MP3 STEGANOGRAPHY AND STEGANALYSIS

BY

RAGHU JAYAN MENON

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2009

MASTER OF SCIENCE THESIS

OF

RAGHU JAYAN MENON

APPROVED:

Thesis Committee:

Major Professor _____

_____

_____

_____

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2009

ABSTRACT

This thesis involves research in the field of MP3 steganography and steganalysis. Steganography is the technique of hiding data in a medium in an oblivious manner. Steganalysis is the detection of the presence of steganographic content in carrier. A new and novel method of MP3 steganography is proposed with emphasis on increasing the steganographic capacity of the carrier medium, MP3 in this case. An interesting problem in the field of steganography is achieving optimal trade-off between attaining a high capacity to hide data while making the noise introduced in the carrier indiscernible. The work presented on the development of a new MP3 steganographic technique focuses on attaining high capacity as compared to the existing MP3 steganographic tools. The tool called BvSteg achieves 4 times the capacity of MP3Stego and UnderMP3Cover while introducing comparable noise artifacts in the carrier. The technique is novel with its approach of using Huffman codes of quantized MDCT (Modified Discrete Cosine Transform) coefficients to represent the bit to hide. The modified discrete cosine transform (MDCT) is a Fourier-related transform based on the type-IV discrete cosine transform (DCT-IV), with the additional property of being lapped: it is designed to be performed on consecutive blocks of a larger dataset, where subsequent blocks are overlapped so that the last half of one block coincides with the first half of the next block[1].

The second part of this thesis deals with MP3 steganalysis. MP3 steganalysis analyzes MP3 files for possible presence of steganographic content. MP3Stego and UnderMP3Cover being the only known steganographic tools. Work in this field by Westfeld [1] [2] has helped in detecting these tools with a high level of confidence. Bohme and Westfeld have in addition worked on the problem of MP3 encoder classification which involves classifying a MP3 file based on the encoder used to

[1]Source:wikipedia

produce it. This acts as a filter to the steganalysis stage of the tool described.

ACKNOWLEDGMENTS

I would like to thank to Dr. Victor Fay-Wolfe for his encouragement and support over the years as my advisor. He gave me the freedom to explore and trusted my abilities. His guidance on the practical aspects of research and the work presented here has been critical. I would like to thank Dr. Lutz Hamel for his invaluable suggestions on machine learning techniques, in particular the knowledge I gained in support vector machines through his classes, his book and quick responses to my E-mails. I also thank him for his support and careful reading of my work. I would like to thank Dr. Peter Swaszek for accepting my request to join the defense committee. As an external member of my committee I thank him for his interest, as well as careful examination of my work. I would like to thank Dr. Stuart Westin for accepting the role as the chair of my defense committee. I would like to thank Dr. Andreas Westfeld for his support and responses to my frantic E-mails with regards to his papers.

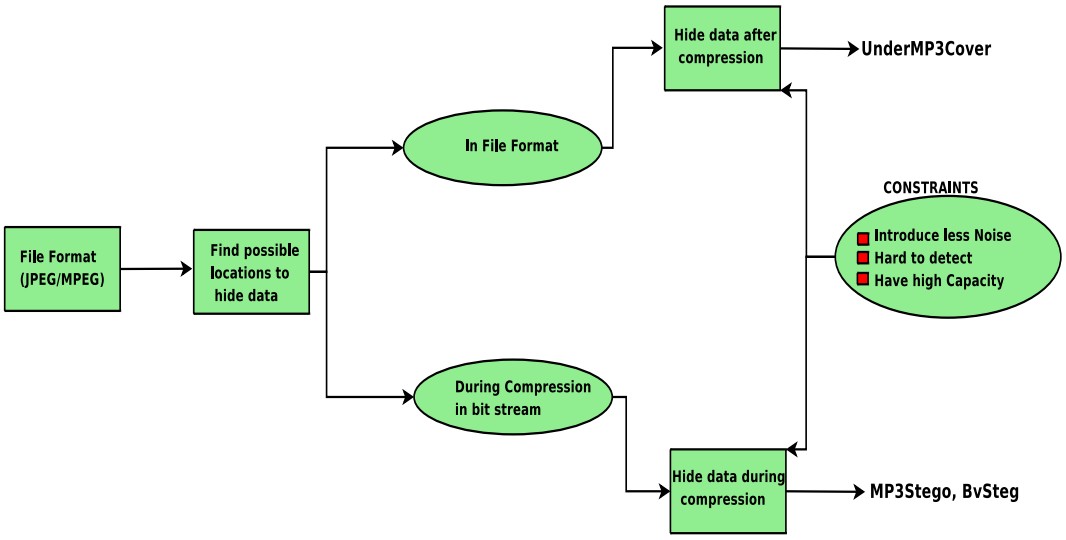
I would like to thank Kevin Bryan, who helped me shape my research in more ways than one. Kevin has been instrumental in providing ideas, technical and moral support. Kevin and I have had many fruitful discussions throughout the course of this work. His direct and indirect impact has been critical to the success of my work. I would like to thank Neil Bennett for his suggestions and careful reading of my work. Neil and I have had many a discussions, a few frustrating ones when it comes to the relevance of steganography. The discussions helped me see both the sides.

I would like to thank everyone at the computer science department for having given me an opportunity to study and work at the University of Rhode Island. Finally, I would like to thank my parents, sister and brother for their patience, understanding, encouragement and unyielding support over the years.

PREFACE

This thesis is written in a manuscript format, and investigates the issues related to MP3 (MPEG I/II Layer III) steganography and steganalysis. Steganography is the technique of hiding data in a medium without raising suspicions about the embedding. Steganalysis is the science of analyzing the cover media for the presence of hidden data. Steganographic techniques predate the evolution of multimedia and computers in general. With the advent of various multimedia formats of JPEG (Joint Photographic Experts Group), MPEG (Motion Picture Experts Group) to store image, video and audio data steganography has created its own niche in secure digital multimedia based communication. Almost all the digital steganographic techniques exploit the lossy aspect of the compressed formats. Lossy formats like JPEG and MPEG attenuate data that is not perceptually relevant. A general methodology to follow in building a steganographic tool for a multimedia format is shown in figure on page vi.

Manuscript 1 of the thesis involves MP3 steganography. The nascent nature of MP3 based steganographic techniques is evident from the number of tools available for the purpose. The work analyzes the existing MP3 steganographic tools MP3Stego and UnderMP3Cover in terms of the techniques employed to hide data along with the capacity and noise introduced. In the process the work exposes a bug in the MP3Stego hiding technique that results in the process hanging. Both the tools have identical payload capacity bounds though MP3Stego is theoretical since it involves encryption for security purposes which reduces its payload capacity. The BvSteg tool proposed in the work is a MP3 steganographic tool that hides data in the quantized MDCT coefficients. In terms of capacity the BvSteg tool exceeds that of MP3Stego and UnderMP3Cover by a factor of nearly 4. In addition safeguards to prevent perceivable noise distortion have been put



into the BvSteg tool by limiting the data hiding to region2 in the bigvalue region of the longblocks. The higher frequency ranges in region2 as a result of the MDCT compaction property provide good cover in terms of imperceptibility of the noise patterns introduced by the data hiding. In addition, the hiding technique uses Huffman pair swaps to hide data based on the magnitude relationships among pairs of quantized MDCT coefficients. Analysis of the noise introduced in the original signals reveals that BvSteg is comparable in terms of the noise introduced in the carrier with MP3Stego and UnderMP3Cover. BvSteg employs SHA1 hash algorithm to hash a user given passphrase to generate the seed for a pseudo-random number generator. A pseudorandom number generator (PRNG) is an algorithm for generating a sequence of numbers that approximates the properties of random numbers [2]. The bits from the pseudo random generator determine which blocks to embed and which ones to skip. Introducing randomness using a passphrase enhances the tool security. The detectability of this technique has not been studied even though the Huffman pair swaps ensure that the changes to the cover data are very similar to that using LSB (Least Significant Bit) hiding which is hard to

[2]Source:wikipedia

detect.

Manuscript 2 of the thesis deals with MP3 steganalysis. The work is primarily an implementation of the methods put forth by Westfeld in his papers [1] [2] in detecting MP3Stego and UnderMP3Cover. Machine learning techniques, primarily support vector machines (SVM) are used for the step of encoder classification. MP3 encoders are software that convert a wav file [3] to MPEG I/II Layer III format (MP3). MP3 files can achieve a compression ratio of 1/12. Even though MP3 technology is patented, no single party owns it wholly. With the intent of achieving speed and high audio quality MP3 encoders have mushroomed over the years. The first step in building the steganalysis tool involves MP3 encoder classification using a multi class SVM.

We thus use SVMs after evaluating the suitability for the purpose of encoder classification. To build statistically significant models for encoder classification bootstrapping was performed with 200 samples of the original data with optimal parameters to obtain the 95% confidence interval for accuracy. An overall accuracy of 90.47% was achieved with regards to classifying the MP3 files to the appropriate encoder class using a polynomial kernel of degree 2. The error rate of 9.53% is solely attributed to the misclassification of 8Hz and SoloH. The encoder classification is succeeded by the steganalysis step. The only files that are passed onto the steganalysis stage are the ones that are encoded using 8Hz and SoloH. MP3 steg tools MP3Stego and UnderMP3Cover are built on top of the open source 8Hz encoder. One of the objectives of the encoder classification is to be able to reduce the false negatives during the steganalysis stage. To achieve this the inputs to this stage are limited to those files that are classified as either 8Hz or SoloH. MP3Stego detection is implemented using QDA (Quadratic Discriminant Analysis) as the classifier with the auto-regression coefficients β_0, β_1 and β_2 over the block

[3] Microsoft, IBM file format

lengths as attributes. The classifier separates the files encoded using 8Hz and MP3Stego perfectly. This perfect classification is attainable due to the larger variance observed in the block length in MP3Stego as opposed to 8Hz. The variance is a result of the hiding scheme used in MP3Stego which modifies the block length to obtain a bit parity which is the same as the bit to hide. UnderMP3Cover detection is worked into the tool by incorporating the updet program written by Westfeld for the purpose [2].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

MANUSCRIPT 1

BvSteg - A High Capacity MP3 Steganographic Tool using Spectral Pair Swaps in Bigvalue Region of Longblocks

Abstract

Steganography is the technique of hiding information in plain sight. Digital steganographic techniques embed data in multimedia and files with various formats such that a warden perceives the file as "normal". The advent of compression techniques for image, audio and video data has also given rise to avenues galore, for hiding data in these formats. This paper presents a new technique for hiding data in MPEG I/II layer III compressed audio files. The technique has a higher capacity as compared to the existing methods used in MP3Stego and UnderMP3Cover for hiding data in MP3 files. The steganographic method proposed hides data in the bigvalue region of long blocks by modifying pairs of spectral values before they are Huffman coded. Further, the technique reduces the noise introduced by embedding data in region2 of the bigvalue region. Region2 holds spectral information in the high frequency range (5-14 KHz at 44.1 KHz sampling rate), which as per the psychoacoustic model would have low amplitude values, thus introducing lower noise in the carrier when perturbed.

1.1 Introduction

Steganography or data hiding is the technique of embedding a message (payload) in a medium (carrier), without causing suspicion about the existence of hidden data in the medium. The perturbations to the medium are carried out in such a manner that there is no perceivable noise component introduced.

One way to illustrate the concept of steganography would be to analyze Sim-

mons' Prisoners' problem [3]. Two prisoners are allowed to communicate through a medium via an agent trusted by the warden. The prisoners' are discouraged from discussing any plans of an escape from the prison. The warden himself though has a vested interest in letting them communicate as he wants to catch them in the act of hatching an escape plan or by foiling their plans by modifying the message itself. In the case of a passive warden a cryptographic technique would have worked. In this case which involves an active warden however, the message needs to look innocuous and hence cryptography fails. Steganography comes to the prisoners' rescue. The prisoners', with a strong intention of planning an escape have already exchanged a codeword before they were captured. They use this codeword to secretly exchange messages in the process deceiving the warden by hiding the message in plain sight. The codeword lets them embed and extract information. A possible technique would be to use the codeword as a position compass for hiding and extracting letters from the message exchanged. The warden is oblivious to the existence of a secret message. The medium mentioned in the problem above could be photographically produced microdots used by espionage agents during World War II, a Bacon cipher that uses different typefaces to hide information or a digitally altered JPEG image file using Steghide [1]. In all the above mentioned methods the priority is to hide messages in plain sight and make the carrier look innocuous.

