Journal homepage www.jzs.uos.edu.krd
Journal of Zankoi Sulaimani
Part-A- (Pure and Applied Sciences)

Using Rubik's Cube and a Modified LSB for Audio Steganography

Kaziwa Saleh , Mihran Muhammad , Khanda Ahmed

Department of Computer Science, School of Science, University of Sulaimani, Sulaimani, Kurdistan Region - Iraq
email: Kaziwa.saleh@univsul.edu.iq

Article info

Original: 2 May 2015
Revised: 2 June 2015
Accepted: 25 June 2015
Published online:
20 Dec. 2015

Key Words:
Audio Steganography LSB
Rubik's Cube

Abstract

Nowadays steganography plays an important role in information security. There are various techniques to apply steganography; one such technique is Least Significant Bit (LSB). Due to consecutive embedding of data, LSB is highly vulnerable to data retrieval and manipulation. This study proposes a mixture between Rubik's cube principle to scramble the audio data, and a modified LSB technique to hide the secret data. The modified LSB technique includes embedding using only irredundant bits of binary representation of each character in the secret message, and hiding in the lowest sample between two consecutive samples of the cover audio. The used technique makes the retrieval of secret message harder because it adds two levels of protection (scrambling, and hiding in the lowest sample) against the attempts of obtaining data; and makes the embedded data imperceptible.

Introduction

Exchanging data has been carried out for decades fairly smoothly with the availability of Internet as well as communication systems. Although today more advanced systems are available, data cannot be sent over the network openly for security reasons. The problem lies in the fact that the network is usually open to every user; sending secret data can be an easy target for illegal users who may tamper it easily. For protecting the process of data transmission, the senders can resort to data shielding techniques which can hide the data and as such protecting it from any attempts of tampering [1]. One of the data protection techniques is steganography, which is known as 'covered writing'. Steganography is "the art of hiding and transmitting data through apparently innocuous carriers in an effort to conceal the existence of the data" [2]. The carrier can be an image, a text, an audio, or a video file.

This study focuses on audio steganography which is an approach to hide the secret messages in an audio file. There are many algorithms that have been developed over years to achieve this in a way that is both imperceptible and has high payload. A survey of current audio steganography methods can be found in [3].

One of the most popular algorithms is LSB. This is one of the earliest and easiest techniques studied in the information hiding of digital audio (as well as in other media types). In this technique LSB of binary sequence of each sample of digitized audio file is replaced with binary equivalent of secret message. Since the least significant bits are last binary bits, their modification will not have significant effect and it cannot be perceived by human ears [4].

The problem of data hidden with LSB is that it is very easy to retrieve and alter; simply by taking the least significant bit of every byte the hidden information can be obtained. However, if the data is embedded in a random way then the retrievable data would be meaningless. This paper proposes implementing Rubik's cube methodology for scrambling the audio data before embedding anything in it. This results in spreading the secret data in the audio file. The authors in [5] have applied Rubik's cube principle in colored image for steganography while [6] have implemented the same principle in grayscale image for watermarking. Also [7] and [8] have implemented Rubik's cube idea in image encryption and steganography.

In this paper, after the data is scrambled, the secret data is embedded using a modified LSB technique in the sample with lowest value between two consecutive samples, and finally the cover data is descrambled to return it to its original form. This method adds two levels of protection to the hidden data; meanwhile it increases the amount of data that can be made secretive.

Proposed Method

A. Data Hiding

In this proposed method, the message is embedded in the cover media through four phases (Figure-1).

A.1 Scrambling Phase

First Rubik's cube principle is used to shuffle the cover data. Because audio data is one dimension vector, the shuffling cannot be performed on it directly, thus the data is converted into a two dimensional array. If the size of the audio is not sufficient to fill the matrix, then the last row can be discarded.

Then, the data is scrambled using the following algorithm (Figure-2):

1. Scramble the rows:
   - For the even rows, shift by X to the right.
   -For the odd rows, shift by X to the left.
2. Scramble the columns:
   - For the even columns, shift by X upwards.
   - For the odd columns, shift by X downwards.

