ISBN: 978-981-94-1156-6
2024 the 14th International Workshop on Computer Science and Engineering (WCSE 2024)
doi: 10.18178/wcse.2024.06.055

Secure Information Hiding in Audio Digital Files

Bruno Carpentieri [+]

Dipartimento di Informatica, Universit àdi Salerno, Italy

Abstract. In this paper we discuss the possibility of hiding information in audio digital files by using the Echo Hiding technique that involves the insertion of information within audio tracks by adding echoes according to certain rules within the track that you want to modify. The main problem we copy with it is the insertion of information without the user playing the modified track being able to distinguish it from the original one. Here we present our implementation of a variation of the Echo Hiding technique and its experimental evaluation that gives confirmation of its high potentiality.

Keywords: Echo Hiding, Steganography, Audio Digital Files, Information Hiding, Data Compression, AES, MP4-ALS

1. Introduction

Steganography consists of the set of techniques that allow information to be hidden in the context of a communication between two or more interlocutors. It includes that set of techniques that allow you to hide messages or information within any type of data, including audio tracks. Here we are interested in audio steganography. Audio steganography techniques consist of hiding information within an audio file (typically speech or songs), varying their characteristics in terms of signal amplitude or volume of certain sections of the track. The main problem appears to be the actual insertion of information without the user playing the modified track being able to distinguish it from the original one. In this paper we discuss Echo Hiding (see for example D. Gruhl, A. Lu and W. Bender in [1] and H. B. Dieu in [2]) that involves the insertion of information within audio tracks by adding echoes within the track that you want to modify, according to certain rules.

This paper is organized as follows: in Section 2 we discuss Echo Hiding and our variation of this technique. In Section 3 we discuss the experimental results we have obtained on a test data set of nine digital audio files and in Section 4 we present our final conclusions and possible new research directions.

1.1. Audio Digital Files

An Audio File Format is a digital format for storing files containing audio (dialogue, music, sounds, etc.). Digital audio can be in compressed or uncompressed format. Compressed files can be compressed lossless (i.e., without loss of information) or lossy (with loss of information). The lossless compression of audio files is almost always based on predictive coding schemes (see for example the MPEG-4 Audio Lossless Coder, also known as MPEG-4 ALS) while other popular data compression schemes are not commensurate to this type of data. For example, Lempel Ziv Coding, which is often used both in the context of one-dimensional data and in image compression (see for example [3] and [4]) does not seem to perform adequately on this type of data nor the ad hoc encoding techniques that are often used for other types of data (see for example [5]). The relationship between data compression and cryptology has been experimentally studied in [6].

Waveform Audio File Format (WAV) is a standard audio file format, developed by IBM and Microsoft, to store and transmit an audio bitstream. It is used in Microsoft Windows systems for uncompressed audio. The usual bitstream encoding is the LPCM (Linear Pulse-Code Modulation). The WAV format is an application of the bitstream Resource Interchange File Format (RIFF) for storing data in blocks. The RIFF format serves as a "wrapper" for different audio encoding formats. Although WAV files are generally

[+] *E-mail address*: bcarpentieri@unisa.it.

uncompressed audio in the linear pulse-code modulation (LPCM) format, WAV can also include compressed audio, in this paper, only uncompressed WAV files will be considered.