Digital steganography often uses compressed/uncompressed image, video and audio formats. Image steganography has grown in prominence with tools like Outguess [4], F5 [5], Steghide [6], to name a few. Compressed audio formats like MP3 and Ogg lag in their usage as a medium for steganography. The only known steg tools that use MP3 as a carrier are MP3Stego [7] and UnderMP3Cover [8]. MP3Stego hides data into a MP3 file during the encoding process. The technique

[1] An open source steganography tool.

uses power of parity [9] to embed a bit in the part2_3_length[2] of a granule in a MP3 file. The desired value of part2_3_length is obtained in the inner_loop that quantizes the input data (spectral data) by increasing the quantizer step size until the quantized data can be encoded using the available number of bits. The additional condition that hides the data bit is the check on parity of the part2_3_length variable. If the parity is the same as the bit to be hidden the loop exits. The outer loop checks if the bound put on the quantization noise (that gets introduced in the inner_loop) has been breached. UnderMP3Cover [8] is a steg tool that embeds data by applying LSB steganography on global_gain[3] parameter in a MP3 granule. LSB or **L**east **S**ignificant **B**it steganography as the name suggests embeds data in the least significant bit of a carrier byte. In case of UnderMP3Cover the carrier byte is the global_gain value. UnderMP3Cover works on an already encoded MP3 file unlike MP3Stego which hides data during the encoding process of Pulse Code Modulation (PCM) samples to MP3.

This paper proposes a new method of steganography in MP3 files in the big-value region of long blocks using a spectral pair swap method. The layout of the paper is as described. Section 1.2 of the paper gives an overview of the MPEG layer III audio encoding algorithm. Section 1.3 covers the existing MP3 based steg tools. Section 1.4 delves into the proposed high capacity steg technique BvSteg. Section 1.5 provides notes on the tool development along with a link to the source code. Section 1.6 of the paper discusses the noise introduced and compares the capacity of the tools. Section 1.7 concludes highlighting future work.

[2]Indicates the number of bits used for encoding part2(scalefactors) and part3(Huffman encoded data).

[3]Used to determine quantizer step size

1.2 MPEG Audio Compression

An uncompressed audio file is stored as PCM samples. The PCM samples are a digital representation of the analog waveform of an audio signal.

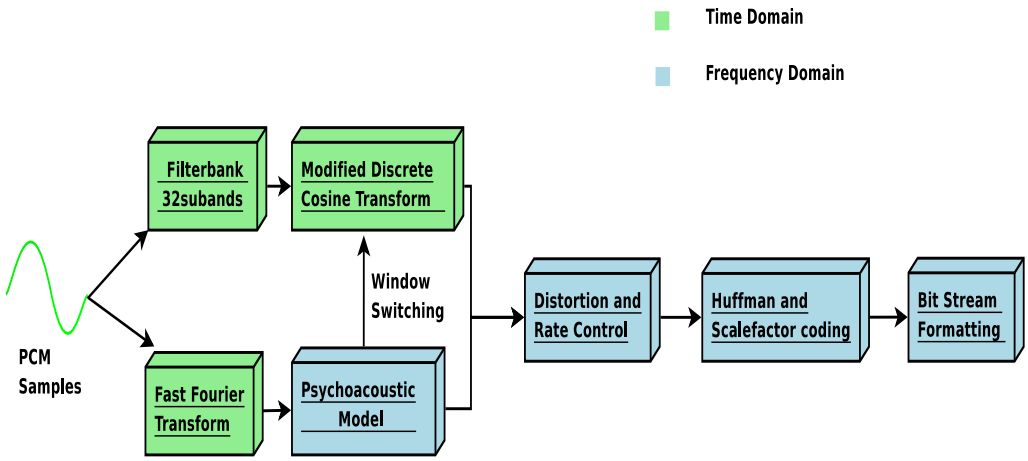


Figure 1: MP3 Encoding process

The MP3 encoding process has the following main components.

1. **Analysis filter bank**

   The analysis filter bank used for MP3 encoding consists of two cascaded filter banks. The first one is a polyphase filter bank which is the same as in Layer I/II. The polyphase filter bank serves the purpose of making Layer III backward compatible with layers I/II. The polyphase filter bank has 32 equal bandwidth filters. The input audio is critically sampled to produce 32 spectral components. Each of the 32 components is further split into 18 bands via a Modified Discrete Cosine Transform (MDCT) which is the second filter bank. The MDCT filter bank has been introduced in Layer III to provide better frequency resolution which further helps in removing possible redundant frequencies for tonal signals. This improves the coding efficiency [10]. The output of the MDCT block is a set of 576 spectral lines.

   One of the other improvements that the augmented filter bank provides is

a better control over the error signal. MP3 has two possible window sizes for analysis/coding of the signal. MP3 uses a long window with 576 samples for steady state signals, which provides good frequency resolution or 3 short windows each containing 192 samples for transient signals which provides good time resolution. The short windows get introduced when there is an "attack" (transient), since using a long window would spread the noise introduced over a wider range of adjacent frequencies. The shift from a long window to a short and vice-versa employs "start" and "stop" windows as part of the transition. The output of the analysis filter bank is a set of spectral values with the property of energy compaction introduced by MDCT. Each frame in MP3 audio has 2 granules. Each granule contains 576 spectral values.

2. **Psychoacoustic model**

   A parallel process runs alongside the analysis filter bank which first converts the time domain samples to frequency domain using the FFT and then provides the output of the Fourier transform to the psychoacoustic model. A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. A Hann window is used prior to the FFT to reduce the edge effects. The Fourier analysis provides the psychoacoustic model with the spectral change over time. Once the PCM samples are converted to the frequency domain using FFT, the psychoacoustic model runs algorithms on the data. These algorithms model the human auditory system. The algorithms provide directives on window switching to reduce noise spreading and compute the allowable distortion in scalefactor bands which closely resemble the critical bands of human hearing [11]. More importantly, it provides information on parts of audio that are audible and

inaudible. The inaudible part gets eliminated. This is the lossy part in MP3 compression process.

3. **Quantization**

   Traditional data compression techniques are employed to further compress the spectral data. The psychoacoustic analysis compresses complicated sounds better than simpler sounds. Quantization and Huffman coding are used to further enhance the compression of these simpler sounds. The 576 frequency bins are further split into 12 or 21 scalefactor bands depending on the use of short or long blocks respectively. Each scalefactor band represents a range of frequencies. The frequencies are then quantized using a non-uniform power law quantizer. Any error that is introduced in the process is what appears as quantization noise.

   The FFT analysis mentioned in the analysis filter bank has an important role to play in determining how much precision is needed in a scalefactor band. The FFT/Psychoacoustic model analyzes the signal for sounds that would be masked by neighboring sounds (masking threshold). In this case the weaker signal can be effectively scaled down without loss of perceptual quality thus reducing the number of bits needed to code that part of the signal. On the flip side when the signal is scaled back up during decoding there is noise introduced due to rounding errors introduced during the encoding process. An encoder therefore needs to keep track of when the noise introduced makes the SNR (Signal to Noise Ratio) perceptually unfavorable while at the same time keeping track of the number of bits needed to encode the part of the signal. SNR is defined as the ratio of a signal power to the noise power corrupting the signal. A reconciliation between the number of bits used to encode a granule and the noise introduced as a result of quanti-

zation is achieved though a feedback process called the outer-inner loop. The inner-loop uses Huffman coding to assign shorter codes for more frequently occurring quantized values. It computes the total number of bits required to code a block of data and checks if the number is within the bounds provided for a frame of data as determined by the sampling and bit rate[4]. If not the quantization step size is increased by increasing the global_gain. The quantization step size is changed until the required the number of bits is within the allotted bits for the frame.

The outer loop on the other hand is responsible for shaping the quantization noise according to the masking threshold that is computed by the FFT/Psychoacoustic model for each scalefactor band. The scalefactor bands that have quantization noise above the masking threshold after quantization, i.e. after the inner-loop iteration, are amplified to reduce the noise. In the process of amplification the number of bits needed to encode spectral values of the amplified bands goes up increasing the precision thus reducing the noise in these bands. Amplification of scalefactor bands also mandates a call to the inner loop to check if the bits required to encode the spectral lines is within the set bound. This process of quantization and noise shaping is an iterative process with the outer loop calling the inner loop every time the scalefactor bands are amplified.

The terminating condition arises when all the scalefactor bands have noise within the permissible limits and the number of bits used to encode the block is within the allotted value. This however is not always feasible, and hence additional conditions are used in order to terminate the iteration [12].

4. **Bit stream Formatting and Huffman Encoding**

[4]For example a 44.1 KHz, 128bit MP3 file is allotted 419 bytes per frame

Huffman codes are variable length codes. They are used in the lossless part of MP3 compression. Huffman codes are used to assign shorter codes to more frequently occurring strings and longer codes for less frequently occurring ones. MP3 encoding process makes use of 32 Huffman tables to encode quantized spectral data in various scalefactor bands. Tables 4 and 14 are never used. The quantized spectral values fall in the range [-8191, 8191]. One of the results of modelling compression based on psychoacoustics is that the resultant signal has high amplitude values associated with low frequency components. The amplitude decreases as the frequency increases. The quantized spectral values are hence arranged according to increasing frequency. Regions of spectral lines are formed according to various frequency ranges. Most of energy in the audio signal is concentrated in the 20Hz to 14KHz frequency range [13] [12]. This frequency range corresponds to the big_value region in a MP3 file. Further, the big_value region is split into 3 sub-regions with typical frequency range split up of 0-2 KHz (region0), 2-5 KHz (region1), 5-14 KHz (region2) for a MP3 file which has been sampled at 44.1 KHz. Each of the regions use a different Huffman table for encoding the quantized values. The selection of the table is done on the basis of the local region statistics of the signal.

The higher frequency components which have magnitudes of -1, 0, 1 form the count1 region. The rzero region consists of high frequency spectral values with amplitude 0. The rzero region information is not transmitted across as part of the MP3 file. The count1 region uses 2 separate Huffman tables to encode contiguous quadruples of spectral values. The big_value regions on the other hand encodes pairs of values using one of the 30 Huffman tables. The Huffman encoding tables can be found in the standard [14]. The rzero

section does not need any Huffman encoding.

The MP3 decoder uses a decode tree mechanism to decode the Huffman values and form the pairs/quads of spectral values. This is one of the reasons why decoding a MP3 file is faster than encoding PCM samples to MP3. For a more elaborate discussion on MPEG layer III encoding/decoding refer to [14] [15] [12] [10] [16] [11].

Figure 2 describes the layout of a MP3 frame [17]. Each block in the diagram indicates a size of 1 bit.

