = round(rawdelay/downsamp);
  else
    % seems like predelay is fixed in mpg123-1.9.0
    delay = 0;
  end
else
  delay = DELAY;
end

if downsamp == 1
  downsampstr = '';
```

```
else
  downsampstr = [' -',num2str(downsamp)];
end
FS = SR/downsamp;

if forcemono == 1
  nchans = 1;
  chansstr = ' -m';
else
  chansstr = '';
end

% Size-reading version
if strcmp(FMT,'size') == 1
   Y = [floor(smpspfrm*nframes/downsamp)-delay, nchans];
else

  % Temporary file to use
  tmpfile = fullfile(tmpdir, ['tmp',num2str(round(1000*rand(1))),'.wav']);

  skipx = 0;
  skipblks = 0;
  skipstr = '';
  sttfrm = N(1)-1;

  % chop off transcoding delay?
  %sttfrm = sttfrm + delay;  % empirically measured
  % no, we want to *decode* those samples, then drop them
  % so delay gets added to skipx instead

  if sttfrm > 0
    skipblks = floor(sttfrm*downsamp/smpspfrm);
    skipx = sttfrm - (skipblks*smpspfrm/downsamp);
    skipstr = [' -k ', num2str(skipblks)];
  end
  skipx = skipx + delay;

  lenstr = '';
  endfrm = -1;
  decblk = 0;
  if length(N) > 1
    endfrm = N(2);
    if endfrm > sttfrm
      decblk = ceil((endfrm+delay)*downsamp/smpspfrm) - skipblks + 10;
      % we read 10 extra blks (+10) to cover the case where up to 10 bad
      % blocks are included in the part we are trying to read (it happened)
      lenstr = [' -n ', num2str(decblk)];
      % This generates a spurious "Warn: requested..." if reading right
      % to the last sample by index (or bad blks), but no matter.
    end
 end

  % Run the decode
  cmd=['"',mpg123,'"', downsampstr, chansstr, skipstr, lenstr, ...
       ' -q -w "', tmpfile,'"  "',FILE,'"'];
```

```
  %w =
  mysystem(cmd);

  % Load the data
  Y = wavread(tmpfile);

%  % pad delay on to end, just in case
%  Y = [Y; zeros(delay,size(Y,2))];
%  % no, the saved file is just longer

  if decblk > 0 && length(Y) < decblk*smpspfrm/downsamp
    % This will happen if the selected block range includes >1 bad block
    disp(['Warn: requested ', num2str(decblk*smpspfrm/downsamp),' frames,
returned ',num2str(length(Y))]);
  end

  % Delete tmp file
  mysystem([rmcmd,' "', tmpfile,'"']);

  % debug
%  disp(['sttfrm=',num2str(sttfrm),' endfrm=',num2str(endfrm),'
skipx=',num2str(skipx),' delay=',num2str(delay),' len=',num2str(length(Y))]);

  % Select the desired part
  if skipx+endfrm-sttfrm > length(Y)
      endfrm = length(Y)+sttfrm-skipx;
  end

  if endfrm > sttfrm
    Y = Y(skipx+(1:(endfrm-sttfrm)),:);
  elseif skipx > 0
    Y = Y((skipx+1):end,:);
  end

  % Convert to int if format = 'native'
  if strcmp(FMT,'native')
    Y = int16((2^15)*Y);
  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function w = mysystem(cmd)
% Run system command; report error; strip all but last line
[s,w] = system(cmd);
if s ~= 0
  error(['unable to execute ',cmd,' (',w,')']);
end
% Keep just final line
w = w((1+max([0,findstr(w,10)])):end);
% Debug
%disp([cmd,' -> ','*',w,'*']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function a = tokenize(s,t)
```

```
% Break space-separated string into cell array of strings.
% Optional second arg gives alternate separator (default ' ')
% 2004-09-18 dpwe@ee.columbia.edu
if nargin < 2;  t = ' '; end
a = [];
p = 1;
n = 1;
l = length(s);
nss = findstr([s(p:end),t],t);
for ns = nss
  % Skip initial spaces (separators)
  if ns == p
    p = p+1;
  else
    if p <= l
      a{n} = s(p:(ns-1));
      n = n+1;
      p = ns+1;
    end
  end
end
```