X is the number of the desired shifts. Value of X can be any random number smaller than number of columns or rows.

A.2 Comparison Phase

To add another level of protection, instead of embedding in sequential samples, the data is hidden in the lowest sample between two consecutive samples. For this, a comparison is done between the two samples, the result is observed as the following:

- If they are equal, they are skipped and the next two samples are chosen. Because, embedding may make the altered sample bigger than the other one and this results in extracting an incorrect bit.
- If the difference between the two samples is 1, the data is embedded in the lowest sample, and the resulted sample is compared with the other one, if the difference between them has become 0, these two samples are skipped and the next two samples are chosen to hide the same bit.
- If the difference between them was more than 1, the data is embedded in the lowest sample.

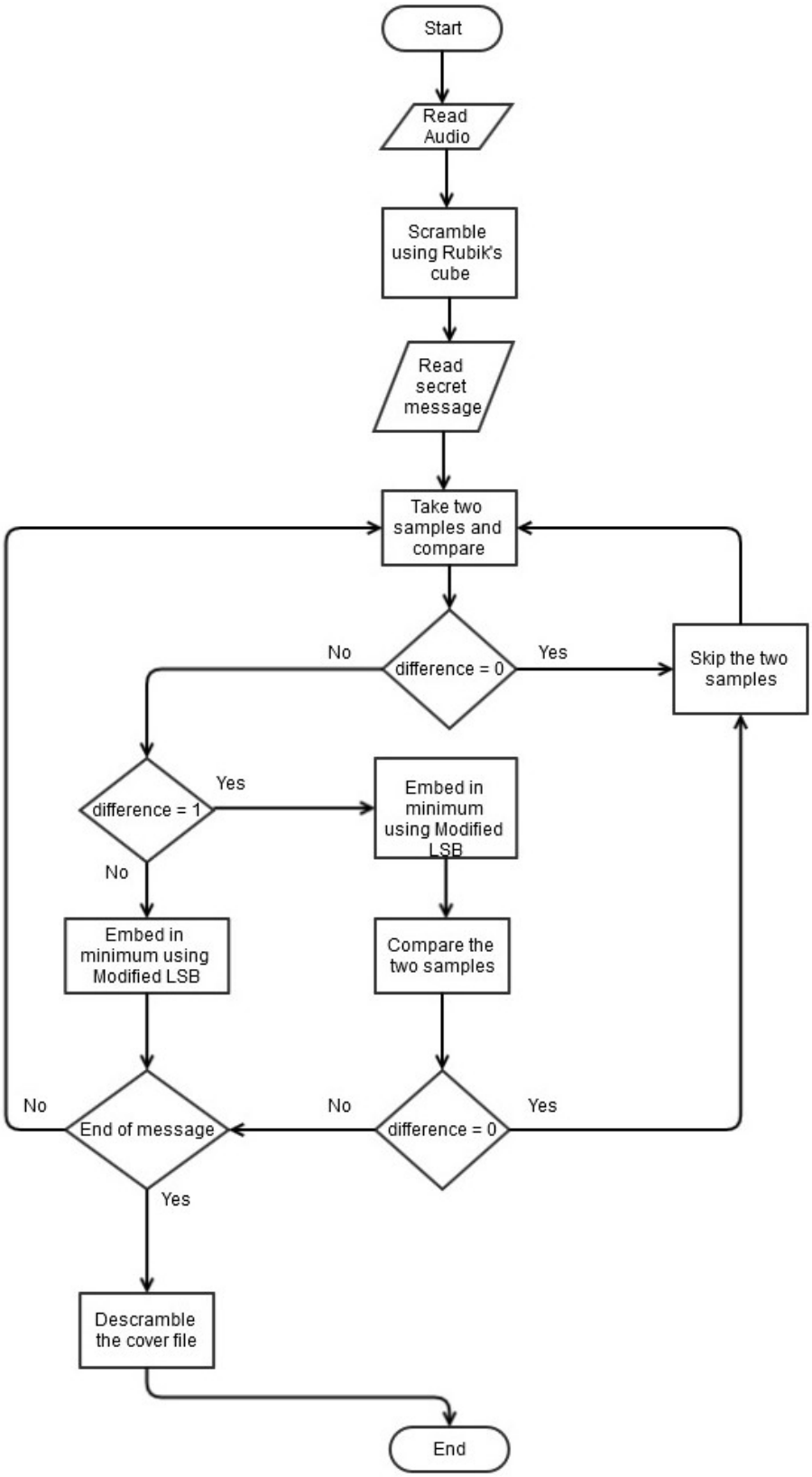

Figure 1: Flowchart of the proposed method

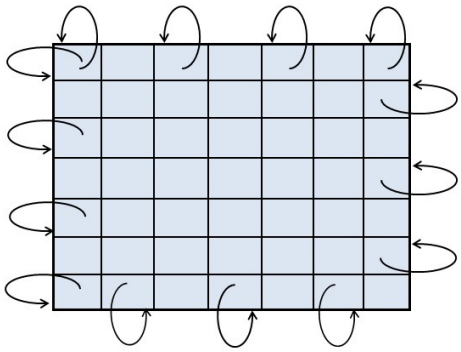

Figure 2: Scrambling the data

A.3 Embedding Phase

Zaher in [9] proposes the use of only 5 bits for embedding each uppercase or lowercase letter or numbers. The idea of the algorithm is based on the redundancy of the last 4 bits of each of alphabetic characters and numbers. It can be seen that there is a repeated sequence between capital and small letters representation. In the case of small letter if we take the binary code of last four bits, it will be "0110" or "0111". For capital letter the last four bits will be "0100" or "0101"; which means the last three bits are identical so they can be discarded. Therefore only 5 bits can be embedded instead of 8 bits. In case of small letters from 'a' to 'o', the four bits are the irredundant bits (the first 4 bits of each character) and the fifth bit to represent the first sequence from 01h to 0Fh. If the fifth bit equal to 1, we represent the second sequence from 0h to Ah, that is small letters from 'p' to 'z'; and so on [9].

To distinguish between small letters, capital letters, numbers, and space, Zaher is using another 5 bits as a control character, namely 1Bh, 1Ch, 1Dh, and 1Fh; which are encoded before each set of corresponding characters.