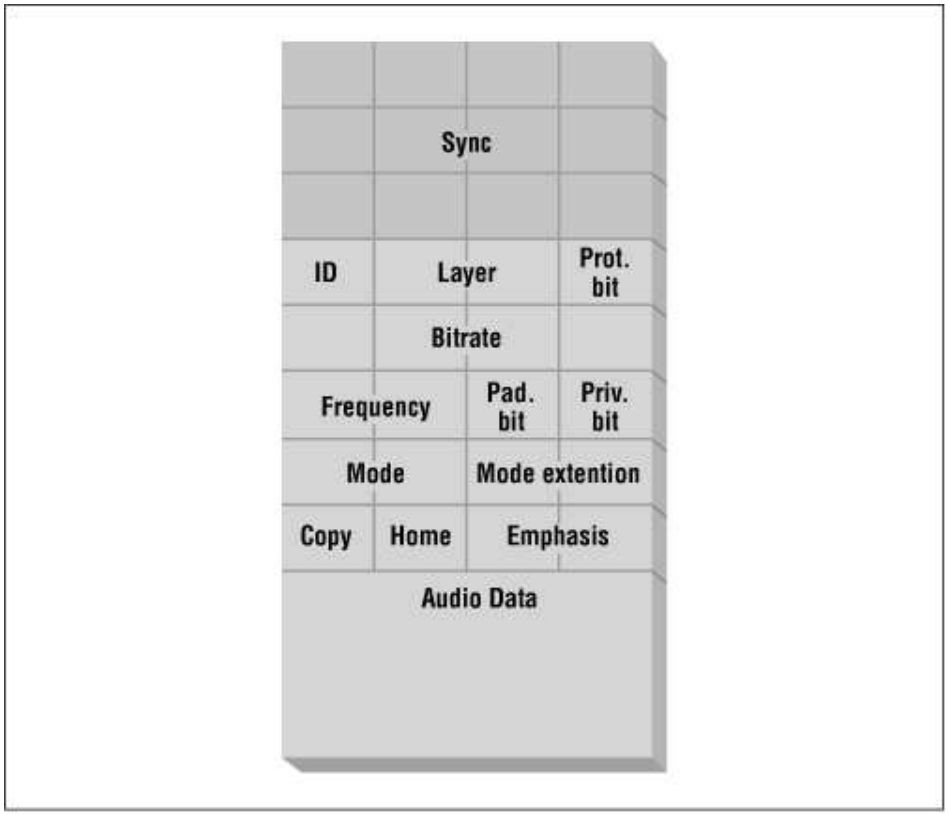


Figure 2: MP3 Frame Structure

Each frame in a MP3 bit stream is further split into 2 granules. The side information for each granule in a frame contains information needed to decode the

main data. Figure 3 describes the component fields along with the size in bits for the side information. The size represent the requirements in single channel mode as well as the double that would be needed in a dual channel mode.

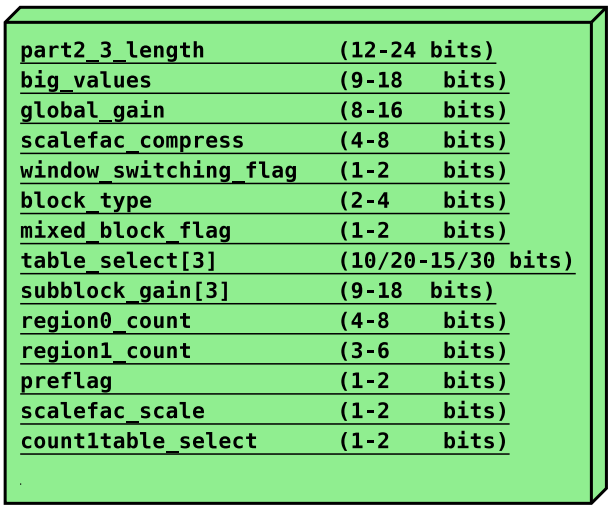


Figure 3: Side Information for each granule

1.3 Overview of existing steg techniques in MP3

As mentioned earlier MP3 as a steganographic medium is still in its infancy. One of the reasons is the terse MP3 specifications and involved MP3 encoder implementation. Any steganographic tool built using MP3 as a medium would have to be built on top of an existing encoder. As we will see the encoder of choice in the 2 tools mentioned below is 8Hz. Less surprising then would be the fact that BvSteg too is built on top of 8Hz encoder.

1.3.1 MP3Stego

MP3Stego [7] is one of the earliest MP3 based steganographic tools. Data hiding in MP3Stego is done as part of the encoding process. The tool uses the power-of-parity [9] principle to embed data in part2_3_length of a granule in a MP3 file. The part2_3_length variable indicates the total number of bits re-

quired to encode the scalefactors and the Huffman coded data. The granules to be modified are randomly chosen using SHA-1. The value and hence the parity of part2_3_length variable is modified in the inner-loop during quantization. In addition to the original condition that the number of bits used for encoding a block be within a bound the inner-loop terminates only if the parity of the variable part2_3_length is the same as the bit to be embedded . The inner-loop in the MP3Stego hiding procedure is shown in Listing 1.1.

Listing 1.1: MP3Stego inner-loop for hiding

```
do
{
    do
    {
        cod_info->quantizerStepSize += 1.0;
        quantize(xrs,ix,cod_info);
    }while(ix_max(ix,0,576)>(8191+14));              /* within table range? */

    calc_runlen(ix,cod_info);                        /* rzero,count1,big_values*/
    bits = c1bits = count1_bit_count(ix,cod_info);    /* count1_table selection*/
    subdivide(cod_info);                             /* bigvalues sfb division */
    bigv_tab_select(ix,cod_info);                    /* codebook selection*/
    bits += bvbits = bigv_bit_count(ix, cod_info);    /* bit count */

    switch(hiddenBit)
    {
        case 2:
            embedRule = 0;
            break;
        case 0:
        case 1:
            embedRule = ((bits + part2length) % 2) != hiddenBit;
            break;
        default:
            ERROR("inner_loop:_unexpected_hidden_bit.");
    }
}while((bits>max_bits) | embedRule);
```

Problems with MP3Stego

Analyzing MP3Stego gave rise to

1. **Hiding process hangs**

   The hiding process in MP3Stego hangs on occasion because of its inability to satisfy the parity condition. In cases when the inner-loop cannot encode the

spectral lines within the bit ration it sets all the spectral values to 0 [10]. In doing so the number of bits used to encode the spectral values part3 (Huffman coding) reduces to 0. In the code in Listing 1.1, **bits** would be set to 0 on Line 13 as **bigv_bit count** would return a 0 in the above said condition. In **case 1** on Line 22 in the code were the embedding occurs, the condition **((bits + part2length)) % 2 != hiddenBit** essentially reduces to **(part2length % 2) != hiddenBit**, when **bits**=0. The part2length variable which is the number of bits needed to encode the scalefactors is fixed for a granule and does not change during the processing of the inner-loop. Suppose that the inner-loop set all the spectral values to 0s and we have a 0 to embed i.e. **hiddenBit**=0. If the variable **part2length** is odd the process (do loop) will execute forever.

A crude way to overcome this problem is to change the passphrase and hence change the seed to the pseudo random number generator which is responsible for selecting the blocks that would undergo embedding. In changing the passphrase one can hope that the seed of the random number generator is changed and the granule which was causing the problem during embedding is not selected.

2. **Size constraint**

   To begin with, MP3Stego has low embedding rates. This is also not helped by the fact that the maximum capacity of $4 * number\_of\_frames$ is never achieved. The hiding capacity of the carrier is diminished by two facts,

   a. **zlib consumption**

      The overhead associated with compressing a 0 byte file with zlib result in the usage of 24 bytes [1]. This overhead reduces the capacity of the carrier.

b. **Skip random**

The logic in MP3Stego skips random blocks while embedding thereby leading to a loss in capacity.

A method of detecting files stegged using MP3Stego is discussed in [1] and is implemented in the second paper.

1.3.2 UnderMP3Cover

UnderMP3Cover [8] is a MP3 LSB steganographic tool that uses global_gain to hide data. Unlike MP3Stego, UnderMP3Cover hides data in an already encoded MP3 file. The modification is done on the LSB of global_gain variable in selected granules to reflect the embedded bit. The tool uses a spacing parameter to select the granules to embed in. MP3Stego and UnderMP3Cover have comparable data-hiding rates for a given carrier file. A method for detecting files stegged using UnderMP3Cover is discussed in [2] and is implemented in the second paper.

The maximum steg capacity of the carrier when used with MP3Stego and UnderMP3Cover is $4 * number\_of\_frames$ bits. Both the programs can embed a maximum of 4 bits in a frame if the signal is stereo since a stereo signal has 2 channels and each channel has two granules.

1.3.3 Other Methods of Data Hiding in MP3

Do-Hyoung et al. [18] discuss a method of data insertion into MP3 bitstream using linbits characteristics. As mentioned in the paper the method does not have high capacity but is good for watermarking applications. Litao Gang et al. [19] analyze data hiding schemes in amplitude domain, phase domain and also discuss a noise substitution scheme. N Moghadam and Sadeghi [20] propose a watermarking scheme in MDCT domain. They describe a genetic algorithm to select the best coefficients to embed the watermark.

Very few implementations of MP3 based steg techniques exist. In addition the techniques of watermarking though have a similar requirement of security through obscurity impose constraints on robustness which is not very essential for steganographic techniques. In addition watermarks usually have a small payload size which makes them thrifty when it comes to the payload size while steganography is more demanding in terms of the payload capacity of the carrier. These reasons make watermarking techniques usually inadequate for steganography.

1.4 BvSteg

BvSteg has almost 4 times the steganographic capacity of MP3Stego and UnderMP3Cover. The tool hides data in the big_value region of the long blocks. Embedding is carried out in region2 which for a 44.1 KHz sampling rate corresponds to the frequency range of 5-14 KHz. Due to the energy compaction properties of MDCT [21] most of the spectral energy is concentrated in region0 and region1 of the signal. Changes in region2 introduces low noise components in the signal and, hence, the perturbed audio signal as perceived by the human ear is not significantly different from the signal without the embedding. The actual algorithm used for embedding is based on the magnitude relationship between the pairs of spectral values that occur in the big_value region (region2) during the encoding process of a MP3 file.

Algorithm 1 BvSteg hiding process

```
1: procedure StegMP3(passphrase, file_to_hide)
2:     file_size ← 0
3:     hide_status ← HIDE_IN_BLOCK
4:     file_size_bit_count ← 0
5:     file_size ← GetFileSize(file_to_hide)
6:
7:     while file_size_bit_count < 32 do
8:         if hide_status == HIDE_IN_BLOCK then
9:             bit_to_embed ← file_size >> 1
10:            file_size_bit_count = file_size_bit_count + 1
11:        end if
12:        hide_status ← Hide(passphrase, bit_to_embed)
13:    end while
14:
15:    file_size_bit_count ← 0
16:    hide_status ← HIDE_IN_BLOCK
17:    while file_size_bit_count < file_size do
18:        if hide_status == HIDE_IN_BLOCK then
19:            bit_to_embed ← bit from file_to_hide
20:            file_size_bit_count = file_size_bit_count + 1
21:        end if
22:        hide_status ← Hide(passphrase, bit_to_embed)
23:    end while
24: end procedure
```

Require: First time Hide call in a longblock ⇒ *embed_count* ← 1

```
25: procedure Hide(passphrase, bit_to_embed)
```

Ensure: First time Hide call in a longblock ⇒ *embed_count* ← 1

```
26:    if (In Huffman-coding a long block) then
27:        if (In region2 of bigvalues) then
28:            if ((equivmap[x][y] == 1) && (embed_count ≤ 4)) then
29:                hide_or_skip ← GetRandBit(passphrase)
30:                if (hide_or_skip == HIDE_IN_BLOCK) then
31:                    if ((bit_to_embed == 0)&&(x < y)) then
32:                        SWAP(x, y);
33:                    else if ((bit_to_embed == 1)&&(x > y)) then
34:                        SWAP(x, y);
35:                    end if
36:                    embed_count ← embed_count + 1
37:                    return HIDE_IN_BLOCK
38:                end if
```

```
39:             end if
40:         end if
41:     end if
42:     return SKIP_BLOCK
43: end procedure
```

In Algorithm 1, the pair (x,y) represents the quantized spectral values in the bigvalue region. The bit_to_embed as the name suggests is the bit that is to be hidden. **equivmap** is a global array, one for each of the 30 tables (Refer [14]). equivmap[x][y] is set to 1 if the number of Huffman bits to encode the pair (x,y) is the same as the number of Huffman bits required to encode the pair (y,x). The routine SWAP flips the pair (x,y) . The functions GetFileSize and GetRandBit which are not defined explicitly perform the following operations. GetFileSize function returns the size of the file that is to be hidden. GetRandBit takes as the argument the passphrase that the user inputs. The passphrase is then hashed using the SHA1 algorithms and part of the hash is used to seed a pseudo random number generator. The return value of the function is the LSB of the random number generated. The embed_count is the maximum number of changes that are allowed per longblock and is limited to 4.

The number of embedded bits is restricted to 4 per granule in the long block. Our experiments show that this number is a good trade-off between high steg capacity and low noise. Embedding is skipped if the granule does not contain a long block. Within a long block embedding is further restricted to spectral value pairs in region2 so that the amount of noise introduced is minimal.