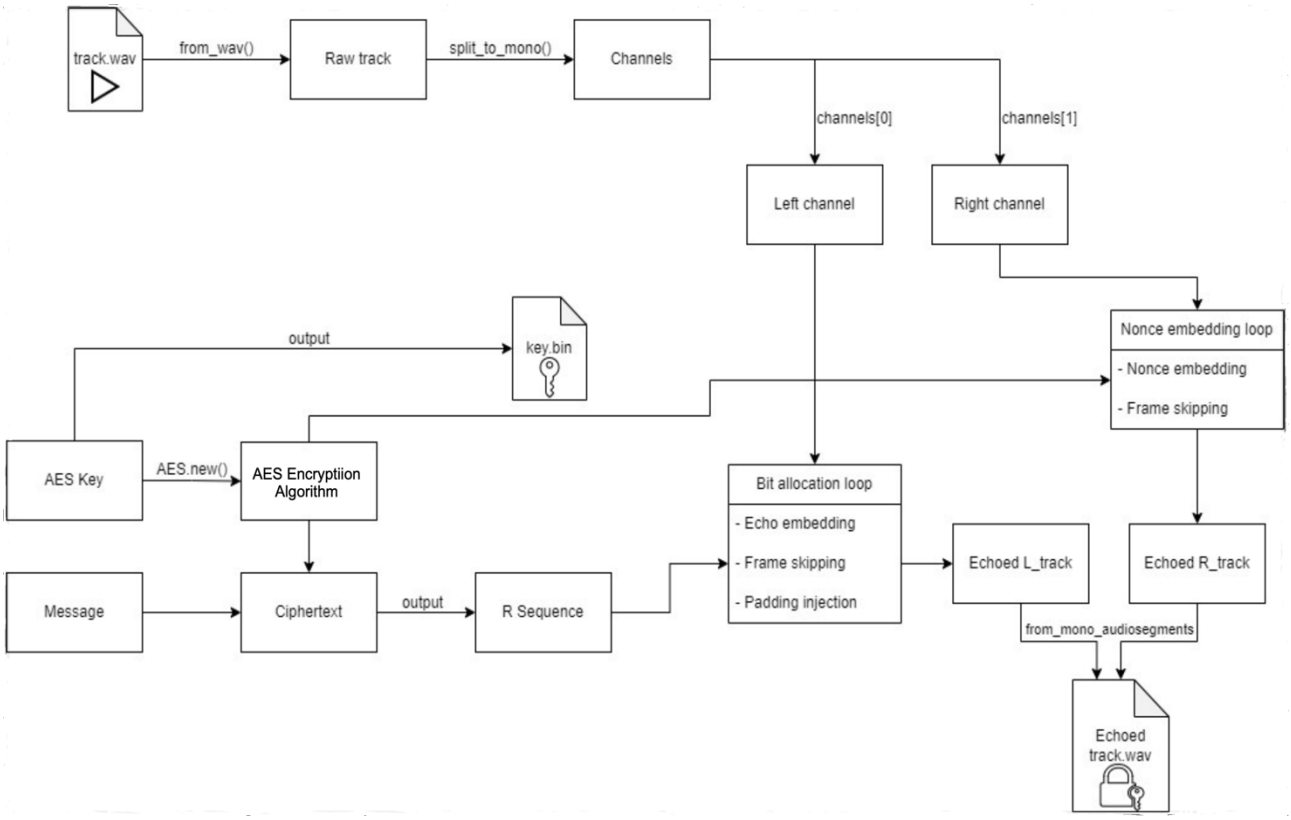

Fig. 1: The Echo Hiding encoding algorithm.

2. Echo Hiding

The concept behind Echo Hiding is very simple. The technique works by receiving an audio track as input. This track will be modified in several places by adding a certain amount of echo. It is, of course, necessary to establish in advance the intensity of the echo applied to the target portion and how late to apply it: this is one of the key points in which all the echo hiding algorithms differ. The data that will be hidden in the audio track consists of binary strings, this is particularly useful because it allows you to hide not only text (made up of ASCII characters), but any type of data (from images to small executables).

We have developed our implementation of the Echo Hiding Technique by opportunely modifying the work described in [2]. Our implementation has used the Python programming language and has been successfully tested on different platforms (Windows 11, Mac OS Ventura 13.0.1, Linux). The Echo Hiding process consists of two algorithms: one for coding, i.e. embedding the bits within the track and one for decoding, i.e. extracting the bits from the modified audio track and subsequently recomposing the message.

2.1. Coding

The coding algorithm takes as input: the message *M* to be hidden (in our case it is an ASCII text string), of length *L*, the audio track *T* in which to insert echo (typically a WAV file: a lossless audio coding format), a pair of integers: *k = (a, seed)* which represents the key of the algorithm.

Figure 1 represents the block diagram of our encoding algorithm. The algorithm proceeds to create a binary representation of the message, thus transforming it into a string composed of 0s and 1s, in which each character of the ASCII text will be encoded with 8 bits. We call this sequence *M'* and its length *L'*: the algorithm from here on will continue considering this representation of the message.

This sequence is defined as a function of *k*, as described in the following snippet:

ALGORITHM 1: GenerateR

Input: $k = (a, seed), L'$
Output: $R=R_1,R_2,\ldots,R_{L'}$
$currentNumber \leftarrow seed$
for i from 1 to L' **do**
    $x \leftarrow (a \times currentNumber) + (2 \times a)$
    if $(x \bmod 6) > 2$ **then** $R[i] \leftarrow True$
    else if $(x \bmod 6) \leq 2$ **then** $R[i] \leftarrow False$
    end if
    $currentNumber = x$
end for

Once the *R* sequence has been generated, we move on to the actual embedding phase. This phase therefore involves adding a certain amount of echo to the audio track. When an audio track is sampled to store it in a digital medium, it will be composed of a certain number of samples, which represent the (numerical) value of the audio signal at a specific moment in time. Depending on the sampling rate, an audio track will have more or fewer samples.