```
function mp3write(D,SR,NBITS,FILE,OPTIONS)
% MP3WRITE   Write MP3 file by use of external binary
%   MP3WRITE(Y,FS,NBITS,FILE) writes waveform data Y to mp3-encoded
%     file FILE at sampling rate FS using bitdepth NBITS.
%     The syntax exactly mirrors WAVWRITE.  NBITS must be 16.
%   MP3WRITE(Y,FS,FILE) assumes NBITS is 16
%   MP3WRITE(Y,FILE) further assumes FS = 8000.
%
%   MP3WRITE(..., OPTIONS) specifies additional compression control
%     options as a string passed directly to the lame encoder
%     program; default is '--quiet -h' for high-quality model.
%
%   Example:
%   To convert a wav file to mp3 (assuming the sample rate is
%   supported):
%     [Y,FS] = wavread('piano.wav');
%     mp3write(Y,FS,'piano.mp3');
%   To force lame to use 160 kbps (instead of default 128 kbps)
%   with the default filename extension (mp3):
%     mp3write(Y,FS,'piano','--quiet -h -b 160');
%
%   Note: The actual mp3 encoding is done by an external binary,
%     lame, which is available for multiple platforms.  Usable
%     binaries are available from:
%     http://labrosa.ee.columbia.edu/matlab/mp3read.html
%
%   Note: MP3WRITE will use the mex file popenw, if available, to
%     open a pipe to the lame encoder.  Otherwise, it will have to
%     write a large temporary file, then execute lame on that file.
%     popenw is available at:
```

```
%       http://labrosa.ee.columbia.edu/matlab/popenrw.html
%     This is a nice way to save large audio files as the
%     incremental output of your code, but you'll have to adapt the
%     central loop of this function (rather than using it directly).
%
%   See also: mp3read, wavwrite, popenw.

% 2005-11-10 Original version
% 2007-02-04 Modified to exactly match wavwrite syntax, and to
%   automatically find architecture-dependent binaries.
% 2007-07-26 Writing of stereo files via tmp file fixed (thx Yu-ching Lin)
%
% $Header: /Users/dpwe/matlab/columbiafns/RCS/mp3write.m,v 1.2 2007/07/26
15:09:16 dpwe Exp $

% find our baseline directory
[path] = fileparts(which('mp3write'));

% %%%%% Directory for temporary file (if needed)
% % Try to read from environment, or use /tmp if it exists, or use CWD
tmpdir = getenv('TMPDIR');
if isempty(tmpdir) || exist(tmpdir,'file')==0
  tmpdir = '/tmp';
end
if exist(tmpdir,'file')==0
  tmpdir = '';
end
% ensure it exists
%if length(tmpdir) > 0 && exist(tmpdir,'file')==0
%  mkdir(tmpdir);
%end

%%%%%% Command to delete temporary file (if needed)
rmcmd = 'rm';

%%%%%% Location of the binary - attempt to choose automatically
%%%%%% (or edit to be hard-coded for your installation)
ext = lower(computer);
if ispc
  ext = 'exe';
  rmcmd = 'del';
end
lame = fullfile(path,['lame.',ext]);

%%%% Process input arguments
% Do we have NBITS?
mynargin = nargin;
if ischar(NBITS)
  % NBITS is a string i.e. it's actually the filename
  if mynargin > 3
    OPTIONS = FILE;
  end
  FILE = NBITS;
  NBITS = 16;
  % it's as if NBITS had been specified...
  mynargin = mynargin + 1;
```

```
end

if mynargin < 5
  OPTIONS = '--quiet -h';  % -h means high-quality psych model
end

[nr, nc] = size(D);
if nc < nr
  D = D';
  [nr, nc] = size(D);
end
% Now rows are channels, cols are time frames (so interleaving is right)

%%%%% add extension if none (like wavread)
[path,file,ext] = fileparts(FILE);
if isempty(ext)
  FILE = [FILE, '.mp3'];
end

nchan = nr;
nfrm = nc;

if nchan == 1
  monostring = ' -m m';
else
  monostring = '';
end

lameopts = [' ', OPTIONS, monostring, ' '];

