Performance analysis of current data hiding algorithms for VoIP

Harrison Neal and Hala ElAarag
Department of Mathematics and Computer Science
Stetson University
DeLand, FL, USA
{hneal,helaarag}@stetson.edu

Keywords- VoIP, G.711, steganography, covert communication

Abstract

Steganography techniques attempt to hide pertinent information inside of other harmless information, so as to avoid detection by an adversary. A good amount of research has been done thus far on mediums that aren’t processed in real-time. User-friendly products using these techniques can easily be purchased by average consumers. Real-time mediums have recently received attention as VoIP and other real-time media has entered the mainstream. Several approaches for hiding information within VoIP streams in real-time have been proposed, but little has been done to compare the performance of the algorithms proposed in the literature. In this paper, we test several current data hiding techniques on a variety of G.711 audio recordings, with the intent of giving readers a clearer understanding of which of the algorithms would best suit their purposes. We use important performance metrics to evaluate the algorithms, namely, throughput, noise-to-signal ratio and the Perceptual Evaluation of Speech Quality algorithm. Our results show that the method by Aoki allows for best throughput during silence and low volume conditions, and both methods by Ito et al, and Miao and Huang offer good throughput in noisy environments. Vulnerability to steganalysis is also considered. We devise a technique that illustrates that the algorithm by Miao and Huang is detectable as well as other LSB-based algorithms which have already been shown to be detectable by other means.

1. INTRODUCTION

Steganography is the science of covert communication. Typically, a cover message that may be observed by an adversary is modified in a way to hide a hidden message at the cost of reducing quality and/or user experience by a negligible amount [1]. Papers on steganography originally focused on mediums that weren’t processed in real-time, such as images and audio files. As the Internet grew in popularity, modifying redundant protocol headers was proposed [2]. When real-time mediums such as Voice over Internet Protocol (VoIP) calls permeated mainstream audiences, attention to these mediums increased in the literature. Recently proposed algorithms include approaches by Ito et al [3], Aoki [4] and Miao and Huang [5]. The approach by Ito et al [3] uses a lower bitrate codec to process audio in tandem with G.711, and will determine how many least significant bits of the G.711 audio can be tampered with while still preserving full quality of the lower bitrate codec. Aoki [4] designed an algorithm that treats the sign bit of G.711 audio as a least significant bit when the magnitude is 0, and is considered lossless by the author. Finally, Miao and Huang [5], rather than manipulating bits, start by taking the average signed magnitude of a group of G.711 samples, finding the difference between each sample’s magnitude and the average, and manipulating that difference to encode data, and claims this algorithm avoids detection by typical steganalysis techniques. In section 2, we give a background about the G.711 codec. Section 3 presents the related work in the literature, and in section 4 we evaluate those algorithms. Finally, we conclude our paper in section 5.

2. BACKGROUND

G.711 is the oldest voice codec in use not only by today’s VoIP systems but also domestic public switching telephone networks (PSTNs) [6]. Some standardization authorities consider G.711 mandatory for VoIP devices as a codec common to all systems for interoperability [7]. In regards to steganography, there appears to be a much greater wealth of available research on G.711 as opposed to other voice codecs.

As illustrated in Figure 1, the G.711 codec takes signed linear audio samples at 8kHz [8]. For each sample, the codec outputs one byte (eight bits); this results in a 64kbits/s bitrate. There are three components in the output: the sign (positive or negative, represented by s in Figure 1), the magnitude on a logarithmic scale and a sub-magnitude on a linear scale (represented by abcd in the figure). The first bit of output is the sign bit and is identical to the sign bit from the input. The second through fourth output bits represent the magnitude of the input on a logarithmic scale. The codec checks the linear sample input for how many “0” bits follow the sign bit before a “1” bit is found, with more “0” bits before the first “1” implying a lesser magnitude. If a “1” bit immediately follows

<table>
<thead>
<tr>
<th>s1abcdefgijkl → s11abcd</th>
<th>s01abcdefgijkl → s10abcd</th>
</tr>
</thead>
<tbody>
<tr>
<td>s001abcdefgijh → s01abcd</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>s0000001abcdef → s001abcd</td>
<td></td>
</tr>
<tr>
<td>s00000000abcdef → s000abcd</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Output of G.711 codec from signed linear audio input, prior to inversion of every other bit
the sign bit, the codec adds “111” to the output. If “01” immediately follows the sign bit, the codec adds “110” to the output, signifying a lesser magnitude than if “1” immediately followed the sign bit. This pattern continues until the case where at least seven consecutive “0” bits follow the sign bit, which would result in an output of “000” for logarithmic magnitude. The remaining 4 bits of output are set to the first four bits in the input that follow the bits used to determine logarithmic magnitude. In transit, systems typically invert every other bit – xor 01010101 (0x55) – this is not shown in the figure for simplicity.