The Echo Hiding coding algorithm works by considering frames, i.e. data sets made up of 1024 samples each. A bit is hidden within each frame by adding an echo. In practice, adding echo to a digital track means altering the numerical value of the samples. Let's consider the following situation: let b_i be the *i*-th bit to hide in the trace. Bit b_i will be hidden in the *i*-th frame: let's call it f_i. Four cases can be distinguished:

1. If the *i*-th component of the *R* sequence (*R[i]*) has a value of 1 and $b_i = 1$, then no echo is added to the audio track and the f_i samples are not altered.

2. If *R[i]* = 1 and $b_i = 0$ then echo is added to the track, altering the value of the samples of f_i.

3. If *R[i]* = 0 and $b_i = 1$ then echo is added to the track, altering the value of the samples of *fi*.

4. If *R[i]* = 0 and $b_i = 0$ then no echo is added to the track and the *fi* samples are not altered.

We call this procedure *EmbedMessage(T, M', R)*.

Once the frames have been processed and subjected to the embedding process, the main part of the algorithm ends, and a new audio track is created which contains the first *L'* frames of the audio track, which may or may not have undergone modification (depending on the *R* sequence) and all other frames of the edited audio track that were not edited. The entire procedure ends by providing as output the edited audio track and the *L'* value.

The addition of actual echo to the track is carried out considering two parameters: a numerical value representing the *gain* value in decibels and the *delay*, i.e. a value that represents (in milliseconds) from which point to start applying the *gain*. Below is the pseudocode of the Echo Hiding encoding algorithm. *T'* indicates the modified audio track produced by the coding algorithm.

ALGORITHM 2: Echo Hiding Encoding

Input: $M, k=(seed,a), T$
Output: T'

$R \leftarrow$ GenerateR(k)
$M' \leftarrow$ ConvertToBinary(M)
$F =$ GroupInFrames(T)

if $|F| < |M'|$ **then** exit()
else
    $T' \leftarrow$ EmbedMessage(T,M',R)
    $T' = T' \cup$ *all the other frames that have not been modified*
end if

return T'

2.2. Decoding

The decoding algorithm takes as input: the audio track into which the echo was inserted, the original audio track, a pair of integers $k = (a, seed)$ which represents the secrete key of the algorithm, the length of the message M.

The algorithm works in a mirror image to the coding one, that is, initially, it generates the pseudorandom accessory binary sequence R, of length L'. It then divides the audio track into frames made up of 1024 samples each and obtains 1 bit per frame in the following way:

• Compare the original track with the modified one:

• For each frame i:

– If i differs in the edited track, 0 or 1 is extracted based on $R[i]$.

– Repeat L' times

• The binary message obtained from the sequence of bits b is returned

2.3. Parametrization of the Algorithm

The version of Echo Hiding proposed by [2] involves inserting the message at the beginning of the track. In the case of embedding very long messages, this could lead to the listener's perception of audio artefacts like “static” noises, very similar to those of FM radio in the presence of a very weak signal. We therefore proceeded to add support for selective embedding of bits within frames.

Formally, this modification involves defining a value n, that is chosen in function of the lengths of both the audio and of the encoded message, such that if a bit is inserted inside the frame i, the next bit will be hidden inside the frame $i + (n + 1)$. This way the echo will be distributed across the entire track rather than just at the beginning. If $n = 0$, then the implementation is identical to the traditional one.

From an implementation point of view, this strategy is implemented by defining a *frames_to_skip* variable whose value must be identical for encoder and decoder.

In the main loop of the encoding algorithm, a binary variable *skip* is used that keeps track of whether a frame needs to be skipped in the current iteration. Moreover, at each iteration, before moving on to the next iteration, the *skip* variable is incremented by 1 and the operation $skip = skip \% (frames\_to\_skip + 1)$ is performed.

For example, if you want to insert echo in one frame and not in the two immediately following ones, *frames_to_skip* must be set to 2 and consequently the *skip* variable can only take on values in $X = \{0,1,2\}$. During the execution of the main loop, the *skip* value is checked and if it is greater than 0, then the frame is skipped, otherwise the embedding procedure is started.

2.4. Support for Bit Embedding in both Channels of the Audio Track

Nowadays, most audio tracks are stereo, that is, they have two channels (left and right), which for most of the duration of the track are identical but can be differentiated if necessary, during the audio encoding to generate a particular sound divergence effect.