%if exist('popenw') == 3
if length(which('popenw')) > 0

  % We have the writable stream process extensions
  cmd = ['"',lame,'"', lameopts, '-r -s ',num2str(SR),' - "',FILE,'"'];

  p = popenw(cmd);
  if p < 0
    error(['Error running popen(',cmd,')']);
  end

  % We feed the audio to the encoder in blocks of <blksize> frames.
  % By adapting this loop, you can create your own code to
  % write a single, large, MP3 file one part at a time.

  blksiz = 10000;

  nrem = nfrm;
  base = 0;

  while nrem > 0
    thistime = min(nrem, blksiz);
    done = popenw(p,32767*D(:,base+(1:thistime)),'int16be');
    nrem = nrem - thistime;
```

```
    base = base + thistime;
    %disp(['done=',num2str(done)]);
  end

  % Close pipe
  popenw(p,[]);

else
  disp('Warning: popenw not available, writing temporary file');

  tmpfile = fullfile(tmpdir,['tmp',num2str(round(1000*rand(1))),'.wav']);

  wavwrite(D',SR,tmpfile);

  cmd = ['"',lame,'"', lameopts, '"',tmpfile, '" "', FILE, '"'];

  mysystem(cmd);

  % Delete tmp file
  mysystem([rmcmd, ' "', tmpfile,'"']);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function w = mysystem(cmd)
% Run system command; report error; strip all but last line
[s,w] = system(cmd);
if s ~= 0
  error(['unable to execute ',cmd,' (',w,')']);
end
% Keep just final line
w = w((1+max([0,findstr(w,10)])):end);
% Debug
%disp([cmd,' -> ','*',w,'*']);



function Hash = DataHash(Data, Opt)
% DATAHASH - Checksum for Matlab array of any type
% This function creates a hash value for an input of any type. The type and
% dimensions of the input are considered as default, such that UINT8([0,0])
and
% UINT16(0) have different hash values. Nested STRUCTs and CELLs are parsed
% recursively.
%
% Hash = DataHash(Data, Opt)
% INPUT:
%   Data: Array of these built-in types:
%           (U)INT8/16/32/64, SINGLE, DOUBLE, (real or complex)
%           CHAR, LOGICAL, CELL (nested), STRUCT (scalar or array, nested),
%           function_handle.
%   Opt:  Struct to specify the hashing algorithm and the output format.
%         Opt and all its fields are optional.
%         Opt.Method: String, known methods for Java 1.6 (Matlab 2009a):
```

```
%              'SHA-1', 'SHA-256', 'SHA-384', 'SHA-512', 'MD2', 'MD5'.
%            Known methods for Java 1.3 (Matlab 6.5):
%              'MD5', 'SHA-1'.
%            Default: 'MD5'.
%         Opt.Format: String specifying the output format:
%            'hex', 'HEX':      Lower/uppercase hexadecimal string.
%            'double', 'uint8': Numerical vector.
%            'base64':          Base64 encoded string, only printable ASCII
%                               characters, shorter than 'hex', no padding.
%            Default: 'hex'.
%         Opt.Input: Type of the input as string, not case-sensitive:
%            'array': The contents, type and size of the input [Data] are
%                     considered  for the creation of the hash. Nested CELLs
%                     and STRUCT arrays are parsed recursively. Empty arrays
of
%                     different type reply different hashs.
%            'file':  [Data] is treated as file name and the hash is
calculated
%                     for the files contents.
%            'bin':   [Data] is a numerical, LOGICAL or CHAR array. Only the
%                     binary contents of the array is considered, such that
%                     e.g. empty arrays of different type reply the same
hash.
%            Default: 'array'.
%
% OUTPUT:
%   Hash: String, DOUBLE or UINT8 vector. The length depends on the hashing
%         method.
%
% EXAMPLES:
% % Default: MD5, hex:
%   DataHash([])               % 7de5637fd217d0e44e0082f4d79b3e73
% % MD5, Base64:
%   Opt.Format = 'base64';
%   Opt.Method = 'MD5';
%   DataHash(int32(1:10), Opt)  % bKdecqzUpOrL4oxzk+cfyg
% % SHA-1, Base64:
%   S.a = uint8([]);
%   S.b = {{1:10}, struct('q', uint64(415))};
%   Opt.Method = 'SHA-1';
%   DataHash(S, Opt)            % ZMe4eUAp0G9TDrvSW0/Qc0gQ9/A
% % SHA-1 of binary values:
%   Opt.Method = 'SHA-1';
%   Opt.Input  = 'bin';
%   DataHash(1:8, Opt)          % 826cf9d3a5d74bbe415e97d4cecf03f445f69225
%
% NOTES:
%   Function handles and user-defined objects cannot be converted uniquely:
%   - The subfunction ConvertFuncHandle uses the built-in function FUNCTIONS,
%     but the replied struct can depend on the Matlab version.
%   - It is tried to convert objects to UINT8 streams in the subfunction
%     ConvertObject. A conversion by STRUCT() might be more appropriate.
%   Adjust these subfunctions on demand.
%
%   MATLAB CHARs have 16 bits! In consequence the string 'hello' is treated
as
```