3. RELATED WORK

Mazurczyk and Lubacz [9] suggested a method they dubbed Lost Audio paCKets steganography, or LACK. LACK functions by intercepting audio packets at a given interval, replacing the audio payload with data to be transmitted covertly, holding the packet for an interval long enough for a receiver to consider it lost, optionally adding additional jitter to avoid being caught by a simple statistical analysis, and finally sending the modified packet. The interval at which LACK intercepts packets should be based on which audio codec is in use and what rate of packet loss for that codec would result in unacceptable quality; that is to say, LACK is adjustable for any given codec. Optionally, this interval can also be adjusted based on the expected call duration (which can be estimated based on statistics and refined as the call progresses) and amount of data needed to be sent. Mazurczyk [10] thereafter tested the method, showing that G.711 appeared best suited for use with LACK, both due to it having the highest bitrate amongst the codecs tested and it encountering the least degradation of quality as the ratio of lost to total packets increased. There is a concern that a mechanism should be in place to ensure packets arriving late on purpose and packets arriving late due to other reasons beyond our control can be differentiated; the jitter introduced in the interval at which packets will be delayed and modified could be produced by a pseudorandom number generator (PRNG) with a seed known to both parties. There is also a concern that a mechanism needs to exist for recovering lost steganographic packets. Additionally, there is an expectation that there is a single static message of known length, which is used during the optional step of regulating the interception interval based on expected call duration and message length.

Hamdaqa and Tahvildari [11] suggest ReLACK, which operates in very similar fashion to LACK, but uses a modified version of Shamir’s Secret Sharing Scheme [12] on the message being transmitted, increasing the message size to the degree necessary for a specified fault-tolerance, which could be the amount of data that can be comfortably transmitted with LACK without unacceptable quality loss. With this scheme applied, if the receiver obtains at least as much data as was in the original message, the original message can be reconstructed, which addresses the possible issue of lost data. However, if the receiver doesn’t obtain at least as much data as was in the original message, the entire message is lost.

Other methods in the literature attempt to subtly manipulate audio data as opposed to manipulating the flow of that audio data across the network and outright replacing the audio. One of the simplest approaches to steganography for many mediums is to tamper with the least significant bit (LSB) of each unit of data [13]. The sender simply replaces the LSB of a unit with the desired hidden bit to send, and the receiver reads the sent LSB. When using the LSB method, you expect that the modification of the least significant portion of the data will be negligible and go unnoticed by any observing parties.

An approach by Aoki [4] works in similar fashion to LSB in a special case. G.711 can transmit both a +0 and -0 signal, and Aoki’s [4] algorithm takes advantage of this by using the sign bit as least significant when the magnitude is 0. This technique is virtually lossless in terms of audio quality, but quickly becomes ineffective in areas with moderate background noise. Additionally, Aoki [4] created a semi-lossless method, which works by increasing the absolute magnitude of all non-zero samples by a variable amount, j, allowing zero-magnitude samples to have both their sign bit and true LSBs manipulated for storing hidden data.

The approach by Ito [3] uses a lower bitrate codec, G.726, in conjunction with G.711. The algorithm tests how many least significant bits of output from G.711 can be freely manipulated before a transcoding to the lower bitrate codec begins producing different results. This assures the quality of the G.711 samples will at least match that of the lower bitrate codec. As both the sender and receiver for a given audio stream can use the same method to determine how many bits can be tampered with freely before unacceptable degradation would result, both the sender and receiver would come to the same conclusion and retrieve the correct tampered bits.

Miao and Huang [5] created an approach that isn’t related to LSB-tampering. For each group of N samples, the approach treats each G.711 sample as a sign bit followed by a seven bit magnitude integer, and obtains the average for the N samples. It then, for all but one sample, determines the difference between that sample and the average, and will plan to embed a number of hidden bits equal to the log of the absolute value of the difference in that sample, rounded down. The sample will then be modified to equal the average plus an altered difference that will contain the hidden bits. The altered difference will be the sum of two numbers: some integer with a bit length equal to the number of bits to hide, plus two raised to the number of bits to hide. Finally, the one sample that was originally excluded will be manipulated to restore the average signed magnitude of the group back to the average originally calculated before anything was modified. This allows for a receiver to properly determine the number of bits that would be hidden in each sample, and subtract the correct average to recover the hidden integer in each sample.

