MP3 Steganography Techniques

Yalinne Castelan
Illinois Institute of Technology
School of Applied Technology
201 East Loop Road
Wheaton, IL 60189
+1 (708) 244-8819
ycastela@hawk.iit.edu

Ben Khodja
Illinois Institute of Technology
School of Applied Technology
201 East Loop Road
Wheaton, IL 60189
+1 (630) 815-9149
bkhodja@hawk.iit.edu

ABSTRACT
The MP3 audio encoding format is currently one of the most popular encoding formats in use. On average, it takes significantly less space to store than, for example, a PCM-encoded WAV file, even though it retains most of the original, uncompressed audio information. Most users of MP3 files are unaware that they can be used as carriers of steganographically-hidden information. Although it has not yet been greatly explored, there are various types of steganography techniques that exist which allow for the hiding of information within MP3 files of any type. Two of the techniques the authors studied include those implemented by MP3Stego developed by Fabien Petitcolas and MP3Stegazaurus by former Illinois Institute of Technology student Mikhail Zaturenskiy. The technique implemented by MP3Stego performs the injection of covert information during the encoding process, while the technique implemented by MP3Stegazaurus uses areas of an already-existing MP3 file known to be skipped or ignored by MP3 playing and decoding applications to store covert information.

The authors have developed a tool which implements a technique allowing for the hiding of covert information within the audio information-containing portions of an MP3 file in a way that does not yet appear to have been explored. Unlike the technique implemented by MP3Stego, which involves hiding information within the audio information-containing portions of an MP3 file during the encoding process, this new technique involves hiding it after the encoding process has already taken place. This paper details the research carried out by the authors as well as how the tool that makes use of this technique works.

Categories and Subject Descriptors
H.5.5 [Sound and Music Computing], E.3 [DATA ENCRYPTION]

General Terms
Algorithms, Measurement, Documentation, Experimentation, Security, Theory, Verification

Keywords
Steganography; MP3; Secret Writing; Hidden Information; Hide Information; Covert Data

RIIT'15, September 30–October 3, 2015, Chicago, IL, USA
 ISBN 978-1-4503-3836-3/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2808062.2808074

1. INTRODUCTION
The MP3 (MPEG-1, Layer 3) audio encoding format was introduced in 1993 and released in 1995 as International Standard ISO/IEC 11172-3. MPEG stands for Motion Pictures Experts Group which is a group of authorities formed by ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) to set standards for audio and video compression and transmission. MPEG-1 is a standard for the lossy compression of digital audio and video information and Layers 1, 2, and 3 are audio encoding formats for digital audio information.

When looking at the steganography tools available for MP3, the authors realized they could possibly develop a new technique and tool with the ability to hide larger amounts of covert information.

2. MP3 FILE FORMAT
To create an MP3 file, the original, uncompressed (PCM or WAV) audio is broken down into frames containing 26 milliseconds worth of audio information, and the number of frames contained within an MP3 file is directly proportional to the duration of the audio. The sample frequency and bitrate selected during the encoding process determine the size in bytes of each frame. These frames are then processed to determine which information will be kept and which information will be removed during the encoding process, and different masks will be applied to determine which information cannot be perceived by humans.

As an example, an MP3 file encoded using a specified constant bitrate of 128kbps and sampling frequency of 44,100 Hz will have a frame size of 417 bytes and sometimes 418 bytes when padding must be applied to maintain the exact average bitrate throughout the entire MP3 file. An MP3 file encoded using a specified constant bitrate of 192kbps and sampling frequency of 44,100 Hz will have a frame size of 626 bytes. Optionally, each frame may contain a 2-byte CRC value and the entire MP3 file may begin with an ID3v2 tag or end with an ID3v1 tag. The following figure illustrates the components that make up a complete MP3 file as well as how they are organized.

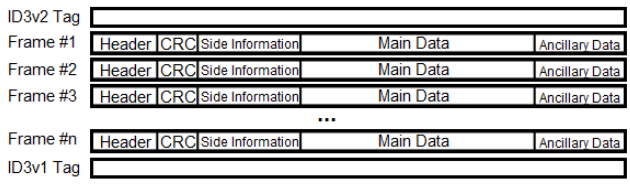

Figure 1. Structure of a complete MP3 file
Note: ID3 Tag components and CRC portions are optional.