```
%   UINT16('hello') for the binary input method. Use this to get the hash of
an
%   ASCII-string (Result as defined in RFC 1321!):
%     Opt.Method = 'MD5'; Opt.Input = 'bin';
%     DataHash(uint8('abc'), Opt);    % '900150983cd24fb0d6963f7d28e17f72'
%
%   DataHash uses James Tursa's smart and fast TYPECASTX, if it is installed:
%     http://www.mathworks.com/matlabcentral/fileexchange/17476
%   As fallback the built-in TYPECAST is used automatically, but for large
%   inputs this can be more than 100 times slower.
%
% Tested: Matlab 7.7, 7.8, 7.13, WinXP/32, Win7/64
% Author: Jan Simon, Heidelberg, (C) 2011-2015
matlab.THISYEAR(a)nMINUSsimon.de
%
% See also: TYPECAST, CAST.
% FEX:
% Michael Kleder, "Compute Hash", no structs and cells:
%   http://www.mathworks.com/matlabcentral/fileexchange/8944
% Tim, "Serialize/Deserialize", converts structs and cells to a byte stream:
%   http://www.mathworks.com/matlabcentral/fileexchange/29457
% Jan Simon, "CalcMD5", MD5 only, much faster C-mex:
%   http://www.mathworks.com/matlabcentral/fileexchange/25921

% $JRev: R-v V:022 Sum:68JCxGh2/Q0N Date:30-Mar-2015 01:35:37 $
% $License: BSD (use/copy/change/redistribute on own risk, mention the
author) $
% $File: Tools\GLFile\DataHash.m $
% History:
% 001: 01-May-2011 21:52, First version.
% 007: 10-Jun-2011 10:38, [Opt.Input], binary data, complex values
considered.
% 011: 26-May-2012 15:57, Fixed: Failed for binary input and empty data.
% 014: 04-Nov-2012 11:37, Consider Mex-, MDL- and P-files also.
%      Thanks to David (author 243360), who found this bug.
%      Jan Achterhold (author 267816) suggested to consider Java objects.
% 016: 01-Feb-2015 20:53, Java heap space exhausted for large files.
%      Now files are process in chunks to save memory.
% 017: 15-Feb-2015 19:40, Collsions: Same hash for different data.
%      Examples: zeros(1,1) and zeros(1,1,0)
%                complex(0) and zeros(1,1,0,0)
%      Now the number of dimensions is included, to avoid this.
% 022: 30-Mar-2015 00:04, Bugfix: Failed for strings and [] without
TYPECASTX.
%      Ross found these 2 bugs, which occur when TYPECASTX is not installed.
%      If you need the base64 format padded with '=' characters, adjust
%      fBase64_enc as you like.

% OPEN BUGS:
% Nath wrote:
% function handle refering to struct containing the function will create
% infinite loop. Is there any workaround ?
% Example:
%   d= dynamicprops();
%   addprop(d,'f');
%   d.f= @(varargin) struct2cell(d);
```

```
%   DataHash(d.f) % infinite loop

% Main function:
================================================================
% typecastx creates a shared data copy instead of the deep copy as Matlab's
% TYPECAST - for a [1000x1000] DOUBLE array this is 100 times faster!
persistent usetypecastx
if isempty(usetypecastx)
   % Java is needed:
   if ~usejava('jvm')
      Error_L('NoJava', 'Java is required.');
   end

   usetypecastx = ~isempty(which('typecastx'));  % Run the slow WHICH once
only
end

% Default options: ------------------------------------------------------
---
Method    = 'MD5';
OutFormat = 'hex';
isFile    = false;
isBin     = false;