The problem with this technique is that it cannot be used to hide a secret message that contains special characters such as '@, ., ,, ;, (,), …etc.'; simply because they will be confused with uppercase and lowercase letters, even with using control characters.

Therefore we consider modifying the above technique to include special characters too. If we look at binary code for the special characters starting from '00100001' to '01111110' we can group them into 2 groups: the first group from '00100001' to '00111111', and the second group from '01000000' to '01111110'. The last 3 bits are similar in the first group, so we can use the first 5 bits of each character with one other bit (0) manually inserted into the fifth position, which makes it 6 bits in total (This is to distinguish them from control characters used to differentiate between uppercase, lowercase letters, and numbers).

For example the binary code for ';' is '00111011, we take the first 4 bits (1011) and manually insert '0' to the fifth position and then we put the fifth bit of the binary code (1) to the sixth position. By the end, we embed '101011' instead of '00111011'.

In the second group the last 2 bits are identical; hence we can use the first 6 bits of each character and manually put a '0' to the fifth bit, thus 7 bits are embedded.

Consequently the modified algorithm[1] deals with the following cases:

- For lowercase letters, embed the control character '1Bh' before embedding the 5 bits of the small letter.
- For uppercase letters, put the control character '1Ch' before the 5 bits of the capital letter.
- In case of space, embed only its control character '1Dh' without embedding its binary code.

[1]This algorithm can be considered as a lossless compression algorithm.

- For the first group of the special characters along with numbers, use the control character '1Eh', then embed 5 bits of binary code for the numbers (1-9), otherwise embed 6 bits for the special characters.
- Use '1Fh' as a control character for the second group of special characters, before embedding the converted 7bits.