2.1 Frame Header

The header portion of a frame contains information related to that particular frame. The bitrate and sample frequency values, for example, are stored here. The optional CRC component, if one exists, is also specified in this portion and the Protection field of every frame's header portion will contain a zero value. The following 2 bytes after every frame header's portion form the CRC value. The algorithm used to calculate the CRC value is CRC16. The last 16 bits (two bytes) of the header portion along with the entire side information portion are used as inputs to the CRC16 algorithm. [10] Frames that contain Protection field values that specify the use of CRC values and have missing or incorrect CRC values will be skipped by MP3 playing and decoding applications. [8] The header portion is composed of the first 32 bits (four bytes) of every frame.

2.2 Side Information

The side information portion contains information related to the main data portion of each frame. Information about Huffman tables and other information needed to decode the main data portion is stored here. The size of the side information portion changes, and depends on the channel mode specified during the MP3 encoding process. For MP3 files encoded using the single channel mode, the 17 bytes following a frame's header portion (or CRC value, if one exists) will compose the frame's side information portion. For MP3 files encoded using the dual channel, stereo, or joint stereo mode, the 32 bytes following a frame's header portion (or CRC value, if one exists) will compose the frame's side information portion. [7] In this portion there is also information about the location of certain bits that will be necessary to decode the main data portion. For example, the "region0_count" and "region1_count" fields hold the lengths in bits of "region0" and "region1." These can be used to determine the beginning and ending locations of a channel's particular regions.

Some other relevant bits for this project are the "part2_3_length" field that holds the total length in bits for a particular channel (or granule in the case of a single-channel MP3 file). It can be used to determine the beginning and ending locations of a particular granule or channel. The "table_select" and "count1table_select" fields are used to specify which Huffman Coding tables are to be used in order to decompress the Huffman-encoded data stored within the three regions of a particular channel (or granule in the case of a single-channel MP3 file). These Huffman Coding tables are pre-defined and are part of the MP3 standard. They appear in the "Table B.7 – Huffman codes for Layer III" section of the standard [4].

2.3 Main Data

The main data portion of a frame is where the actual audio information is stored. It is divided into two granule portions (Granule 0 and Granule 1) which then are broken down into two channel portions (Left channel portion and Right channel portion) in the case of dual channel, or is not further divided in the case of single channel. In the case of single channel, the entire granule portion will be also considered the channel portion. The audio information present in the main data portion is encoded by Modified Discrete Cosine Transform (MDCT) and then compressed using Huffman Coding which is a lossless form of compression. The size of a frame's main data portion is dependent upon the values of the bitrate and sampling frequency fields present in the frame's header portion. For performance reasons, the audio information contained within each channel (scalefactor values and MDCT-encoded frequency subband values) is split into 3 regions named "region0," "region1," and "region2," and each of these regions is compressed individually using different Huffman Coding tables (if necessary). As mentioned previousy, these tables are pre-defined and are a part of the MP3 standard. The standard defines 15 large tables for these regions, where each table outputs two frequency samples for a given code word. The tables are designed to compress the "typical" content of the frequency regions as much as possible. [2]

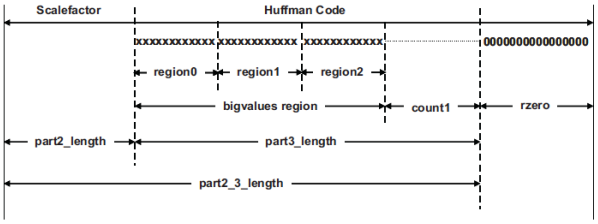

Figure 2. Structure of main data portion [11]

It is important to note the audio information referenced by a particular frame's header and side information portions may actually exist within the main data portion of one or more preceding frames. It is possible for audio information to not exist at all in the main data portion of the frame in which it is referenced; this is due to a space-saving feature of the MP3 encoding format known as the bit reservoir. The bit reservoir feature allows frames to share unused main data portion space with any number of frames that follow. As an example, a frame may be allocated 500 bytes of main data portion space based on its bitrate and sampling frequency field values. If only 300 of those bytes are used to store audio information referenced by that frame's header and side information portions, the frame that follows will store the audio information referenced by its header and side information portions starting with the first unused byte of the preceding frame's main data portion (the 301st byte in this case). This ensures that no unused space exists within the main data portion of a frame and may result in a decrease in overall MP3 file size. [8]

2.4 Ancillary Data

The ancillary data portion of a frame is located after the main data portion and its size varies because its main function is to round the frame's main data portion up to an integer number of bytes. The number of bits within the ancillary data portion usually ranges from one to seven (if it exists) depending on the number of bits left over in the last incomplete byte present in a frame's main data portion. In some cases, the authors found the number of bits in a frame's ancillary data portion to exceed seven bits. This was especially true for frames containing a partially empty main data portion, usually found toward the end of an MP3 file.