1.4.1 LSB Steganography using Spectral Pairs and Huffman Values

The flowchart in Figure 4 has each of the constraints that are enforced on the candidate pairs of spectral values in the diamond boxes. Two driving forces behind the constraints imposed on the candidate pairs for embedding are:

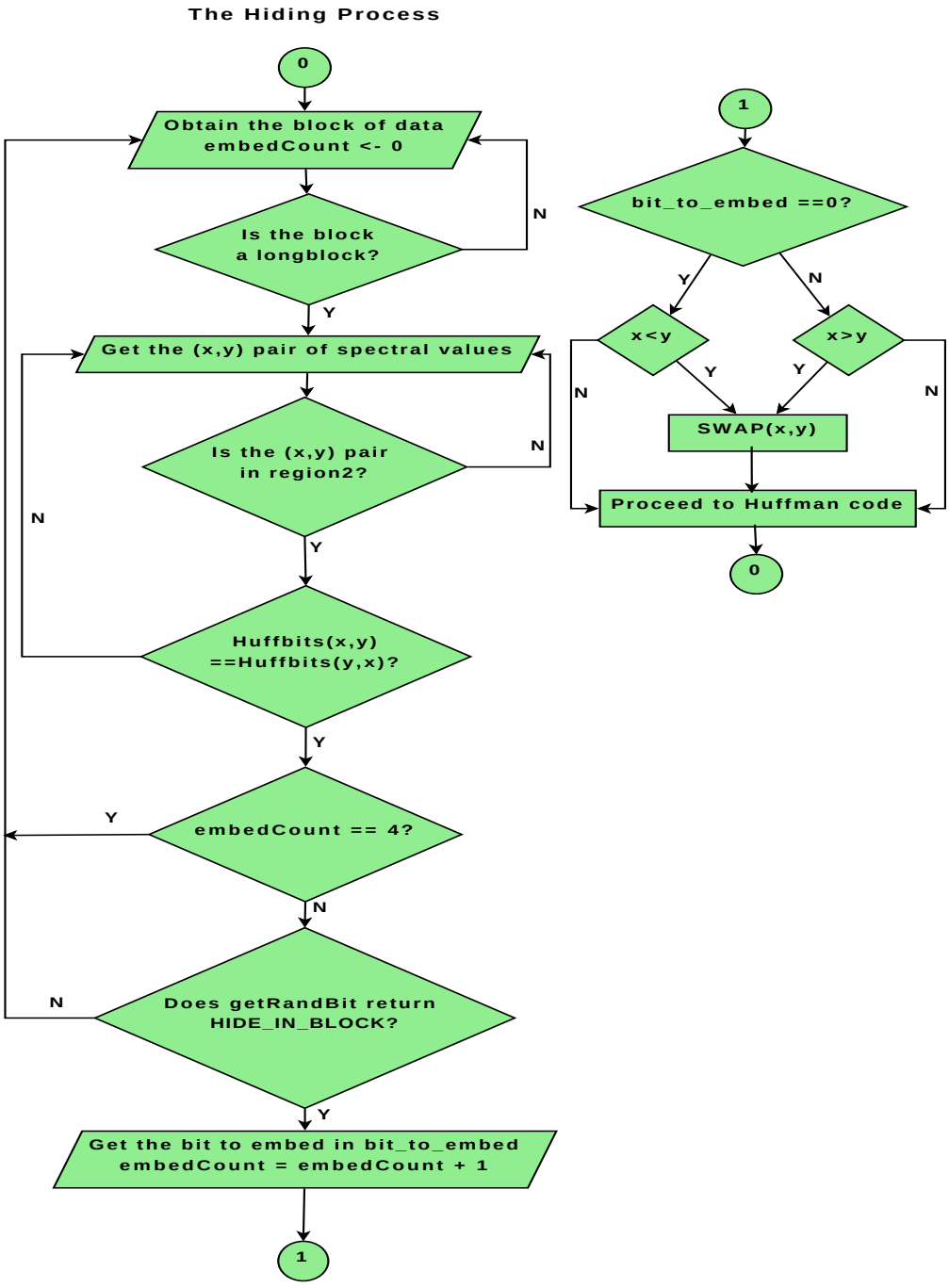


Figure 4: The bit hiding process in detail

1. **The overall bitrate**

   Constant Bit Rate (CBR) MP3 encoding imposes size constraints on a MP3 frame. The overall number of bits per frame can be computed using the formula $FrameSize = 144 * (\frac{BitRate}{SampleRate+Padding})$. For example, a MP3 file encoded at a sampling rate of 44.1 KHz and a bitrate of 128 Kbps would have a frame size of approximately 417 bytes. The distribution of bytes among the granules is encoder dependent. As discussed in section 1.2, it is the responsibility of the inner-loop to ensure that the total number of Huffman bits needed to code a block of signal data is within the allotted number for a frame which implicitly puts a limit on the granule.

   The problem that arises as a result of swapping a spectral pair (x,y) is that Huffbits[5](x,y) need not be equal to Huffbits(y,x). This causes a change in the bit count in a granule. In order to keep the total number of bits in a granule within set bounds, two strategies could be adopted.

   a. **Restricting (x,y) pairs**

      In this approach we impose a constraint of only modifying those (x,y) pairs which satisfy the condition $Huffbits(x, y) = Huffbits(y, x)$. This case typically arises if the codes for (x,y) and (y,x) are the right and left leaves of a node in the Huffman code tree. A statistic on the number of such (x,y) pairs reveals that of the 4995 (table,x,y) triplets, 3302 can be changed even with the above restriction imposed. This implies that 66% of spectral pairs over all the Huffman tables can be swapped while preserving the bit count in terms of the Huffman bits used to encode the granule.

[5]Huffbits computes the number of bits required to encode a spectral pair using Huffman codes. For example:Huffbits(1,1)=3 for table 1, which is the hlen column value, refer page 56 [14]

b. **Count1 region modification**

The second approach is to modify the count1 region. As mentioned the count1 region uses 2 Huffman tables to encode quads of contiguous values that are either -1, 0 or 1. The encoding of the count1 region follows the encoding of the big_value region. The extra bits that might have been used to encode the pair (y,x) after the swap could then be compensated by finding a suitable quad in the count1 region that can be replaced with another one with a bit count lesser by the number of extra bits used to encode the pair (y,x). This approach has the advantage of providing all the spectral pairs as candidates for swap thus increasing the data hiding capacity of the carrier. The noise introduced due to count1 region modification is also minimal as count1 consists of mostly high frequency components.

The disadvantage however is that count1 modification is not always possible. A suitable replacement for a quad may not exist in some cases or all possible substitutions might have been performed leading to a shortage of quads to change. In addition, if the distance between the pairs (x,y) is large then swapping them might introduce more noise than that desired.

As an example of count1 modification consider the scenario wherein modification (swaps) to spectral pairs in bigvalue region caused a bit excess of 4 in the granule. The extra bits would be the result of longer Huffman codes used to encode the pair (y,x) after the swap. Now suppose that table A was used to encode the quads in the count1 region. The bit compensation logic would replace 6 bit codes with 5 bit ones and 5 with 4 bits ones. If the logic was able to find 4 such substitutions

the granule encoding would have been accomplished in the stipulated bit limit. The problem however arises in cases were no such substitution is possible for example if the count1 region uses table B which only uses 4 bit codes a suitable substitution is not possible to reduce the excess bits in bigvalue region as there are no codes in the table that are shorter than 4 bits.

Even though experiments done with the count1 modification show that the noise introduced using method (b) is not discernible, due to its failings on occasion when the procedure cannot find a suitable quad replacement, approach (a) has been followed in this paper. There is however a reduction in steg capacity when compared to approach (b).

2. **Least distance between spectral pair magnitudes**

   Difference in the magnitude between the pair (x,y) was the second driving force on the constraints imposed on the candidate pairs. The pairs of spectral values represent the quantized MDCT coefficients within a scalefactor band. Swapping these values has the effect of pronouncing one of the frequencies in a band over the adjacent one. As in DCT based steganographic methods the amount of change and hence the noise introduced can be minimized by the technique of least significant bit modification. We could impose an additional constraint on the (x,y) pairs that are modified, such that $|x - y| = 1$. This constraint creates an effect **similar** to LSB steganography, but by imposing this constraint however we reduce the steg capacity of the carrier drastically. The total number of pairs that satisfy this condition with all the previous conditions in place falls to 524 thereby reducing the swappable pairs from 66% to 11%. This leads to a drastic reduction in capacity albeit with the advantage of reduced noise. Experiments however, have shown that the noise

introduced without this constraint is negligible when compared to the loss incurred in the steg capacity of the carrier with the constraint included. We therefore do not impose this restriction in the implementation. This could however be interesting for MP3 watermarking applications.

1.4.2 The Extraction Process

The data extraction process is straightforward. The pairs of spectral values that are obtained after Huffman decoding are compared for their magnitude relationship. Algorithm 2 details the extraction process. The equivmap array in the condition is checked for either (x,y) or (y,x) pair being set. The logic is self explanatory and mirrors that of the hiding process.

1.5 Tool Development

There are quite a few open source MP3 encoders available on the web. The 8Hz MP3 [22] encoder was used for the development of the tool. 8Hz source code base is also used by MP3Stego and UnderMP3Cover tools. 8Hz MP3 encoder is not the best available encoder in terms of speed and the quality of sound produced. It is however one of the earliest encoders and has been the source base for the development of numerous encoders, prominent among them is the LAME [23] encoder that started off as a patch to the 8Hz encoder to its present status as one of the prominent open source encoders.

The code change primarily involved manipulating the quantized MDCT values in the Huffmancodebits function of the l3bitstream.c file. The quantized MDCT values are compared to determine if a swap is needed to encode the hidden bit before they are passed onto the Huffman encoding function HuffmanCode. The extraction logic is implemented in the decode.c file in III_hufman_decode function. The reverse logic as mentioned in algorithm 2 is coded to extract the hidden

Algorithm 2 BvSteg- The extraction process

```
1: procedure RetrieveMesg(passphrase)
2:     retrieve_status ← GOT_BIT
3:     file_size ← GetFileSize(file_to_hide)
4:     file_size_bit_count ← 0
5:
6:     while file_size_bit_count < 32 do
7:         retrieve_status ← Extract(passphrase, 0, FILE_SIZE_BITS)
8:         if retrieve_status == GOT_BIT then
9:             file_size ← collectFileSize(bit_retrieved)
10:            file_size_bit_count += 1
11:        end if
12:    end while
13:
14:    retrieve_status ← GOT_BIT
15:    file_size_bit_count ← 0
16:
17:    while ((file_size_bit_count < file_size) &&
18:            (retrieve_status != EXTRACTION_COMPLETE)) do
19:        retrieve_status ← Extract(passphrase, file_size, MESSAGE_BITS)
20:        if retrieve_status == GOT_BIT then
21:            message ← gatherBits(bit_retrieved)
22:            file_size_bit_count += 1
23:        end if
24:    end while
25: end procedure
```

Require: First time call in a granule ⇒ $change\_per\_granule \leftarrow 1$
Require: First time call of Extract procedure ⇒ $total\_bits \leftarrow 0$

```
26: procedure Extract(passphrase, file_size, file_size_bit_or_mesg_bit)
```

Ensure: First time call in a granule ⇒ $change\_per\_granule \leftarrow 1$
Ensure: First time call of Extract procedure ⇒ $total\_bits \leftarrow 0$

```
27:    if (In Huffman-decoding a long block) then
28:        if (In region2 of bigvalues) then
29:            if (((equivmap[x][y]==1)||(equivmap[y][x]==1))) then
30:                if (change_per_granule ≤ 4) then
31:                    retrieve_or_skip ← GetRandBit(passphrase)
32:                    if then(retrieve_or_skip == RETRIEVE)
33:                        if (x < y) then
34:                            Hiddenbit ← 1;
35:                        else if (x > y) then
36:                            HiddenBit ← 0;
37:                        end if
```

```
38:                         change_per_granule ← change_per_granule + 1;
39:                         if (file_size_or_mesg_bit == MESSAGE_BITS)
    then
40:                             total_bits ← total_bits + 1;
41:                         end if
42:                     end if
43:                 end if
44:             end if
45:         end if
46:     end if
47:     if (file_size_or_mesg_bit == MESSAGE_BITS) then
48:         if total_bits == file_size then
49:             return EXTRACTION_COMPLETE
50:         end if
51:     end if
52: end procedure
```

bit based on the magnitude relationship of the quantized Huffman pairs. Helper functions were written to build a database of equivalent spectral pairs in terms of Huffman code length.