The presence of this double channel represents a concrete possibility of extending the Echo Hiding technique, exploited because of the introduction of symmetric key cryptography, described in the next section.

2.5. Message Encryption Using a Symmetric Key System

Echo Hiding is a steganography algorithm, with everything that entails. The information is hidden but not encrypted: if in fact the scheme were to be discovered and the parameters that make up k were identified (when the pair $(a, seed)$ is made up of relatively small numbers) it would be immediately being able to trace the information by following the rules described by Dieu in [2].

For this reason, it was decided to add an additional security layer, implementing a private key encryption scheme and for this purpose, the AES algorithm was chosen.

In our version of Echo Hiding the plaintext message is no longer hidden but the encrypted one and this improves security as shown in [7].

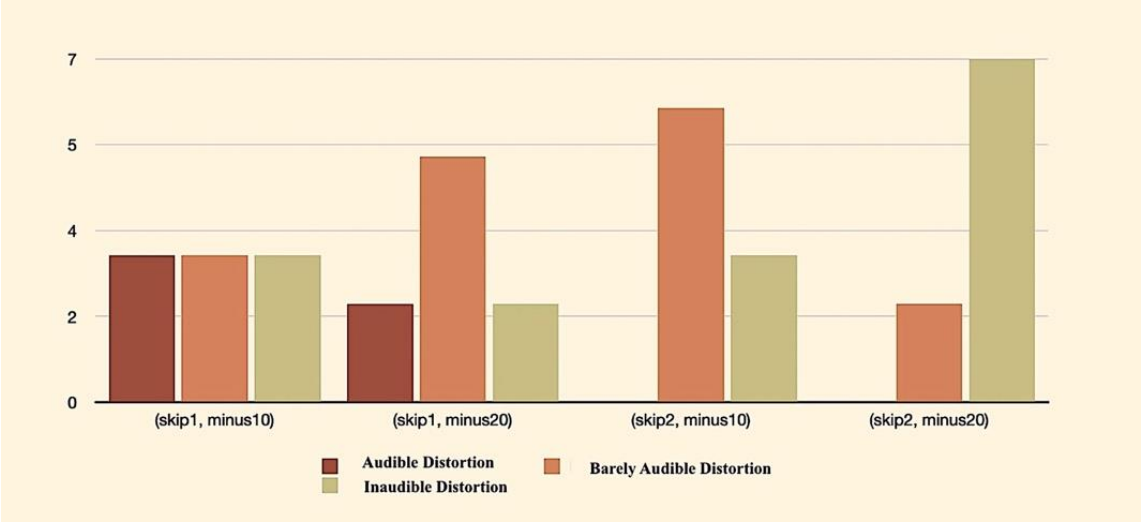

Fig. 2: Distortion obtained in function of the *(skip,gain)* pair.

3. Testing

The testing activities carried out following the implementation we propose evaluated the performance of the algorithm in terms of artefacts and audio distortions introduced due to the addition of the echo to the track. For this testing activity, nine audio tracks of different types were used, and echo was applied using different parameters. The names of the tracks are, grouped by type: **Speech-type tracks:** voice_herm_stereo.wav, low_quality.wav voice. **Instrumental tracks:** dreams.wav, lifelike.wav, powerfulbeat.wav, electronic_future.wav. **Tracks with vocals:** christmas.wav, synth.wav. **Tracks with natural noises:** wood.wav.

For each of these tracks, messages were embedded by using the following parameter combinations: *frames_to_skip* value set equal to 1 (*skip1*), *gain* value set to -20 (*minus20*), *frames_to_skip* value set equal to 1 (*skip1*), *gain* value set to -10 (*minus10*), *frames_to_skip* value set equal to 2 (*skip2*), *gain* value set to -20 (*minus20*), *frames_to_skip* value set equal to 2 (*skip2*), *gain* value set to -10 (*minus10*), thus, obtaining a total of 36 generated tracks. Testing was carried out by embedding messages of approximately 200 bits because we wanted to operate in the same conditions for all the traces which are of different lengths. For shorter tracks, the message length was adapted to the maximum number of embeddable bits. It should be underlined that, with every combination of values, none of the tracks has an echo perceptible to the human ear. However, there are sporadic noises like white noise, probably attributable to the necessary addition of padding caused by the malfunction of *FFmpeg* software suite we used in our implementation, which however can be attenuated by varying the parameters. Furthermore, regardless of the variation of the parameters it was always possible to extract the hidden message.