3. EXISTING MP3 STEGANOGRAPHY TECHNIQUES AND TOOLS

3.1 MP3Stego

MP3Stego is a tool developed by Fabien Petitcolas. It hides information within the main data portions of an MP3 file, during the encoding process.

MP3Stego accepts a WAV file and a file to be hidden as inputs. It will compress and encrypt the file to be hidden first. Then, it will inject this file into the MP3 audio bit stream. The hiding is performed during the "inner_loop" stage of the encoding process which is where quantization of the input audio takes place.

3.2 MP3Stegazaurus

MP3Stegazaurus is a tool recently developed by Mikhail Zaturenskiy. It injects covert information into MP3 files by overwriting the fields within the non-audio-information-containing portions of an already-existing MP3 file's frames that MP3 playing and decoding applications are known to skip or ignore. It can also retrieve previously-injected covert information as long as the user is aware of, and is able to, specify exactly which method was used to inject it and which file type extension is to be appended to it. Finally, it cleans potential carrier MP3 files of all previously-injected covert information by using a version of the injection method that overwrites the fields within the specified portions of interest with a repeating zero value.

4. NEW MP3 STEGANOGRAPHY TECHNIQUE AND TOOL: MP3STEG

MP3Steg is a tool that allows the user to inject and retrieve covert information into/from the main data portions of an existing MP3 file's frames.

MP3Steg allows the user to select a single file (of any type) containing the covert information to be hidden, and inject it within the main data portions of an MP3 file's frames. This tool is capable of distributing the information within several frames of an MP3 file. Using the audio-information-containing (main data) portions of an MP3 file, this tool takes advantage of the large amount of space that exists there.

MP3Steg works by allowing the user to overwrite the information present within the main data portion of an MP3 file's frames without first decoding, uncompressing, or processing it in any way. The modified frames will then be handled in different ways by MP3 decoding and playing applications.

MP3Steg is a very flexible tool and allows the user to make some important decisions such as specifying the number of consecutive non-carrier frames that are to exist between any two sets of consecutive carrier frames. The user also needs to specify the range of frames to be considered when determining where covert information is to be stored, as well as specifying the maximum number of frames to be used consecutively in order to store the covert information. The covert information will be hidden in the frames that meet the specifications provided.

MP3Steg is a Java application that accepts single and dual channel MP3 files that use constant or variable bitrates. The bitrate and sample frequency combination is not relevant to the tool. For testing purposes, the authors collected a sample of 200 clean MP3 files, retrieved from various sources including Amazon, iTunes, Audacity, GoldWave, and FreeCorder.

4.1 Injection

To perform the injection of covert data into an MP3 file, it first must be determined the space that is available for this purpose. MP3Steg will calculate the average of the main data portion space present in the selected MP3 file's frames then will add all the averaged main data portions of the file's frames; this will be the total space available to inject covert information. With this information, the user can do a thoughtful selection of the covert information to use. The user will need to specify the first and last frame to be considered by MP3Steg, the maximum number of frames to be used consecutively to carry covert information, and the number of consecutive non-carrier frames to exit between any two sets of carrier frames. The user has the option to include in the carrier file, the name and extension of the file selected for injection. If the information to be hidden takes less space than the specifications selected by user, MP3Steg will simply use the specifications until all the covert information is injected. MP3Steg will overwrite the entire main data portion of the frame that meets the specifications selected, and will proceed to the next frame that meets the specifications. After this process is complete, a new file will be created in the same folder that contains the MP3Steg.jar executable and it will be named the same as the original carrier MP3 file plus a timestamp value, to ensure uniqueness. Present in the newly-created file will be slight audio distortions that could be more or less noticeable depending on various factors, such as the loudness or quietness of the MP3 audio file.

4.2 Retrieval

To perform the retrieval process, MP3Steg will look for the specifications contained in the carrier file, and based on that will recompose the information or file that was previously injected. The recomposed information is an exact bit-by-bit copy of the information that was selected to be hidden. If the name and extension of the covert file were selected to be hidden, the user will have the retrieved file in the same folder containing the MP3Steg.jar executable.

4.3 Testing

The testing plan first consisted of gathering (downloading as well as generating) a total of 200 MP3 files determined to be free of steganographically-hidden information from various sources and software tools. The clean MP3 files were downloaded from Amazon, iTunes, and Microsoft, while the software tools used to generate clean MP3 files include Audacity, Goldwave, Freecorder, and BladeEnc. Next, a varying number of each MP3 file's frames were overwritten with information (the data contained within the selected covert file) using varying values for each of MP3Steg's four user-configurable injection specifications. These injection specifications include the following:

1. The first frame within the selected carrier MP3 file that is to be considered when determining which frames are to be used as carriers of the selected covert file's data.