4. EVALUATION

4.1 Overview of testing environment

We collected several audio recordings for testing. These audio recordings were grouped into three categories: no voice with low background noise, high volume voice with low background noise and high volume voice with moderate background noise. The no voice with low background noise category had two recordings. The first recording was a muffled thunderstorm recorded in a sealed building, with volume
comparable to light background noise. The second was computer-generated silence. The high volume voice with low background noise category included automated voice mail and operator prompts (VM prompts), and English as a second language study material (ESL). Finally, the high volume voice with moderate background noise category included our peers speaking a short story (Peers) and recordings of telephone calls considered of historical significance and publically available (Historic).

To test the algorithms, we used three performance metrics:

1. **Throughput**: which is the number of bits of the secret data embedded over time
2. **Noise-to-Signal ratio**: which is calculated according to Equation 1.

\[
\sum_{i=1}^{n} \left\{ \left( \frac{A_N}{S} \right)^2 \right\}
\]

(1)

where,

- \(S\) denotes the Signal
- \(N\) denotes the Noise
- \(A_N\) is the original linear amplitude of a given sound sample (G.711 has 8,000 samples per second)
- \(A_N\) is the maximum difference between the amplitude of the original sample and the sample after embedding the data on a linear scale
- \(n\) is the number of samples

3. **Perceptual Evaluation of Speech Quality (PESQ)**. PESQ algorithm [14] compares unmodified and degraded audio in a way that aims to report how degraded a human would perceive the audio to be. Higher scores are better, with scores of at least 4 considered good and scores of at least 3 considered acceptable but with noticeable degradation.

A higher throughput, a higher PESQ and a lower Noise-to-Signal ratio means better performance.

We first studied the performance of the algorithms in the worst case scenario, where embedding data would result in the greatest possible noise-to-signal ratio.

For Aoki’s [4] approach, we evaluate it in lossless (LL) and semi-lossless (SLL) modes. In the former, only the sign bit is manipulated when the magnitude is 0; in the latter, the sign bit and the least significant bit of the magnitude is manipulated i.e. the variable j in [4] is set to 1.

For the algorithm by Ito et al [3], G.726 could operate at four different bitrates, namely, 16kbits/s, 24kbits/s, 32kbits/s and 40kbits/s. Using the 32kbits/s mode offered more throughput than the 40kbits/s mode without much additional noise. Using 24kbits/s and 16kbits/s compared to 32kbits/s again offered additional throughput, but with a substantial noise increase. We will show results for 32kbits/s and 24kbits/s here.

For the approach by Miao and Huang [5], we use values of 5 and 13 for N, and a value of 96 for the maximum lambda. 13 was the N value used by [5] when presenting results of their algorithm. Their paper suggested that lowering the value of N would increase hidden throughput at the cost of noise; we chose an N of 5 to see this effect. The maximum lambda serves as a mechanism to prevent overflow – should any embedding operation have the potential to cause a sample to vary from the average magnitude more than the maximum lambda, no bits will be embedded in that sample. Should the maximum lambda be exceeded when adjusting the sample used to restore the average, all embedding operations for the entire group will be canceled, and the audio for the entire group will be unmodified.

### 4.2 Performance analysis

Figures 2, 3 and 4 show the average throughput of secret data for no voice with low background noise recordings, the high volume voice with low background noise and the high volume voice with moderate background noise categories, respectively. The Figures assume a no packet loss scenario. Aoki’s [4] algorithms perform well under no voice and low background noise conditions but poorer under high volume voice conditions. The algorithms by Ito et al [3] and Miao and Huang [5] perform well under high volume voice conditions, and poorly with generated silence. For Miao and Huang’s [5] algorithm, a group of five samples (that is, N=5) provided better performance than N=13 for all recordings with the exception of Thunderstorm; for the Thunderstorm recording, N=13 provided much better performance compared to N=5.

In Figure 5, we show average Noise-to-Signal ratios on modified audio files. The method by Miao and Huang [5] generated the most noise in every instance, with N=13 generating more noise than N=5. As N=5 also provides better throughput in all but one case as well as less noise in all cases, the suggestion by Miao and Huang [5] in their paper to keep N small appears valid. In both recordings from the high volume voice and moderate background noise category (peers and historic), the algorithm by Ito et al [3] generates more noise than Aoki’s [4], but, as shown in Figure 4, Aoki’s generates substantially less throughput. In the high volume voice and low background noise category (VM prompts and ESL), Ito et al [3]’s algorithm generated more noise than Aoki’s for VM recording and less noise in the ESL recording, but for both recordings Aoki’s [4] algorithm generated less throughput (see Figure 3). In the no voice with low background noise category, Aoki’s [4] algorithm generated more noise than Ito’s, but generated substantially more throughput (see Figure 2). Ito et al’s algorithm [3] generated neither noise nor throughput for the silence recording, hence not having a data point in Figure 5.