The hiding and extraction process is integrated into a Python script which calls the modified encode and decode executables of the 8Hz encoder. Bit conversion routines that convert a text file to a stream of bits for hiding and vice-versa after extraction are incorporated into the script. In addition, two helper functions in the main hiding and extraction procedures include a pseudo random generator based on SHA1 hash and a file size embedding logic. A part of the SHA1 hash of the user input passphrase is used to the seed a replicable pseudo random number generator. This provides the logic to randomly hide and skip blocks. The file size of the payload file is hidden into the first 32 randomly selected bits. This limits the size of payload to 4GB which is expected to sufficient in terms of payload capacity.

1.6 Steg Capacity and Noise Analysis

The table in Figure 5 shows the capacity of various wav files under different steg tools. The files have been selected from different genre. In addition the size of wav file indicates different duration.

Name	Size of wav	MP3Stego	UnderMP3Cover	Bvsteg
beatles3	9	1	1	3
jazz2	65	7	7	15
sting10	64	7	7	20
vanmorris7	50	5	5	17
guitar19	105	11	11	40
nayyar2	60	6	6	24

(a) Size of the wav file is in MB, Steg capacity of the MP3 files after encoding the wav files using MP3Stego, BvSteg is in KB, MP3 files with data hidden using UnderMP3Cover with spacing of 2 have size mentioned in KB.

Figure 5: Size comparison of payload capacity

In order to analyze the noise introduced as a result of stegging an MP3, the MP3 signal was analyzed using audacity [24], an open source MP3 editor. The following steps were performed as part of the analysis

1. Original MP3 signal and the stegged MP3 signal are loaded.

2. The phase of the steg signal was inverted.

3. Both signals were then mixed together to obtain the difference signal.

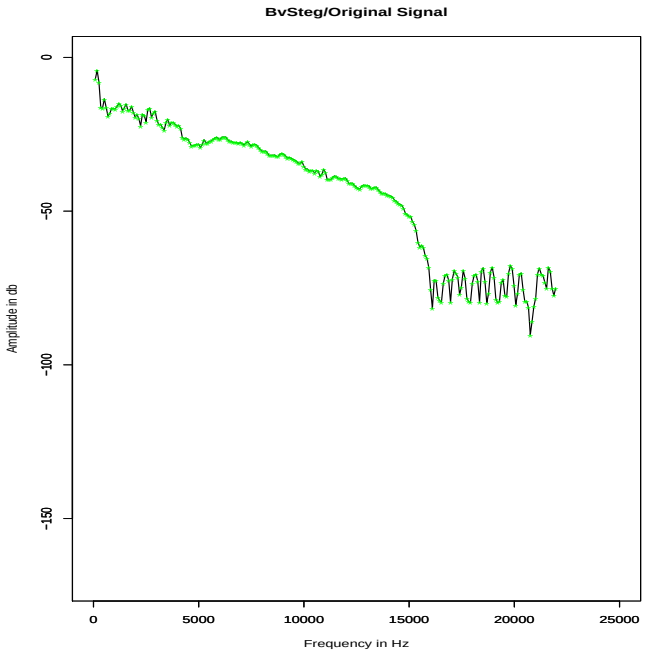
The difference signal shows the noise introduced by using a particular technique for data hiding. Also, the stegged MP3 signal was analyzed to check for the fidelity with the original signal without the hidden data when encoded with 8Hz. All steganographic methods introduce noise in the carrier. A good steg technique is one that does not introduce perceivable noise. MP3Stego introduces noise in the carrier when it changes the quantizer step size in order to meet the additional constraint in the inner-loop i.e. parity(part2_3_length) == bit_to_embed while

UnderMP3Cover introduces noise as part of modification to the global_gain value. BvSteg introduces noise due to the spectral pair swap.

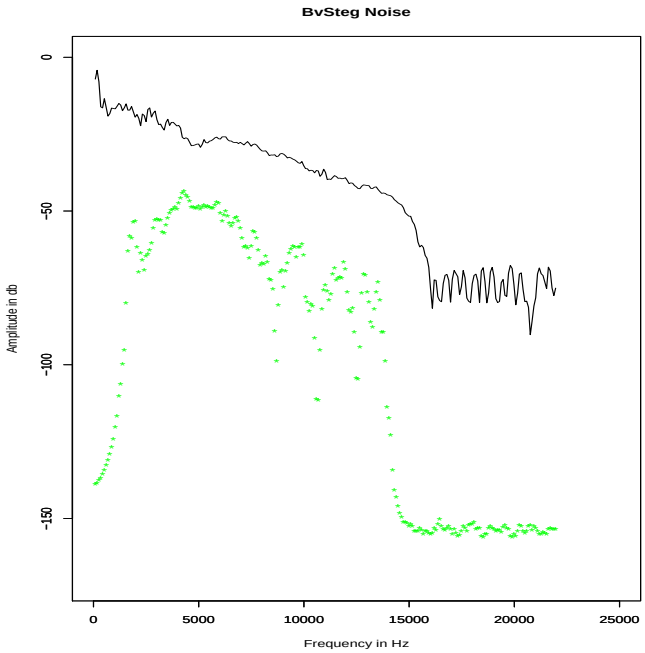
Figures 6a and 6b show the characteristics of the signal and noise respectively for a non-stegged MP3 file and the same MP3 file stegged with BvSteg. The MP3 signal follows the original signal quite faithfully. Figures 7a and 7b indicate the signal and noise plots for MP3Stego and Figures 8a and 8b indicate that of UnderMP3Cover. Though the case for all the 3 steganographic techniques look identical the BvSteg introduced noise is a result of capacity which is 4 times more than that of the other tools in displaying the same noise and signal characteristics. This shows the superiority of BvSteg over both MP3Stego and UnderMP3Cover.

Conclusion

This paper began by introducing the concept of digital steganography. The MPEG layer III audio encoding process was then illustrated highlighting the lossy aspects of the compression process. This was followed by an overview of existing steg tools and techniques. A bug in the MP3Stego tool that was caused by the inner_loop constraints was exposed. A new steg technique BvSteg was proposed. The technique is better over the existing steg concepts and tools in terms of the payload capacity. BvSteg has 4 times the capacity of both MP3Stego and UnderMP3Cover. BvSteg tool modifies the quantized MDCT coefficients in the high frequencies of the big_value region. The changes in the high frequency region are less discernible and hence don't introduce substantial noise. The feasibility of the technique was outlined in presenting the algorithms and flowchart for the embedding and extraction process. Possible methods on how to increase the capacity such as count1 modification and for lowering the noise i.e. least spectral magnitude difference were also discussed. A noise and capacity analysis presented

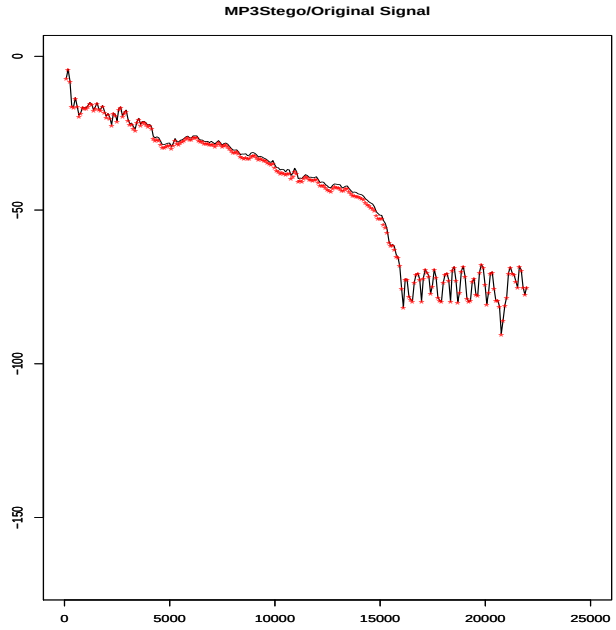


(a) Black: MP3 Signal after encoding the wav file using 8Hz, Green:MP3 signal after hiding using Bvsteg. BvSteg signal follows the original signal with high fidelity

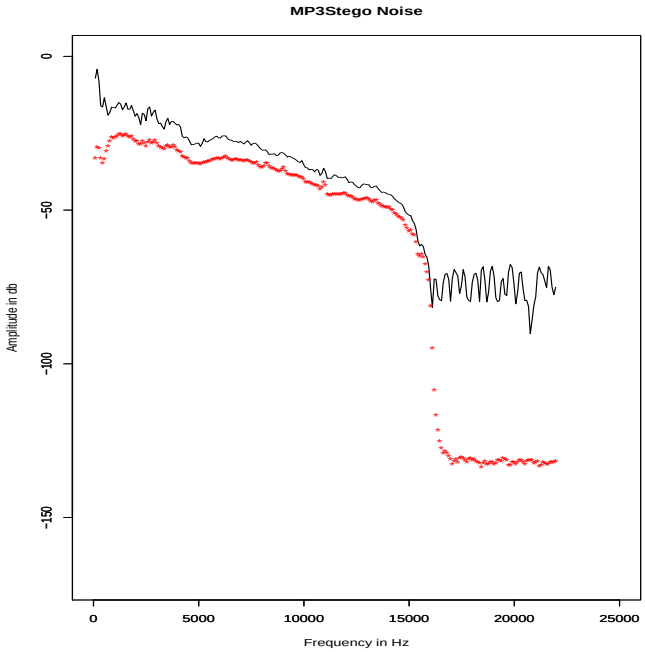


(b) Black: MP3 Signal after encoding the wav file using 8Hz, Green:Noise signal introduced after hiding using Bvsteg

Figure 6: Noise/Signal Analysis of BvSteg

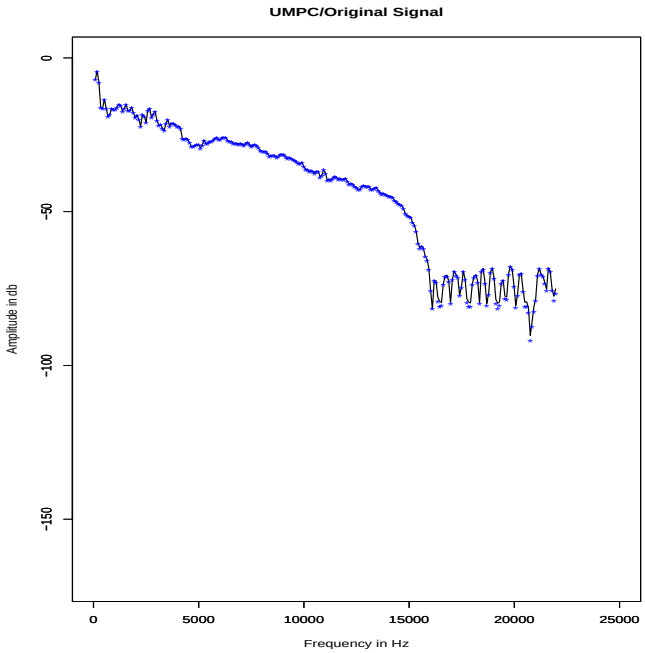


(a) Black: MP3 Signal after encoding the wav file using 8Hz, Red:MP3 signal after hiding using MP3Stego

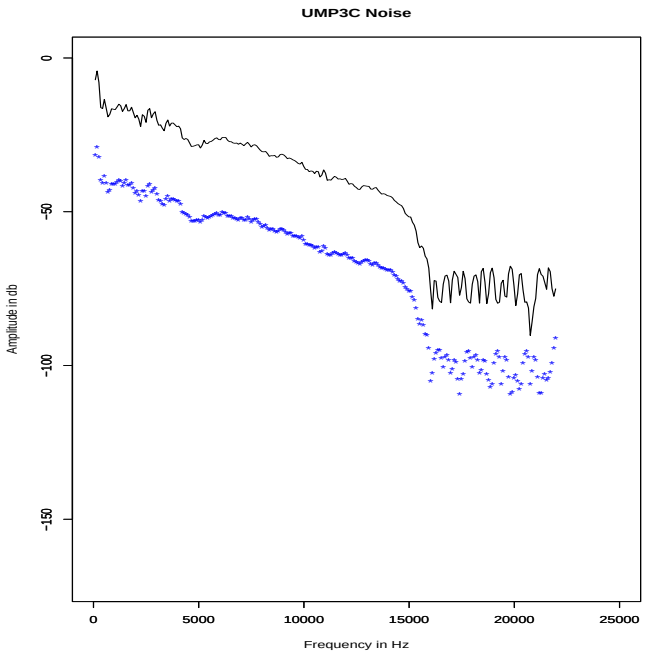


(b) Black: MP3 Signal after encoding the wav file using 8Hz, Red:Noise signal introduced after hiding using MP3Stego.