% Check number and type of inputs: --------------------------------------
---
nArg = nargin;
if nArg == 2
   if isa(Opt, 'struct') == 0   % Bad type of 2nd input:
      Error_L('BadInput2', '2nd input [Opt] must be a struct.');
   end

   % Specify hash algorithm:
   if isfield(Opt, 'Method')
      Method = upper(Opt.Method);
   end

   % Specify output format:
   if isfield(Opt, 'Format')
      OutFormat = Opt.Format;
   end

   % Check if the Input type is specified - default: 'array':
   if isfield(Opt, 'Input')
      if strcmpi(Opt.Input, 'File')
         isFile = true;
         if ischar(Data) == 0
            Error_L('CannotOpen', '1st input FileName must be a string');
         end

      elseif strncmpi(Opt.Input, 'bin', 3)  % Accept 'binary' also
         isBin = true;
         if (isnumeric(Data) || ischar(Data) || islogical(Data)) == 0 || ...
               issparse(Data)
            Error_L('BadDataType', ...
```

```
               '1st input must be numeric, CHAR or LOGICAL for binary
input.');
         end
      end
   end

elseif nArg ~= 1  % Bad number of arguments:
   Error_L('BadNInput', '1 or 2 inputs required.');
end

% Create the engine: ------------------------------------------------------
---
try
   Engine = java.security.MessageDigest.getInstance(Method);
catch
   Error_L('BadInput2', 'Invalid algorithm: [%s].', Method);
end

% Create the hash value: --------------------------------------------------
---
if isFile
   % Check existence of file:
   Found = FileExist(Data);
   if ~Found
      Error_L('FileNotFound', 'File not found: %s.', Data);
   end

   % Open the file:
   FID = fopen(Data, 'r');
   if FID < 0
      Error_L('CannotOpen', 'Cannot open file: %s.', Data);
   end

   % Read file in chunks to save memory and Java heap space:
   Chunk         = 1e6;
   [Data, Count] = fread(FID, Chunk, '*uint8');
   Engine.update(Data);
   while Count == Chunk
      [Data, Count] = fread(FID, Chunk, '*uint8');
      Engine.update(Data);
   end
   fclose(FID);

   % Calculate the hash:
   if usetypecastx
      Hash = typecastx(Engine.digest, 'uint8');
   else
      Hash = typecast(Engine.digest, 'uint8');
   end

elseif isBin             % Contents of an elementary array, type tested
already:
   if isempty(Data)      % Nothing to do, Engine.update fails for empty
input!
      if usetypecastx    % Bugfix: Consider missing typecastx
```

```
         Hash = typecastx(Engine.digest, 'uint8');
      else
         Hash = typecast(Engine.digest, 'uint8');
      end

   elseif usetypecastx   % Faster typecastx:
      if isreal(Data)
         Engine.update(typecastx(Data(:), 'uint8'));
      else
         Engine.update(typecastx(real(Data(:)), 'uint8'));
         Engine.update(typecastx(imag(Data(:)), 'uint8'));
      end
      Hash = typecastx(Engine.digest, 'uint8');

   else                  % Matlab's TYPECAST is less elegant:
      if isnumeric(Data)
         if isreal(Data)
            Engine.update(typecast(Data(:), 'uint8'));
         else
            Engine.update(typecast(real(Data(:)), 'uint8'));
            Engine.update(typecast(imag(Data(:)), 'uint8'));
         end
      elseif islogical(Data)               % TYPECAST cannot handle LOGICAL
         Engine.update(typecast(uint8(Data(:)), 'uint8'));
      elseif ischar(Data)                  % TYPECAST cannot handle CHAR
         Engine.update(typecast(uint16(Data(:)), 'uint8'));
         % Bugfix: Line removed
      end
      Hash = typecast(Engine.digest, 'uint8');
   end

elseif usetypecastx  % Faster typecastx:
   Engine = CoreHash_(Data, Engine);
   Hash   = typecastx(Engine.digest, 'uint8');

else                 % Slower built-in TYPECAST:
   Engine = CoreHash(Data, Engine);
   Hash   = typecast(Engine.digest, 'uint8');
end