In Figure 6, we show PESQ results. Aoki’s [4] algorithms score at least 3.0 in all cases and above 4.0 in recordings from the high voice volume with moderate background noise category, suggesting acceptable to good audio quality. Ito’s algorithm consistently scores above 4, suggesting consistent good quality. Miao and Huang [5]’s method produced modified audio that scored less than 3.0 for multiple (though not all) recordings, suggesting poor to adequate quality.
Figure 2. Average throughput of secret data for no voice with low background noise, assuming no packet loss.

Figure 3. Average throughput of secret data for high volume voice with low background noise, assuming no packet loss.

Figure 4. Average throughput of secret data for high volume voice and moderate background noise, assuming no packet loss.

Figure 5. Average Noise-to-Signal Ratio of modified samples.

Figure 6. PESQ score of modified samples.
4.3 Steganalysis

Miao and Huang [5] showed that a commonly used steganalysis technique would detect LSB-based algorithms, but would not detect their own approach. Since Aoki’s and Ito’s algorithms are LSB-based algorithms, this means that they are detectable. In this section we illustrate that Miao and Huang’s algorithm can also be easily detected. As we explained earlier, the algorithm operates on groups of N samples. It interprets each G.711 sample as a sign-and-magnitude number, allowing for values of -0 to -127 and +0 to +127. What follows is an example of embedding data into a group of N=5 samples. Consider that the original sample group is as follows:

\[ x = \{+15, -5, -24, -7, +12\} \]

The first step in this algorithm, shown in Equation 2, determines the floor of the average of a group of N samples; \( \mu \).

\[ \mu = \left\lfloor \frac{\sum_{i=0}^{N-1} x_i}{N} \right\rfloor = \left\lfloor \frac{-9}{5} \right\rfloor = -2 \]  

Next, the algorithm determines the distance of all but the \([N/2]^{th}\) sample from the average as shown in Equation 3.

\[ d_i = \mu - x_i, i \neq [N/2] \]  

So for our example:

\[ d = \{-17, +3, +5, -14\} \]

The encoding step manipulates these differences to encode a bit pattern in each. First, the number of bits that can be encoded in all but one \([N/2]^{th}\) sample are determined. Miao and Huang [5] group differences based on sign and logarithmic magnitude, and each group have a bit count available for encoding. Equation 4 shows a formula for determining bit count per sample.

\[ b_i = \begin{cases} 0, & i \neq [N/2], |d_i| < 2 \\ \lceil \log_2 |d_i| \rceil, & i \neq [N/2], 2 \leq |d_i| < 16 \\ 4, & i \neq [N/2], |d_i| \geq 16 \end{cases} \]

According to Equation 4, the bit counts for our example are equal to:

\[ b = \{4, 1, 2, 3\} \]

Assume that the encoded bits and their decimal equivalent for our example are as shown below.

\[ m = \{0101_2, 0_2, 10_2, 101_2\} \]

\[ m = \{5, 0, 2, 5\} \]

The new differences are then calculated. Differences are to remain in the same group as they were in before – that is, they should have the same sign and (rounded down) logarithmic magnitude. Hidden data is encoded by manipulating the linear magnitude while maintaining the same sign and (rounded down) logarithmic magnitude, as shown in Equation 5.

\[ d'_i = (\alpha |\log_2 |d_i| + m_i) \cdot (|d_i|/d_i), i \neq \left\lfloor \frac{N}{2} \right\rfloor, |d_i| \geq 2 \]  

So this means that for our example,

\[ d' = \{-21, +2, +6, -13\} \]

The manipulated samples are then calculated. For all but the one sample that has been ignored thus far, the average with the updated difference from the average for that sample is used. The remaining sample will be used to restore the average, to allow the algorithm to successfully decode the hidden data. This is shown in Equation 6.

\[ x'_i = \begin{cases} \mu - d'_i, i \neq \left\lfloor \frac{N}{2} \right\rfloor \\ \mu + \sum_{i=0, i \neq \left\lfloor \frac{N}{2} \right\rfloor}^{N-1} d'_i, i = \left\lfloor \frac{N}{2} \right\rfloor \end{cases} \]