A.4 Descrambling Phase

Once the embedding phase is complete, the cover data is descrambled to recover the audio file. The descrambling is done in the opposite direction of the scrambling, i.e. for the even columns; the samples are shifted by X downward, and the odd ones upwards, then the rows are descrambled. (Figure-3), (Figure-4), and (Figure-5) illustrate wavelet of 'audio sample 2' before and after embedding, and the difference between them.

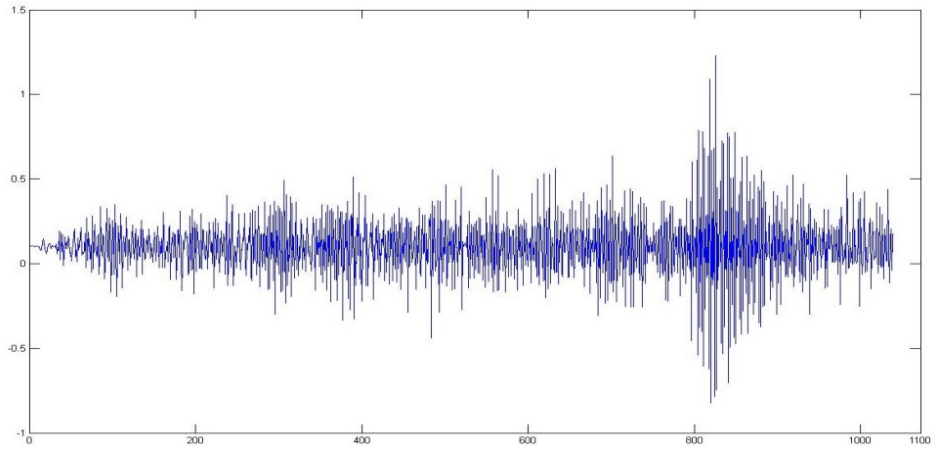

Figure 3: Audio sample before embedding

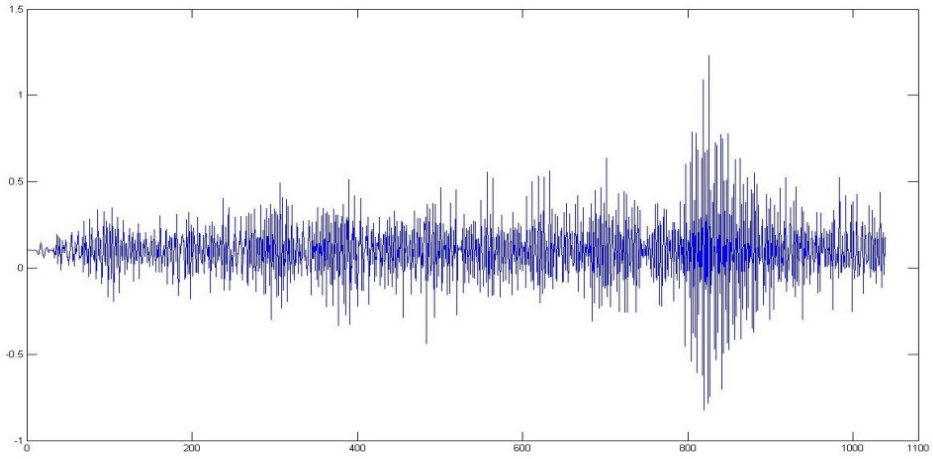

Figure 4: Audio sample after embedding

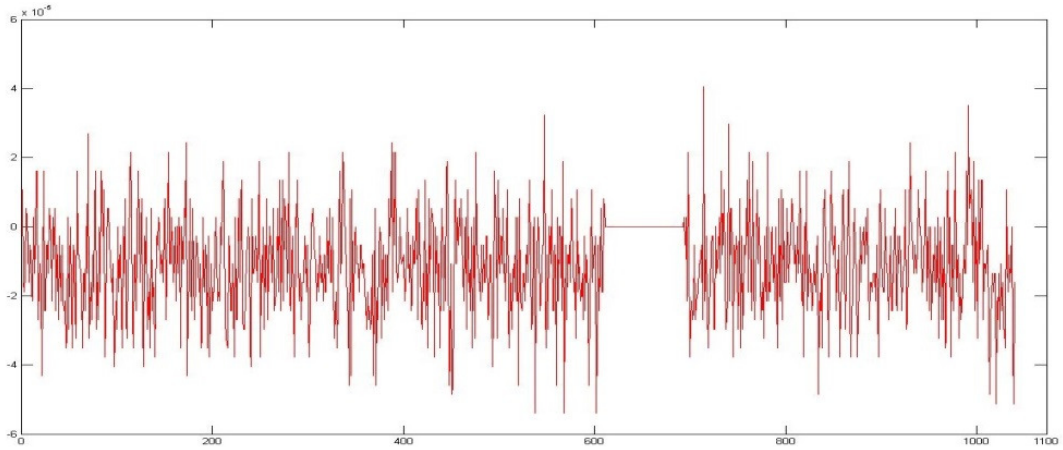

Figure 5: Difference between the original and embedded audio

B. Extraction Phase

To extract the hidden message, the following steps must be performed:

- Scramble the cover data using Rubik's cube principle mentioned in (Section A).
- Find the lowest sample between two consecutive samples (if the two samples are equal, they are skipped and the next two samples are selected for comparison)
- Using LSB algorithm to extract the first 5 bits which represent the control characters:
  - If '1Bh' or '1Ch' then perform 'OR' operation with '60h' on the next 5 bits to obtain 8 bits [9].
  - If '1Dh' then use 20h which represents the ASCII code for space [9].
  - If '1Eh' and the next 5 bits of the data is equal to the first 5 bits of the binary representation for numbers (1–9) then perform 'OR' operation to obtain the ASCII code for numbers. Otherwise get the next bit and remove the fifth bits and perform 'OR' operation to transform to 8 bits.
  - If '1Fh' then get the next 7 bits and remove the bit at the fifth position and do the 'OR' operation.