Table 1 and Figure 2 show the experimental results we have obtained on the test data set. As shown in Figure 2, generally the best combination of parameters regarding *frames_to_skip* in the embedding process and gain to apply is (*skip2*, *minus20*). Furthermore, note that in the case of audio tracks with quality comparable to those produced by a common microphone (such as *voice_low_quality.wav*) no distortions are perceivable regardless of the parameters used.

4. Conclusions and Future Work

In this paper we have discussed the possibility of hiding information in digital file. Here we present our implementation of a variation of the Echo Hiding technique and its experimental evaluation that gives confirmation of its high potentiality. We have performed experiments on nine audio files, that were different in type and usages, and the results obtained have been very encouraging. Moreover, in the case of audio tracks with quality comparable to those produced by a common microphone (such as the test file voice_low_quality.wav) no distortions are perceivable regardless of the parameters used. Future work involves a deeper experimentation and the research of a better way to spread the echoes on the audio file.

Table 1: Experimental results

Audio Track	*Frames to skip*	*Gain*	*Result: distortion*	*Audio Track*	*Frames to skip*	*Gain*	*Result: distortion*
voce_herm_stereo	*Skip1*	*Minus10*	**barely aud.**	*electronic_future*	*Skip1*	*Minus10*	**audible**
	Skip1	*Minus20*	**barely aud.**		*Skip1*	*Minus20*	**barely aud.**
	Skip2	*Minus10*	**barely aud.**		*Skip2*	*Minus10*	**barely aud.**
	Skip2	*Minus20*	**inaudible**		*Skip2*	*Minus20*	**inaudible**
voce_low_quality	*Skip1*	*Minus10*	**inaudible**	*christmas*	*Skip1*	*Minus10*	**barely aud.**
	Skip1	*Minus20*	**inaudible**		*Skip1*	*Minus20*	**barely aud.**
	Skip2	*Minus10*	**inaudible**		*Skip2*	*Minus10*	**barely aud.**
	Skip2	*Minus20*	**inaudible**		*Skip2*	*Minus20*	**inaudible**
dreams	*Skip1*	*Minus10*	**audible**	*synth*	*Skip1*	*Minus10*	**audible**
	Skip1	*Minus20*	**audible**		*Skip1*	*Minus20*	**audible**
	Skip2	*Minus10*	**barely aud.**		*Skip2*	*Minus10*	**barely aud.**
	Skip2	*Minus20*	**barely aud.**		*Skip2*	*Minus20*	**inaudible**
lifelike	*Skip1*	*Minus10*	**inaudible**	*wood*	*Skip1*	*Minus10*	**barely aud.**
	Skip1	*Minus20*	**inaudible**		*Skip1*	*Minus20*	**barely aud.**
	Skip2	*Minus10*	**inaudible**		*Skip2*	*Minus10*	**inaudible**
	Skip2	*Minus20*	**inaudible**		*Skip2*	*Minus20*	**inaudible**
powerfulbeat	*Skip1*	*Minus10*	**inaudible**				
	Skip1	*Minus20*	**barely aud.**				
	Skip2	*Minus10*	**barely aud.**				
	Skip2	*Minus20*	**barely aud.**				

5. Acknowledgement

This work was supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU. The author wants to thank his students Alessandra Zullo, Gabriele Lombari, Mario Villani, Paolo Marrone, Paolo Vigorito, Andrea di Pierno, Marco Russo e Hermann Senatore that, in different years, have contributed to this research producing preliminary implementations of the technique we have presented.

6. References

[1] Daniel Gruhl, Anthony Lu and Walter Bender, Echo hiding. In *Proceedings of the International Workshop on Information Hiding*. LNCS, Volume 1174. 1996.

[2] Huynh Ba Dieu. An improvement for hiding data in audio using echo modulation. In *Proceedings of the Second International Conference on Informatics and Engineering and Information Science (ICIES 2013)*. 2013.

[3] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*. Volume: 24, Issue: 5, September 1978.

[4] Francesco Rizzo, James A. Storer and Bruno Carpentieri. LZ-based image compression. *Information Sciences*. Volume 135, Issue 1-2, Pages 107 - 122, June 2001.

[5] Raffaele Pizzolante and Bruno Carpentieri. Multiband and lossless compression of hyperspectral images. *Algorithms*. Volume 9, Issue 1. 2016.

[6] Bruno Carpentieri. Efficient compression and encryption for digital data transmission. *Security and Communication Networks*. art. no. 9591768. 2018.

[7] Khairul Muttaqin and Jefril Rahmadoni. Analysis And Design of File Security System AES (Advanced Encryption Standard) Cryptography Based. *Journal of Applied Engineering and Technological Science (JAETS).* Volume 1, Issue 2, 2020.