2. The last frame within the selected carrier MP3 file that is to be considered when determining which frames are to be used as carriers of the selected covert file's data.

3. The maximum number of frames (within the specified range of frames) that are to be used consecutively in order to carry the selected covert file's data.

4. The number of consecutive non-carrier frames that are to exist between any two sets of carrier frames.

Covert files containing varying types of information as well as varying amounts of information were also used.

Next, each MP3 file (now carrying covert data within the main data portion of several of its frames) was run through various MP3 playing and decoding applications in order for the authors to subjectively determine (through hearing) whether noticeable audio artifacts had been introduced. These MP3 playing and decoding applications include Windows Media Player, Goldwave, WinAmp, QuickTime, Audacity, and VLC.

5. CONCLUSIONS

It was found that none of the MP3 playing and decoding applications used to determine whether noticeable audio artifacts had been introduced (Windows Media Player, Goldwave, WinAmp, QuickTime, Audacity, and VLC) refused to play MP3 files that had covert data injected into the main data portions of several of their frames by MP3Steg. Instead, these MP3 files played completely with varying degrees of noticeable audio artifacts. The following is a summary of the authors' findings:

Any kinds of modifications made by MP3Steg are fairly unnoticeable (though still definitely noticeable when listening very carefully) when performed within quiet or completely silent portions of an MP3 file (portions where there is very little or no sound) and fairly noticeable (definitely noticeable when listening carefully) when performed within portions of an MP3 file that are not quiet or completely silent (portions where there is medium to very loud sound).

Based on these findings, the authors have determined that, overall, it is best to inject covert data into the quiet portions of an MP3 file by configuring MP3Steg to use very high consecutive carrier frame values and very low consecutive non-carrier frame values (or a consecutive non-carrier frame value of 0). This will allow the user to take advantage of as much quiet MP3 file audio space as possible. When injecting into portions of an MP3 file that are not quiet or completely silent, it is best to configure MP3Steg to use low consecutive carrier frame values and high consecutive non-carrier frame values. This will make it so that any noticeable audio artifacts that are introduced are spread across the MP3 file's loud audio space as much as possible.

6. ACKNOWLEDGEMENTS

The authors would like to thank Professor Bill Lidinsky for providing them with guidance and direction. The authors would also like to thank David Stacey, Don Nelson, and Dawid Broda for providing constant feedback and encouragement.

Finally, the authors would like to thank Mikhail Zaturenskiy and Fabien Petitcolas for their work in developing their MP3 Steganography tools and for taking the time to communicate with them.

7. REFERENCES

[1] Bosi, Marina, and Richard E. Goldberg. Introduction to Digital Audio Coding and Standards. Boston: Kluwer Academic, 2003. Print.

[2] Edström., Björn. "Blog.bjrn.se." Blog.bjrn.se. N.p., n.d. Web. 25 Nov. 2014.

[3] Hacker, Scot. MP3: The Definitive Guide. Sebastopol: O'Reilly Media, 2000. Print.

[4] ISO/IEC 11172-3 – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s, 1993.

[5] Khodja, Ben. "Probabilistically Detecting Steganography within MP3 files." Illinois Institute of Technology, 2014.

[6] Maciak, Lukasz G., Michael A. Ponniah, and Renu Sharma. MP3 Steganography: Applying Steganography to Music Captioning.

[7] Nilsson, Martin, "ID3 tag version 2.3.0", 1999 ID3, http://www.id3.org/id3v2.3.0

[8] Raissi, Rassol. The Theory Behind MP3.

[9] Ruckert, Martin. Understanding MP3: Syntax, Semantics, Mathematics, and Algorithms. Wiesbaden: Vieweg, 2005. Print.

[10] Supurovic, Predrag, "MPEG Audio Frame Header", 1999 DataVoyage, http://www.datavoyage.com/mpgscript/mpeghdr.htm

[11] Thiagarajan, Jayaraman Jayaraman., and Andreas Spanias. Analysis of the MPEG-1 Layer III (MP3) Algorithm Using MATLAB. San Rafael, CA: Morgan & Claypool, 2012. Print

[12] Zaturenskiy, Mikhail. "MP3 Files as a Steganography Medium." Illinois Institute of Technology, 2013.