Results and Discussion

The proposed method was tested for perceptibility and capacity. For testing the perceptibility, 4 stereo audio files of different sizes (bitrate = 1411Kb/s and sampling frequency = 44100 Hz) were used to hide several texts of different sizes, and then SNR (Signal to Noise Ratio) and MOS (Mean Opinion Score) were

Table 1: SNR and MOS results.

Audio Files	Tests	500	1000	2000	3000	4000	5000	8000	15000	16000	20000
Sample1 (1 ms)	SNR	67.7	64.7	61.7	59.9	58.7	57.7				
	MOS	5	4.9	4.8	44.	4.3	4.2				
Sample2 (2 ms)	SNR	86.6	83.6	80.7	79	77.8	76.8	74.7	72	71.7	70.7
	MOS	5	4.9	4.8	4.7	4.6	4.5	4.4	34.	4.2	4.1
Sample3 (2 ms)	SNR	100.5	97.5	94.6	92.8	91.6	90.6	88.6	85.8	85.6	84.6
	MOS	5	5	4.9	4.9	4.8	4.7	4.6	4.5	4.5	4.3
Sample4 (3 ms)	SNR	99	96	92.9	91.2	89.9	89	87	84.2	84	83
	MOS	5	5	4.9	4.9	4.8	4.8	4.7	4.6	4.5	4.4

calculated (Results of SNR and MOS are shown in (Table -1) according to numbers of characters hidden in each audio sample).The MOS was evaluated by listening tests involved 30 people who were asked about the difference between the original audio and the modified audio. To calculate the mean opinion score, five point scales (as shown in (Table- 2) [10]) were used by the individuals after listening to the audio file and the average of all scores was MOS.

Table 2: Mean Opinion Score.

MOS	Impairment
5	Imperceptible
4	Slightly perceptible but not noisy
3	Slightly noisy
2	Noisy
1	Very noisy

The result of MOS in (Table -1) show that, the more characters embedded the more noise is produced and MOS decreases. However, an audio sample with 2ms or more can conceal 16 KB of secret data with keeping a high level of imperceptibility.