Figure 7: Noise/Signal Analysis of MP3Stego



(a) Black: MP3 Signal after encoding the wav file using 8Hz, Blue:MP3 signal after hiding using UnderMP3Cover



(b) Black: MP3 Signal after encoding the wav file using 8Hz, Blue:Noise signal introduced after hiding using UnderMP3Cover.

Figure 8: Noise/Signal Analysis of UnderMP3Cover

validates the claim of BvSteg having higher capacity as compared to the existing tools. The spectrum analysis demonstrated the noise component introduced due to the perturbations to be within the acceptable levels.

The BvSteg tool despite its high steg capacity could still improve in the manner in which the hiding algorithm selects the candidate frequency bands for hiding data. Methods for selection of noiseless bands provides scope for future work. A brief description of one such proposed methodology ensues.

One of the challenges in data insertion as a means of covert communication or watermarking is that any perturbations should have minimal impact on the signal quality. In order to achieve this we need to distinguish noisy bands from noiseless ones. The distortion control/rate loops are designed such that both the constraints of bit rate and allowed quantization noise are met. In order to meet the latter requirement the scalefactor bands that have more than allowable distortion are amplified. The amplification is done so that more number of bits are allocated during the subsequent call to the inner-loop and hence the quantization noise which was introduced earlier in the scalefactor band would be lesser. This also implies stealing bits from bands that had noise within permissible levels. The amplification is done under the assumption that the bands which have tolerable noise levels after quantization can assimilate more noise without crossing the noise threshold set by the psychoacoustic model.

It is proposed that identifying scalefactor bands in granules that are not amplified would help select frequency ranges that would introduce least amount of noise when their amplitude is modified. In addition techniques involving selection of blocks based on the number of bits in the bigvalue region as done by [18] would be an interesting approach. This provides scope for future work.

The paper has been successful in demonstrating a new high capacity steg

technique BvSteg which hides data in region2 of longblock in MP3 files during encoding. This work aim at popularizing the use of MP3 as a steganographic medium and lay foundation for development of steg tools in MDCT domain. The open nature of the tool would also serve as a case study that would aid the steganalysis of any MDCT based steganographic tools.

MANUSCRIPT 2

A Tool Framework for MP3 Steganalysis

Abstract

Steganalysis is the technique of detecting the presence of hidden data in a medium which can act as a possible carrier. Digital steganalysis employs techniques of statistical analysis and machine learning to detect hidden data in multimedia and files with various formats. Most of the steganographic techniques alter statistical characteristics of the underlying media. Steganalysis techniques employed in JPEG/MPEG domain try and detect this change to ascertain steg in the medium.

In the case of MP3 based steganography the tools are mostly built on top of an existing open source encoder. The 8Hz encoder which is one of the oldest MP3 encoders has been the encoder of choice for the steg tools discussed here. This paper proposes a framework for a tool for the detection of MP3 based steganographic tools. MP3Stego and UnderMP3Cover are the only two MP3 steganographic tools known apart from the new technique proposed in the previous paper.

Step 1 of the detection process uses a Support Vector Machine (SVM) to identify the encoder used to encode the MP3 file in question as outlined in [25]. The tool uses a multi-class SVM for encoder classification. Step 2 employs strategies described in [1] [2] that target the detection of specific MP3 steg tools MP3Stego and UnderMP3Cover. Step 1 of the detection process is used for pre-filtering MP3 files in order to reduce the false positive rate. This step would be beneficial in detecting encoders used to implement new steg techniques in the future.

2.1 Introduction

Steganalysis techniques range from simple methods of detecting presence of hidden data by perceptual analysis of the media, exploiting weak hiding techniques for example a JPEG steganographic technique that hides data after the end of file marker to complex ones that employ machine learning and second order statistics [26]. In techniques that employ machine learning the objective is to analyze the alterations in the media that result from the hiding of data and build attributes that can be used to distinguish stegged from non-stegged files. This paper describes a MP3 steganalysis tool built based on the work in MP3 encoder classification and steganalysis by Bohme and Westfeld in their papers [25] [1] [2] . The layout of the paper is as described. Section 2.2 of the paper gives a high level view of the MPEG I/II layer III audio encoding algorithm. Section 2.3 covers the existing MP3 based steg tools. Section 2.4 develops a basic understanding of classification using support vector machines. Section 2.5 delves into the proposed tool framework with test results. Section 2.6 concludes highlighting future work.

2.2 MPEG Audio Encoders

MP3 Encoders are software that implement the MPEG audio specifications in compressing an audio signal to MP3 format in the process achieving a compression ratio of nearly 1:12. An uncompressed audio file is stored as PCM (Pulse code Modulated) samples. PCM samples are a digital representation of the analog waveform of an audio signal. In general the audio format is a WAV (Waveform audio format) file containing uncompressed PCM samples. MP3 encoders that convert WMA (Windows Media Audio) format to MP3 format also exist.

Though no one holds exclusive rights MP3 technology has most of the algorithms patented by Fraunhofer-Gesellschaft. Despite patent issues quite a few MP3 'like' encoders exist and follow the same basic encoding blocks as shown in

Figure 9. The normative elements in the MPEG standard specify the format of the bit stream (compressed audio) and the structure of the decoder. The encoder implementation is completely left to the implementer. This freedom has given rise to different encoders which produce MP3 bit stream with same format albeit distinct properties. Step 1 of MP3 steganalysis detection uses a SVM with these properties as features to differentiate between the MP3 encoders.

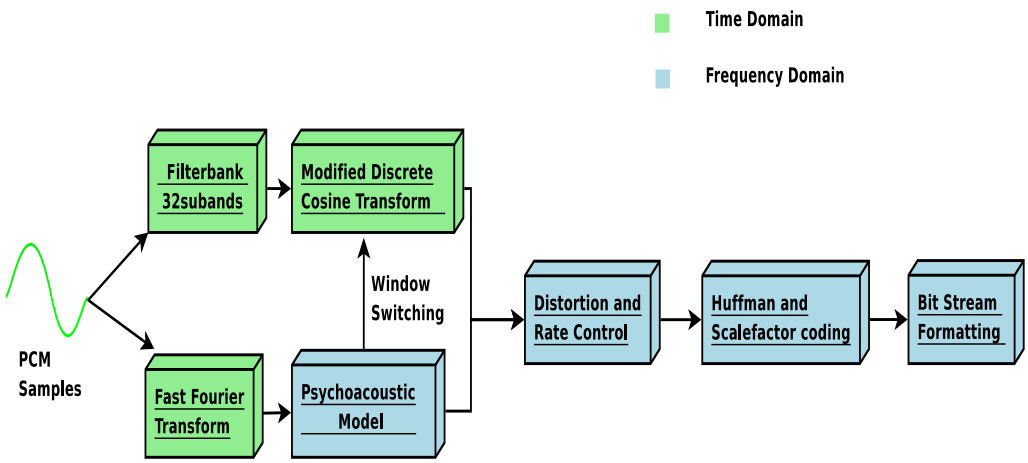


Figure 9: MP3 Encoding process

2.3 Overview of MP3 steg tools

As mentioned earlier MP3 based steganography is still in its early stages. There are 2 known tools for MP3 based steganography MP3Stego and UnderMP3Cover. A description of the hiding procedure employed in the tools is described below.

2.3.1 MP3Stego

MP3Stego [7] is one of the earliest MP3 based steganographic tools. Data hiding in MP3Stego is done as part of the encoding process. MP3Stego uses the power-of-parity [9] principle to embed data in part2_3_length of a granule in a MP3 file. The part2_3_length variable indicates the total number of bits

required to encode the scalefactors and the Huffman coded data. The granules to be modified are randomly chosen (using SHA-1). The value and hence the parity of part2_3_length variable is modified in the inner-loop during quantization. In addition to the original condition that the number of bits used for encoding a block be within a bound, the loop terminates only if the parity of the variable part2_3_length is the same as the bit to be embedded.

2.3.2 UnderMP3Cover

UnderMP3Cover [8] is a MP3 LSB steganographic tool that uses global_gain to hide data. Unlike MP3Stego, UnderMP3Cover hides data in an already encoded MP3 file. The modification is done on the LSB of global_gain variable in selected granules to reflect the embedded bit. The tool uses a spacing parameter to select the granules to embed in. MP3Stego and UnderMP3Cover have comparable data-hiding rates for a given carrier file.

The maximum steg capacity of the carrier when used with MP3Stego and UnderMP3Cover is $4 * number\_of\_frames$ bits. Both the programs can embed a maximum of 4 bits in a frame if the signal is stereo since a stereo signal has 2 channels and each channel has two granules.

2.4 Support Vector Machines and Steganalysis

Support vector machines were invented by Vladimir Vapknik. SVMs are a classification technique which use the concept of maximal margin classifiers as the basis for classification. Maximum margin classifiers allow for better generalization of the classifier by placing the decision surface equidistant from the two classes. In addition the decision surface is placed such that it maximizes the distance between the classes. The data points in either class that constrain the decision surface from moving any further in either direction are called support vectors. A soft margin

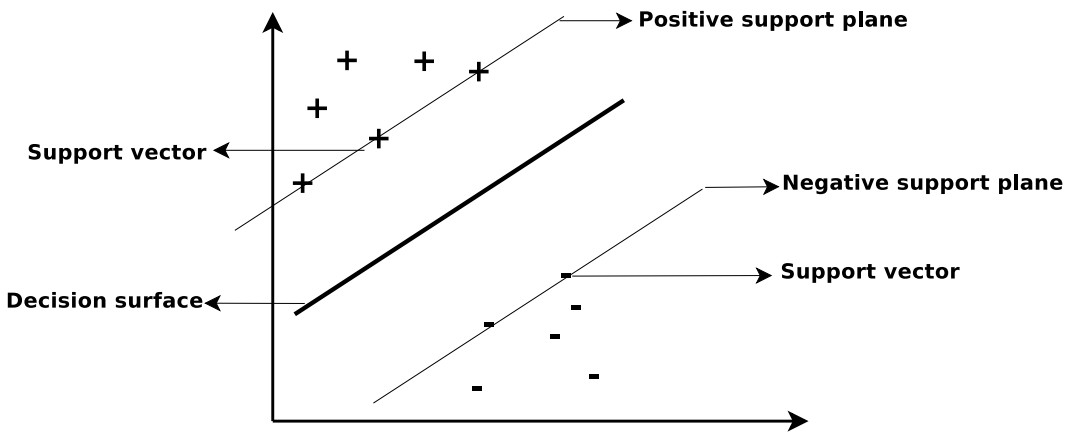


Figure 10: Support Vector Machine Model with a linear decision surface

SVM is one wherein the decision function is allowed to make mistakes. The concept of soft margin classifiers introduces a cost parameter in the formulation of the decision surface. The cost parameter is introduced to penalize the decision surface on erroneous classification. Noisy data points are the reason for the use of soft margin classifiers. The formulation of the decision surface in such a case often is dictated by the noisy points which do not represent the true separation between the classes. In order to have better generalization a slack variable is introduced in the optimization problem for the maximum margin classifier and the classifier is allowed to make mistakes on these noisy points so that the margin can be made wider. With the slack variable the optimization has a trade-off between margin width and error. This trade-off makes a case for having high costs which results in smaller margins due to large penalties resulting from possible misclassification. Small margin also means that the model may not generalize well. On the other hand smaller costs implies wider margins which implies that the model has the ability to learn more while allowing for it to make mistakes thus making the model more in tune with the real world noisy data for classification.

The power of SVMs is highlighted by the fact that the decision surface can be

based on non-linear kernel functions that can be used to separate data that is not linearly separable in higher dimensions. The kernel trick as it is called employs transformation function to data points in the input space where the data is not linearly separable to a higher dimensional space called the feature space thereby making the data linearly separable. The kernel functions are peculiar in the sense that their properties of positive definiteness among others enable the feature space computations to be performed in input space which is quite remarkable.