% Convert hash specific output format: --------------------------------------
---
switch OutFormat
   case 'hex'
      Hash = sprintf('%.2x', double(Hash));
   case 'HEX'
      Hash = sprintf('%.2X', double(Hash));
   case 'double'
      Hash = double(reshape(Hash, 1, []));
   case 'uint8'
      Hash = reshape(Hash, 1, []);
   case 'base64'
      Hash = fBase64_enc(double(Hash));
   otherwise
      Error_L('BadOutFormat', ...
         '[Opt.Format] must be: HEX, hex, uint8, double, base64.');
```

```
end

% return;

%
**********************************************************************
*
function Engine = CoreHash_(Data, Engine)
% This method uses the faster typecastx version.

% Consider the type and dimensions of the array to distinguish arrays with
the
% same data, but different shape: [0 x 0] and [0 x 1], [1,2] and [1;2],
% DOUBLE(0) and SINGLE([0,0]):
% <  v016: [class, size, data]. BUG! 0 and zeros(1,1,0) had the same hash!
% >= v016: [class, ndim, size, data]
Engine.update([uint8(class(Data)), ...
               typecastx([ndims(Data), size(Data)], 'uint8')]);

% Special treatment for sparse arrays - store the indices at first and the
% values afterwards:
if issparse(Data)
   % Replace Data by vector of non-zero elements:
   [i1, i2, Data] = find(Data);
   Engine.update(typecastx(i1, 'uint8'));
   Engine.update(typecastx(i2, 'uint8'));
end

if isstruct(Data)                      % Hash for all array elements and
fields:
   F      = sort(fieldnames(Data));    % Ignore order of fields
   Engine = CoreHash_(F, Engine);      % Consider the fieldnames

   for iS = 1:numel(Data)              % Loop over elements of struct array
      for iField = 1:length(F)         % Loop over fields
         Engine = CoreHash_(Data(iS).(F{iField}), Engine);
      end
   end

elseif iscell(Data)                    % Get hash for all cell elements:
   for iS = 1:numel(Data)
      Engine = CoreHash_(Data{iS}, Engine);
   end

elseif isnumeric(Data) || islogical(Data) || ischar(Data)
   if isempty(Data) == 0
      if isreal(Data)                  % TRUE for LOGICAL and CHAR also:
         Engine.update(typecastx(Data(:), 'uint8'));
      else                             % Complex input:
         Engine.update(typecastx(real(Data(:)), 'uint8'));
         Engine.update(typecastx(imag(Data(:)), 'uint8'));
      end
   end

elseif isa(Data, 'function_handle')
```

```
   Engine = CoreHash_(ConvertFuncHandle(Data), Engine);

elseif (isobject(Data) || isjava(Data)) && ismethod(Data, 'hashCode')
   Engine = CoreHash_(Data.hashCode, Engine);

else  % Most likely this is a user-defined object:
   try
      Engine = CoreHash_(ConvertObject(Data), Engine);
   catch
      warning(['JSimon:', mfilename, ':BadDataType'], ...
         ['Type of variable not considered: ', class(Data)]);
   end
end

% return;

%
**************************************************************************
*
function Engine = CoreHash(Data, Engine)
% This methods uses the slower TYPECAST of Matlab
% See CoreHash_ for comments.

Engine.update([uint8(class(Data)), ...
               typecast([ndims(Data), size(Data)], 'uint8')]);