(Table -1) also shows that the proposed method can maintain a high SNR throughout the increasing of the amount of data embedded. The results of sample3 give higher SNR than sample2, although they have the same length. This is because the changes that need to occur in the least significant bits of each sample in sample3 are fewer than sample2, and that produces higher SNR.

For testing the payload, three different texts were embedded into the same audio file, namely sample4. The texts were different in the number of flips that happen between the characters. The first text was a simple normal text that had some numbers and special characters in it along with normal statements. However, the frequency of the numbers and special characters were increased in the second text; while the third text contained much more numbers and special characters in it, with an increasing number of flips between small letters and capital letters (The results are shown in (Table - 3)).

Table 3: Payload testing results in sample4.

Methods	Text1	Text2	Text3
Proposed method	20000	19810	17736
Traditional LSB	15302	15302	15302

According to the results, 20KB of normal text can be embedded into sample4; while nearly the same amount of text2 can be hidden, the number of characters of text3 that can be hidden is much less than that of text1 and text2. This is due to higher number of flips between the characters, which needs higher number of bits to be embedded as a control character.

As (Table - 3) shows, the traditional LSB can embed the same amount of text (around 15KB) in the audio sample regardless of the type of the text; however this amount is fewer compared to the amount hidden with the proposed method, considering skipping of a sample between two successive samples.

Conclusion

LSB is a simple and popular algorithm with a high capacity of hiding data; however it is vulnerable to data manipulation and retrieval. Rubik's cube is a mechanism for scrambling data. This paper has implemented the idea of Rubik's cube along with a modified LSB technique to ensure the diffusion of the secret data among the cover media. The technique used provides the secret data with two levels of protection against the attempts of its retrieval. The test results show that this technique keeps the SNR of the cover data high and it increases the payload of the cover file, unless the text contains flips between each character, otherwise the payload would decrease (But such case is rare). However, if an attack is attempted to change the cover file,

the secret message cannot be retrieved successfully. Although this may sound as a disadvantage, it can ensure that unintended recipient of the secret message cannot get the message correctly.

References

[1] A.P. Sherly, Sapna Sasidharan, Ashji S. Raj, P. P.Amritha. A novel approach for compressed video steganography. In: N. Meghanathan, S. Boumerdassi, N. Chaki and D. Nagamalai recent trends in network security and applications. Berlin Heidelberg: Springer. 567-575 (2010).

[2] Johnson, N., Duric, Z. and Jajodia, S. Information Hiding: Steganography and Watermarking-Attacks and Countermeasures. Boston, MA: Springer US (2001).

[3] Nosrati, M., Karimi, R. and Hariri, M. Audio Steganography: A Survey on Recent Approaches. World Applied Programming, 2(3), pp.202-205(2012).

[4] Kumar, H. and Anuradha. Enhanced LSB technique for Audio Steganography. In: Third International Conference on Computing communication and networking technologies. IEEE, pp.1-4 (2012).

[5] Amirtharajan, R., Abhiram, M., Revathi, G., Reddy, J., Thanikaislevan, V. and Rayappan, J. Rubik's cube: A Way for random image steganography. Research Journal of Information Technology, 5(3), pp.329-340(2013).

[6] Yen, E. and Lin, L. Rubik's cube watermark technology for grayscale images. Expert Systems with Applications, 37(6), pp.4033-4039(2010).

[7] Loukhaoukha, K., Chouinard, J. and Berdai, A. A secure image encryption algorithm based on Rubik's cube principle. Journal of Electrical and Computer Engineering, 2012, p.7(2012).

[8] Rhine, R. and T Bhuvan, N. Image Scrambling Methods for Image Hiding: A Survey. International Journal of Computer Science and Information Technologies, 5(1), pp.751-755(2014).

[9] Abu Zaher, M. Modified Least Significant Bit (MLSB). Computer and Information Science, 4(1), p.60 (2010).

[10] P. Adhiya, K. and A. Patil, S. Hiding Text in Audio Using LSB Based Steganography. Information and Knowledge Management, 2(3), pp.8-14(2012).