Of the algorithms that implement the SVM methodology the SMO (Sequential Minimal Optimization) is the popular implementation used in machine learning. The concept of VC-dimension in defining model complexity plays an important role is selecting a model that is less complex as they are the ones that are likely to generalize better. In most cases the underlying data might not be a true representation of the data universe in terms of completeness. Without delving into the details the VC-dimension of a model class [1] defined for a dataset set D (n data points) is the largest subset of D (size m) shattered by the model class. In the above definition m is the VC-dimension of the model class. A model class is said to have shattered a data set D if for all the possible label configurations in the data set the models in the model class can separate the data points perfectly. Since the data set used for training a model and hence a classifier is a representation of the data universe we cannot expect to have the knowledge in order to reduce the expected risk. However we could learn from the observed data in the data set and reduce the empirical risk. This is called empirical risk minimization. Overly optimistic empirical risk minimization and reducing the training error to achieve high accuracy can lead to poor generalizable models.

Multi-class SVMs with soft-margins use pairwise classification to build classi-

[1] A model class represents all the possible configurations (rotation and translation) of a decision surface with a given width.

fication models. In general, models with low cost (extremely soft margin) tend to lean towards a heavily weighted (in terms of instances) class. Since in multi-class problems the classes might not be evenly represented in the training data this could cause a problem. It is resolved by pairwise classification between each of the classes. In order to classify an unknown observation a voting scheme is adopted whereby the class that gets the largest number of votes with the pairwise classification is assigned to the unknown observation. For an in-depth understanding of SVMs refer [27] and [28].

The MP3 encoder classification problem uses a soft margin multi-class SVM. Various kernels linear, radial and polynomial are tested to find the best model in terms of accuracy and generalization.

2.5 MP3 Steganalysis Tool Architecture

The ensuing sections elaborate on the architecture of the tool, concepts of machine learning adopted along with the validity and significance of the classification model built.

2.5.1 The tool architecture

Figure 11 describes the layout of the MP3 steganalysis tool in detail. The tool consists of two parts. The first part describes an encoder classification scheme built using a support vector machine (SVM) for classification of MP3 files based on the encoder used to create them. The feature extraction procedure was built using the mpglib MP3 decoding library. All the features are generated as part of the MP3 decoding process and are written onto a file. The feature file is then loaded in the R programming environment for training/testing using SVM. The result of the multi class SVM is an output depicting the class (encoder) which the MP3 file belongs to. The files encoded using 8Hz and SoloH encoders are the only

ones that make it to the second stage of steganalysis.

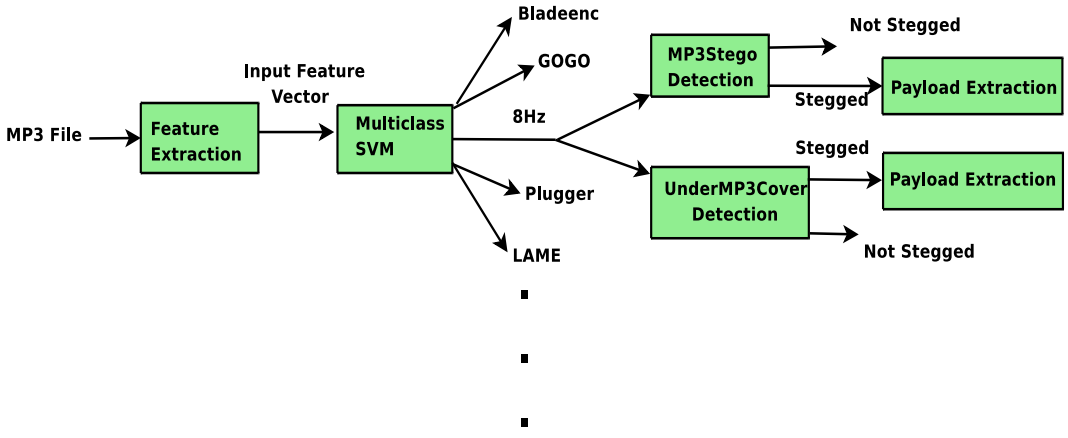


Figure 11: MP3 Steganalysis Tool Framework

The features used for encoder classification are specified in table 1. Details on the significance of using these features can be found in [25].

Table 1: Features used for classification

1	Effective bit rate ratio
2	Granule size balance
3	Reservoir usage ramp
4	preflag ratio
5	Block type transitions
6	SCFSI usage
7	Frame length alignment
8	Huffman table selection
9	Stuffing byte value

A dataset of 2000 MP3 files was generated by encoding 200 wav files using 10 different encoders. The encoders that are detected using the SVM classification are listed in Table 2. In order to generate the dataset a set of 200 wav files was used. Each of the wav file was encoded into the MP3 format by encoding them with the encoders in Table 2. MP3 files encoded with 8Hz and SoloH are carried

Table 2: The encoder list

1	8Hz
2	plugger
3	mp3sEnc
4	fastenc
5	shine
6	gogo
7	SoloH
8	m3e
9	lame
10	bladeenc

over to the second stage. The encoding speed observed is varied with Gogo being the fastest encoder and SoloH being the slowest. The reasons for restricting the encoders to the ones mentioned in the list are

1. Certain Fraunhofer and Xing MP3 encoders are not freely available due to licensing issues.

2. For the purpose of steganalysis we are only interested in pre classifying encoders that have modifiable source code. Steganographic tool development would only happen on open source encoders.

2.5.2 Training and Testing Classifier Models

In order to build a suitable model for the classification of 10 encoders in Table 2 we built a feature extraction program using the mpglib library and extracted features from over 2000 MP3 files. These MP3 files have a mix of different genre and duration. The tracks include live performance from The Beatles, Classical guitar, Van Morrison, Pearl Jam and Sting. Additionally, 1000 MP3 files were assembled for testing purposes. Step 1 of the tool extracts the feature vector from an MP3 file and builds a multi-class SVM for encoder classification. We use the tune function along with the e1071 package (libsvm library) in R for training the

models. The models are built with 10-fold cross-validation to reduce the bias. A soft-margin classifier is used to train a SVM model on these feature vectors whereby we are able to account for outliers (noise) that would mimic the real world scenario. Table 3 shows the range for the free parameters that were used to train models using SVM. Linear, Radial and Polynomial kernels are used. Not all kernels have all the free parameters, a '-' in the table represents the absence of the parameter for the kernel.

Table 3: SVM Training

Kernel	Cost Range	Gamma Range	Coef0 Range	Degree	TrainAcc %
Linear	0.01-1000	-	-	-	89.12
Radial	0.01-1000	0.0625-256	-	-	90.54
Polynomial	0.01-1000	0.0625-256	-100-1000	2,3,5,7,8	91.74

Table 4 shows the best parameters that are selected from the models built. The optimal values for the parameters along with the test accuracy on 1000 pristine MP3 files with a mix of all encoders is shown in Table 4. The polynomial kernel is chosen based on the accuracy results on the test set of 1000 MP3 files. The bootstrap confidence interval range for each of the kernels is also shown in Table 4.

Table 4: SVM Test Results

Kernel	Cost	Gamma	Coef0	Degree	TestAcc %	Bootstrap Interval
Linear	1	-	-	1	85.88	85.71 - 90.50
Radial	10	0.0625	-	-	86.82	86.76 - 90.61
Polynomial	0.01	0.5	10	2	90.47	85.05 - 92.29

2.5.3 Validity and Statistical Significance

The confidence interval represents the impact of the uncertainty of the real world data on the classifiers ability to predict. We ran a bootstrap algorithm with

200 bootstrap samples each having 3000 data points using the optimal parameters obtained for each of the kernels. Each of the 200 samples was sampled from the original dataset with replacement. The replacement in sampling represents the bias in the real world data. A 10 fold cross validated error is computed for each sample. Each fold has a split of 90/10 (hold out method). The cross validated error results are then sorted in the ascending order. In order to derive a 95% confidence error interval we extract the 2.5^{th}% percentile which forms the lower bound and 97.5^{th}% percentile bound which forms the upper bound. Thus the accuracy ranges in the confidence interval column of Table 4 imply that we are 95% sure that with the bias in the real world scenario the models have an accuracy that fall in the given range.

Based on the test results and the bootstrap confidence interval values the polynomial kernel is chosen as it is statistically significant and has a better test accuracy.

The confusion matrix of the polynomial kernel model on the 1000 test samples is given in Table 5. The values represent classification accuracy in terms of a percentage for each of the encoder class. As can be observed the model does not alleviate the problem faced by authors in [25] which is the false classification of 8Hz as SoloH and vive-versa. As mentioned in [25] this is attributed to the similarity in the origin of these encoders. The error rate does not however cause problems in the step 2 as we run the steganalysis detection on the files that have been classified as either 8Hz or SoloH.

Step 2 of the tool detects specific MP3 steg techniques which include MP3Stego and UnderMP3Cover. As mentioned earlier the nascent nature of MP3 steg techniques is the reason for the small number of MP3 steg tools. Both the steg tools have been successfully detected in Westfeld's papers [1] [2].

Table 5: Confusion Matrix for the polynomial kernel

	8Hz	plugger	fastenc	shine	gogo	m3e	lame	SoloH	bladeenc	mp3s
8Hz	86	0	0	0	0	0	0	28	0	0
plugger	0	100	0	0	0	0	0	0	0	0
fastenc	0	0	100	0	0	0	0	0	0	0
shine	0	0	0	100	0	0	0	0	0	0
gogo	0	0	0	0	100	0	0	0	0	0
m3e	0	0	0	0	0	100	0	0	0	0
lame	0	0	0	0	0	0	100	0	0	0
SoloH	24	0	0	0	0	0	0	82	0	0
bladeenc	0	0	0	0	0	0	0	0	100	0
mp3sEncoder	0	0	0	0	0	0	0	0	0	100

2.5.4 MP3Stego Detection

MP3Stego detection employs block length analysis to distinguish MP3 files that are stegged using MP3Stego software from the ones that are encoded using any other MP3 encoder. As explained in [1], a MP3Stego modified MP3 file has the same size as the original file, despite the block sizes being different. This is due to the MP3 rate control process with CBR (Constant Bit Rate) audio which results in encoder compensating for the extra bits in one frame by reducing the bits allocated to a subsequent one. Despite the mean of the block lengths being the same their variance in a steganographically modified file is different from that of a non-stegged MP3 file. Figure 12 shows this difference in variance using a histogram on block lengths on 2 MP3 files, stegged and non-stegged. A non-stegged MP3 files has unimodal distribution of block length which peaks near the average frame length.

MP3Stego detection involves 2 stages. The first stage in building the MP3Stego detection engine involves determining the autoregressive coefficients β_0, β_1 and β_2 as per the block length relationship $block_i = \beta_0 + \beta_1.block_{i-1} + \beta_2.block_{i-2}$ mentioned in [1] . A model using quadratic discriminant analysis (QDA) with β_0, β_1 and β_2 as feature vectors was built to distinguish files encoded using MP3Stego and 8Hz encoder. The model was able to achieve 100% distinction between 8Hz and MP3Stego which is supported by the confusion matrix in Table

2.5.4. The model was built using 1500 MP3 files with equal number of stegged and non-stegged files. The stegged files had data at 50% embedding capacity. The model was tested on 1000 pristine MP3 files with equal number of files from the steg and the non-steg category.

Table 6: Confusion Matrix for QDA model for MP3Stego detection

	MP3Stego	8Hz
MP3Stego	500	0
8Hz	0	500

2.5.5 UnderMP3Cover Detection

The detection of UnderMP3Cover in the tool is mere integration of code cited in the work [2]. The program updet exploits the feature of the steg tool whereby the size information of the file that is hidden is stored in the first 6 bits of the carrier MP3 file. By extracting this data the program **updet** checks if this value is larger than the theoretical maximum which is $4 * total\_number\_of\_frames$. The program has a limitation that it assumes a default spacing of 2 (though adding detection for other values of spacing is trivial).

As part of the UnderMP3Cover detection the tool filters out any MP3 files that have not been encoded using either 8Hz or SoloH. In stage 2 the detector for UnderMP3Cover is invoked with the MP3 files that get filtered from stage 1.