if isstruct(Data)                      % Hash for all array elements and
fields:
   F      = sort(fieldnames(Data));    % Ignore order of fields
   Engine = CoreHash(F, Engine);       % Catch the fieldnames
   for iS = 1:numel(Data)              % Loop over elements of struct array
      for iField = 1:length(F)         % Loop over fields
         Engine = CoreHash(Data(iS).(F{iField}), Engine);
      end
   end
elseif iscell(Data)                    % Get hash for all cell elements:
   for iS = 1:numel(Data)
      Engine = CoreHash(Data{iS}, Engine);
   end
elseif isempty(Data)
elseif isnumeric(Data)
   if isreal(Data)
      Engine.update(typecast(Data(:), 'uint8'));
   else
      Engine.update(typecast(real(Data(:)), 'uint8'));
      Engine.update(typecast(imag(Data(:)), 'uint8'));
   end
elseif islogical(Data)                 % TYPECAST cannot handle LOGICAL
   Engine.update(typecast(uint8(Data(:)), 'uint8'));
elseif ischar(Data)                    % TYPECAST cannot handle CHAR
   Engine.update(typecast(uint16(Data(:)), 'uint8'));
elseif isa(Data, 'function_handle')
   Engine = CoreHash(ConvertFuncHandle(Data), Engine);
elseif (isobject(Data) || isjava(Data)) && ismethod(Data, 'hashCode')
   Engine = CoreHash(Data.hashCode, Engine);
else  % Most likely a user-defined object:
```

```
   try
      Engine = CoreHash(ConvertObject(Data), Engine);
   catch
      warning(['JSimon:', mfilename, ':BadDataType'], ...
         ['Type of variable not considered: ', class(Data)]);
   end
end

% return;

%
**************************************************************************
*
function FuncKey = ConvertFuncHandle(FuncH)
%   The subfunction ConvertFuncHandle converts function_handles to a struct
%   using the Matlab function FUNCTIONS. The output of this function changes
%   with the Matlab version, such that DataHash(@sin) replies different
hashes
%   under Matlab 6.5 and 2009a.
%   An alternative is using the function name and name of the file for
%   function_handles, but this is not unique for nested or anonymous
functions.
%   If the MATLABROOT is removed from the file's path, at least the hash of
%   Matlab's toolbox functions is (usually!) not influenced by the version.
%   Finally I'm in doubt if there is a unique method to hash function
handles.
%   Please adjust the subfunction ConvertFuncHandles to your needs.

% The Matlab version influences the conversion by FUNCTIONS:
% 1. The format of the struct replied FUNCTIONS is not fixed,
% 2. The full paths of toolbox function e.g. for @mean differ.
FuncKey = functions(FuncH);

% ALTERNATIVE: Use name and path. The <matlabroot> part of the toolbox
functions
% is replaced such that the hash for @mean does not depend on the Matlab
% version.
% Drawbacks: Anonymous functions, nested functions...
% funcStruct = functions(FuncH);
% funcfile   = strrep(funcStruct.file, matlabroot, '<MATLAB>');
% FuncKey    = uint8([funcStruct.function, ' ', funcfile]);

% Finally I'm afraid there is no unique method to get a hash for a function
% handle. Please adjust this conversion to your needs.

% return;

%
**************************************************************************
*
function DataBin = ConvertObject(DataObj)
% Convert a user-defined object to a binary stream. There cannot be a unique
% solution, so this part is left for the user...

% Perhaps a direct conversion is implemented:
```

```
DataBin = uint8(DataObj);

% Or perhaps this is better:
% DataBin = struct(DataObj);

% return;

%
**************************************************************************
*
function Out = fBase64_enc(In)
% Encode numeric vector of UINT8 values to base64 string.
% The intention of this is to create a shorter hash than the HEX format.
% Therefore a padding with '=' characters is omitted on purpose.

Pool = [65:90, 97:122, 48:57, 43, 47];  % [0:9, a:z, A:Z, +, /]
v8   = [128; 64; 32; 16; 8; 4; 2; 1];
v6   = [32, 16, 8, 4, 2, 1];

In  = reshape(In, 1, []);
X   = rem(floor(In(ones(8, 1), :) ./ v8(:, ones(length(In), 1))), 2);
Y   = reshape([X(:); zeros(6 - rem(numel(X), 6), 1)], 6, []);
Out = char(Pool(1 + v6 * Y));

% return;

%
**************************************************************************
*
function Ex = FileExist(FileName)
% A more reliable version of EXIST(FileName, 'file'):
dirFile = dir(FileName);
if length(dirFile) == 1
   Ex = ~(dirFile.isdir);
else
   Ex = false;
end

% return;

%
**************************************************************************
*
function Error_L(ID, varargin)

error(['JSimon:', mfilename, ':', ID], ['*** %s: ', varargin{1}], ...
   mfilename, varargin{2:nargin - 1});

% return;
```