For our example,

\[ x' = \{+19, -4, -28, -8, +11\} \]

Here, we show that the average will be successfully restored for any values as shown in Equation 7.

\[ \mu' = \left\lfloor \frac{\sum_{i=0}^{N-1} x'_i}{N} \right\rfloor = \left\lfloor \frac{\sum_{i=0}^{N-1} (\mu - d'_i) + (\mu + \sum_{i=0, i \neq \left\lfloor \frac{N}{2} \right\rfloor}^{N-1} d'_i) + \sum_{i=0, i \neq \left\lfloor \frac{N}{2} \right\rfloor}^{N-1} (\mu - d'_i)}{N} \right\rfloor = \left\lfloor \frac{\sum_{i=0}^{N-1} \mu}{N} \right\rfloor = \left\lfloor \frac{N \mu}{N} \right\rfloor = |\mu| = \mu \]  

Since the floor of the average is restored to the original value, a receiving client will recover the same modified difference values, and will be able to extract the hidden bits.

Consider, now, the true average of a Miao-Huang encoded group, rather than the floor of the average. The math would look the same as in Equation 7, but without the floor symbols. Because \( \mu \) is guaranteed to be an integer because of the floor function used in computing it, we were able to say the floor of \( \mu \) was equal to \( \mu \) itself. Since the entire equation for \( \mu' \) was encompassed by the floor function at all points in the calculation, and the same floor function was just deemed unnecessary for guaranteeing an integer result at the final step of the calculation, we can thus say the floor function wasn’t necessary throughout all of Equation 7 to guarantee \( \mu' \) was an integer – the average of the group’s samples would be an integer, either with the floor function or without. If the average of a set of numbers is an integer, the summation of the set \((x')\) must be evenly divisible by the count of the set \(N\).

From this observation, one could conclude that groups of N samples that had hidden data encoded with the algorithm by Miao and Huang [5] would be more likely to have a sum evenly divisible by N than groups lacking hidden data via this algorithm. Equation 8 shows that the example group of N=5
samples we created for demonstration purposes does indeed have a sum that is evenly divisible by 5 after embedding:

$$\sum_{i=0}^{N-1} x_i' = -10 \quad (8)$$

Thus, we devised an algorithm for detecting embedded data with Miao-Huang that works as follows: break audio data into groups with N length, and test if the summation of samples in each group is evenly divisible by N. Since an adversary wouldn’t know which N is being used, they can test for multiple N values and see which if any looks promising. For audio unmodified by Miao and Huang’s [5] algorithm, after the summation of a group, we might expect a random probability for each possible remainder when dividing by N, like in Equation 9.

$$p(\text{remainder } = i) = \frac{1}{N}, \quad i = 0, 1, \ldots, N - 1 \quad (9)$$

Figure 7 shows that, for unmodified audio recordings, as the assumed N increases, the percentage of groups being evenly divisible (having a remainder of 0) tends to not increase, and tends to closely follow our expected probability in Equation 9. Figures 8 and 9 show this same test on audio modified with Miao and Huang’s [5] algorithm with N=5 and N=13, respectively. These graphs are noticeably skewed, especially when testing the correct N value or a multiple of the correct N value. E.g. for the silence recording in Figure 8 at N = 5, 10, 15, 20, 25, 30, 35 and at N= 13, 39 in Figure 9.

While one might expect the correct N to be at 100% for all audio recordings in Figures 8 and 9, it is important to remember that Miao and Huang’s [5] algorithm attempts to prevent overflows and can cancel an embed operation on a group as a result. In these two figures, any groups that did match on the correct N did contain hidden data, while all others were skipped to prevent overflow. For our example, the overflow checking steps were skipped for simplicity, and would not have affected the results unless the maximum lambda variable was less than 26.

Tests for the skewing shown in the figures would allow easy detection of data being encoded with the algorithm by...
Miao and Huang [5]. Combining this new information with existing research, we conclude that all the algorithms considered in this paper are detectable.

5. CONCLUSION

In this paper, we compared various approaches to hiding data in a VoIP stream. Our experiments showed that:

- The method by Aoki [4] performs well under little-to-no voice volume conditions
- The methods by Ito et al [3], Miao and Huang [5] perform well under high volume conditions, while Aoki’s [4] performs unsuitably
- The method by Miao and Huang [5] tends to perform better with a lower N value
- The method by Miao and Huang [5] can generate excessive noise and may be easily detectable when considering the algorithm design

REFERENCES