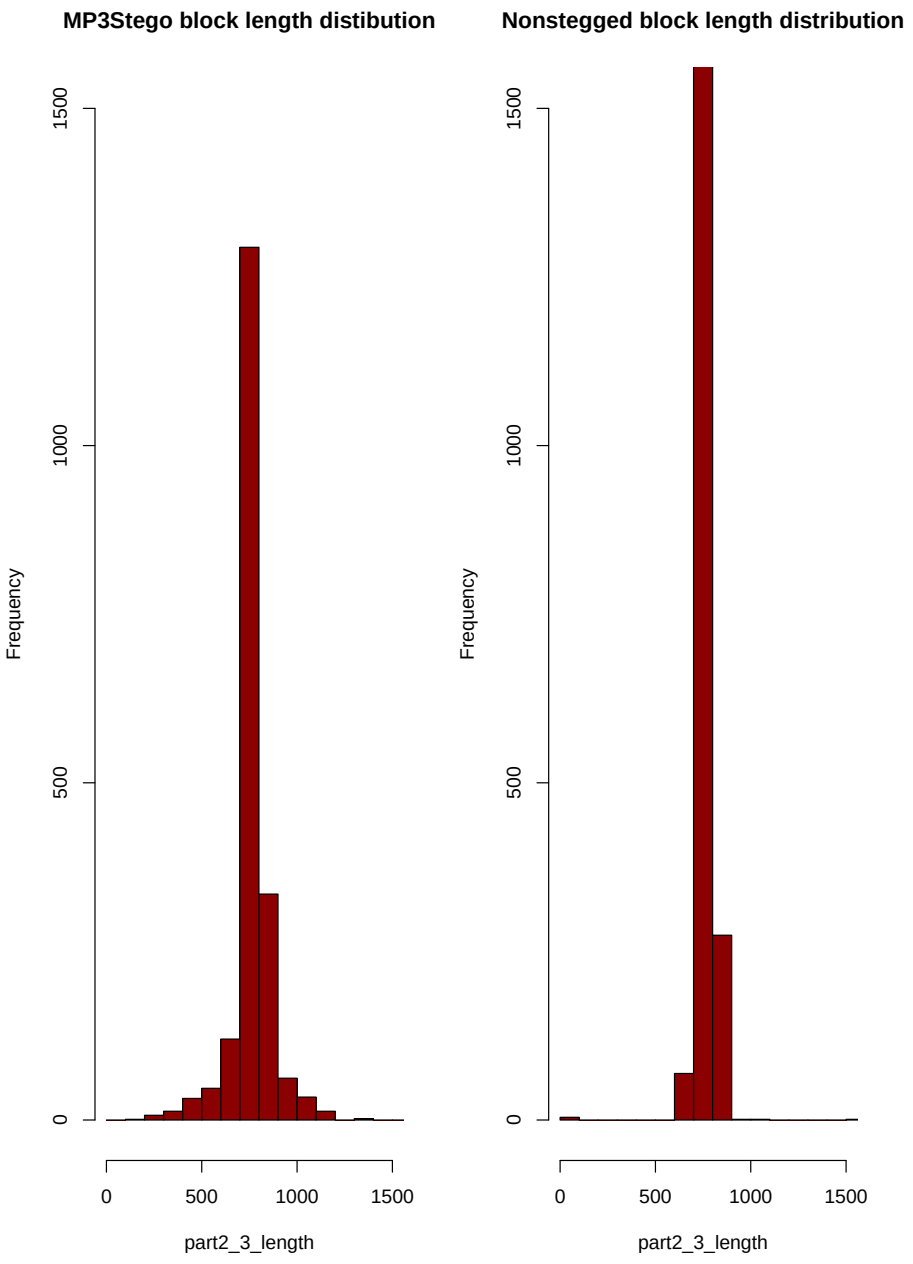


Figure 12: Block length distribution

Conclusion and Future Work

In this paper we have outlined a framework for MP3 encoder classification and steganalysis. We adopted the work in this field by Boheme and Westfeld and extended it to our steganalysis framework which also includes JPEG steganalysis. The implementation uses SVMs for encoder classification with the polynomial kernel chosen as a result of better generalization. In order to analyze the model effectiveness in the real world we performed the statistical significance test via bootstrapping. We were able to achieve comparable results with that in the original work using SVM, in addition by performing bootstrapping we have also demonstrated the viability of the use of the SVM classification methodology in the real world. In terms of steganalysis we built into the tool procedures to detect MP3Stego and UnderMP3Cover as described in Westfeld's work [1] [2].

In future we intend to investigate possible features for accurate classification of 8Hz and SoloH. We propose the use of SVMs for this purpose in order to be able to tune the free parameters in various kernels expecting to reveal decision surfaces that could exploit hidden patterns to differentiate these encoders. The encoders themselves need to be studied to develop new features.

LIST OF REFERENCES

[1] A. Westfeld, "Detecting Low Embedding Rates," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 324–339, 2003.

[2] A. Westfeld, "Steganalysis in the Presence of Weak Cryptography and Encoding," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4283, p. 19, 2006.

[3] G. Simmons, "The prisoners problem and the subliminal channel," in *Proceedings of CRYPTO*, vol. 83, 1984, pp. 51–67.

[4] N. Provos. "Outguess Steganography tool." [Online]. Available: http://www.outguess.org

[5] A. Westfeld, "F5-A Steganographic Algorithm: High Capacity Despite Better Steganalysis," in *Information Hiding: 4th International Workshop, IH 2001, Pittsburgh, PA, USA, April 25-27, 2001: Proceedings.* Springer, 2001.

[6] "Steghide Steganography tool." [Online]. Available: http://steghide.sourceforge.net/

[7] University of Cambridge. "MP3Stego." [Online]. Available: http://www.petitcolas.net/fabien/steganography/mp3stego/

[8] Sourceforge. "UnderMP3Cover." [Online]. Available: http://www.files-library.com/files/UnderMP3Cover.html

[9] F. A. P. Ross J. Anderson, "On The Limits Of Steganography," *IEEE Journal of Selected Areas in Communications*, vol. 16, no. 4, pp. 474–481, May 1998.

[10] K. Brandenburg and H. Popp, "An Introduction to MPEG Layer," 2003.

[11] S. Khalid, *Introduction to Data Compression.* Morgan Kaufmann, 2000.

[12] D. Pan, M. Inc, and I. Schaumburg, "A tutorial on MPEG/audio compression," *Multimedia, IEEE*, vol. 2, no. 2, pp. 60–74, 1995.

[13] A. Servetti, C. Testa, J. De Martin, and D. e Informatica, "Frequency-selective partial encryption of compressed audio," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, vol. 5, 2003.

[14] MPEG, "Coding of Moving Pictures and Associated Audio for Digital Storage Media at Upto 1.5 MBIT/s," 1991.

[15] R. Raissi, "The Theory Behind Mp3," 2002.

[16] A. B. B. John P. Pricen, "Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation," *IEEE transactions on Acoustic,Speech and Signal Processing*, vol. ASSP-34, no. 5, October 1986.

[17] S. Hacker and S. Hayes, *MP3: The Definitive Guide*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2000.

[18] K. Do-Hyoung, Y. Seung-Jin, and C. Jae-Ho, "Additive Data Insertion Into MP3 Bitstream Using linbits Characteristics,"," *Proc. on ICASSP04, IV-181-184*, 2004.

[19] L. Gang, A. Akansu, and M. Ramkumar, "MP3 resistant oblivious steganography," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 3, 2001.

[20] N. Moghadam and H. Sadeghi, "Genetic Content-Based MP3 Audio Watermarking in MDCT Domain," *watermark*, vol. 1, no. 2, p. 3.

[21] Y. Wang, L. Yaroslavsky, M. Vilermo, and M. Vaananen, "Some peculiar properties of the MDCT," in *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on*, vol. 1, 2000.

[22] "8Hz MP3 Encoder." [Online]. Available: http://www.8hz.com/mp3/

[23] "LAME MP3 Encoder." [Online]. Available: http://lame.sourceforge.net/

[24] "Audacity." [Online]. Available: http://www.audacity.sourceforge.net/

[25] R. Böhme and A. Westfeld, "Statistical characterisation of MP3 encoders for steganalysis," in *Proceedings of the 2004 workshop on Multimedia and security*. ACM New York, NY, USA, 2004, pp. 25–34.

[26] T. Pevny and J. Fridrich, "Merging markov and dct features for multi-class jpeg steganalysis," *IS&T/SPIE EI*, vol. 6505, 2007.

[27] L. Hamel, "Knowledge Discovery With Support Vector Machines," unpublished.

[28] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

BIBLIOGRAPHY

"Audacity." [Online]. Available: http://www.audacity.sourceforge.net/

Böhme, R. and Westfeld, A., "Statistical characterisation of MP3 encoders for steganalysis," in *Proceedings of the 2004 workshop on Multimedia and security*. ACM New York, NY, USA, 2004, pp. 25–34.

Brandenburg, K. and Popp, H., "An Introduction to MPEG Layer," 2003.

Burges, C., "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

Do-Hyoung, K., Seung-Jin, Y., and Jae-Ho, C., "Additive Data Insertion Into MP3 Bitstream Using linbits Characteristics,"," *Proc. on ICASSP04, IV-181-184*, 2004.

"8Hz MP3 Encoder." [Online]. Available: http://www.8hz.com/mp3/

"LAME MP3 Encoder." [Online]. Available: http://lame.sourceforge.net/

Gang, L., Akansu, A., and Ramkumar, M., "MP3 resistant oblivious steganography," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, vol. 3, 2001.

Hacker, S. and Hayes, S., *MP3: The Definitive Guide*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2000.

Hamel, L., "Knowledge Discovery With Support Vector Machines," unpublished.

John P. Pricen, A. B. B., "Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation," *IEEE transactions on Acoustic,Speech and Signal Processing*, vol. ASSP-34, no. 5, October 1986.

"Steghide Steganography tool." [Online]. Available: http://steghide.sourceforge.net/

Khalid, S., *Introduction to Data Compression*. Morgan Kaufmann, 2000.

Moghadam, N. and Sadeghi, H., "Genetic Content-Based MP3 Audio Watermarking in MDCT Domain," *watermark*, vol. 1, no. 2, p. 3.

MPEG, "Coding of Moving Pictures and Associated Audio for Digital Storage Media at Upto 1.5 MBIT/s," 1991.

Pan, D., Inc, M., and Schaumburg, I., "A tutorial on MPEG/audio compression," *Multimedia, IEEE*, vol. 2, no. 2, pp. 60–74, 1995.

Pevny, T. and Fridrich, J., "Merging markov and dct features for multi-class jpeg steganalysis," *IS&T/SPIE EI*, vol. 6505, 2007.

Provos, N. "Outguess Steganography tool." [Online]. Available: http://www.outguess.org

Raissi, R., "The Theory Behind Mp3," 2002.

Ross J. Anderson, F. A. P., "On The Limits Of Steganography," *IEEE Journal of Selected Areas in Communications*, vol. 16, no. 4, pp. 474–481, May 1998.

Servetti, A., Testa, C., De Martin, J., and e Informatica, D., "Frequency-selective partial encryption of compressed audio," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, vol. 5, 2003.

Simmons, G., "The prisoners problem and the subliminal channel," in *Proceedings of CRYPTO*, vol. 83, 1984, pp. 51–67.

Sourceforge. "UnderMP3Cover." [Online]. Available: http://www.files-library.com/files/UnderMP3Cover.html

University of Cambridge. "MP3Stego." [Online]. Available: http://www.petitcolas.net/fabien/steganography/mp3stego/

Wang, Y., Yaroslavsky, L., Vilermo, M., and Vaananen, M., "Some peculiar properties of the MDCT," in *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on*, vol. 1, 2000.

Westfeld, A., "F5-A Steganographic Algorithm: High Capacity Despite Better Steganalysis," in *Information Hiding: 4th International Workshop, IH 2001, Pittsburgh, PA, USA, April 25-27, 2001: Proceedings*. Springer, 2001.

Westfeld, A., "Detecting Low Embedding Rates," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 324–339, 2003.

Westfeld, A., "Steganalysis in the Presence of Weak Cryptography and Encoding," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 4283, p. 19, 2006.