Steganography in Audio

Neil Jenkins[1], **Jean Everson Martina**[1*†]

[1] Computer Laboratory
University of Cambridge
15 JJ Thomson Avenue
Cambridge - UK
CB3 0FD

nmj27@cam.ac.uk, Jean.Martina@cl.cam.ac.uk

Abstract. *A program was built to hide an arbitrary file inside audio using three different methods based on current research in the field. Standard libraries were used to compress (with bzip2) and encrypt (using AES) the data before it is embedded. The error-correcting Hamming and Golay codes were implemented to increase the system's resistance to malicious modification. Finally,the LAME MP3 encoder was integrated to work with one of the methods (echo hiding). The finished project provides a means for secure, discreet communication through a monitored, hostile channel.*

1. Introduction

Throughout history, people have needed to communicate covertly. There are many circumstances when concealing the *meaning* of a message through classic cryptography is not enough—the very *existence* of the correspondence must be made undetectable. Even in the UK, under the RIP Act 2000 [Regulation of Investigatory Powers Act 2000], the refusal to supply decryption keys to the police may result in up to five years in jail. Steganography, the art of concealed communication, has been used since antiquity; many ingenious methods have been devised over the last few millennia to hide the presence of messages [Petitcolas et al. 1999], from invisible ink between lines of an otherwise innocent letter to micro-dots containing whole images in miniature. It is part of the broader field of information hiding, which encompasses areas such as watermarking and fingerprinting. This area has begun to gather more research attention from the academic community over the last ten years, driven by a growing interest in embedding copyright notices and watermarks into digital media.

The majority of research into computer-based steganography has used images as the carrier medium. Although the precision of the human auditory system makes it harder to embed secretly into, audio has a number of advantages: the file sizes tend to be larger, potentially giving a higher embedding capacity - to hide more data per amount of medium data -, and more importantly, they are perfect as an innocuous file to exchange—MP3s are routinely emailed, carried around on iPods, or even burnt onto CD and mailed through the post.

Despite the merits of audio as a steganographic medium, there are few existing applications that make use of it. With one exception, all of the programs found only work

[*] Project Supervisor
[†] Supported by CAPES Foundation/Brazil on grant #4226-05-4

with uncompressed audio using the simplest least-significant-bit method of embedding (see §3 for an explanation of this technique). The sole exception was MP3Stego[1], which uses a form of perturbed quantisation to hide data inside the parity of an MP3 file's block lengths. This is a powerful technique, where the embedding process is built into the MP3 encoder at the stage where quantisation is performed on the uncompressed audio. The embedding program can use knowledge of the original signal to change the quantisation value for the block that will be least affected, thus making it harder to detect. This method is not robust to any modifications to the audio file but is fairly secure, although under certain circumstances it can be detected by statistical analysis of the block lengths [Westfeld 2003].

2. Challenges

Hiding data imperceptibly in audio is a difficult task, as the human auditory system (HAS) is acutely sensitive; it operates over a range of power in excess of one billion to one and a range of frequencies of more than a thousand to one. It is also sensitive to additive random noise, capable of detecting such disturbances at volumes as low as 80dB below ambient level. There are still properties that can be exploited though, because while the HAS has a great dynamic range, its differential range is quite small, allowing loud noises to mask quieter ones. It is incapable of perceiving absolute phase, distinguishing only relative phase, and automatically ignores common environmental distortions [Bender et al. 1996].

Despite the difficulties involved, several methods have been proposed that hide data inside generic audio. There are few books on the subject, but several research papers. The best methods are discussed in §3. Of course, weaknesses in the HAS are also utilised by lossy compression algorithms such as MPEG-1 Audio Layer 3 (MP3) to eliminate data entirely; the task of finding a suitable method not stripped out by such manipulations is even harder.

We opted to implement the proposed system using symmetric instead of asymmetric cryptography because what we look for in cryptography is an extra layer of security and to help on adding randomness in data to be hidden with our methods. Working over cryptography related problems, such as key distribution and strength of cryptographic primitives is beyond our intended scope.

3. Embedding Techniques

There are several different ways in which data can be embedded inside an audio file. Here we describe the three most significant algorithms: least significant bit substitution, echo hiding and low-frequency amplitude modification.

The simplest solution is to embed the data into the least significant bit (lsb) of each frame of audio. The distortion caused by this embedding is minimal; a low intensity noise is introduced, weak enough to make it imperceptible in practically any audio recording. However, this basic system does not make the data undetectable; an attacker may simply read every lsb and see if a readable file results. The disadvantage to the use of lsbs for data embedding is robustness. An active warden can simply zero the lsb of every frame in an audio file to remove any possible embedded data, regardless of whether she is able to detect it, and any form of lossy compression destroys the data. However, in comparison

[1] http://petitcolas.net/fabien/steganography/mp3stego/

with other techniques it has a far higher embedding capacity [Bender et al. 1996], so it was deemed worth implementing.

Echo hiding was first proposed by Gruhl et al. in 1996 [Gruhl et al. 1996]. Since then, there have been numerous papers proposing improvements over the original scheme, e.g. [Oh et al. 2001, Kim and Choi 2003]. It takes a different approach to ensuring minimal degradation of the cover audio. Rather than introducing low-power, hopefully-imperceptible noise, it introduces tiny echoes that are similar to the resonance from walls, furniture, etc. and as such are routinely ignored by the brain. Echo hiding was selected for implementation as evaluations (e.g. [Oh et al. 2001]) found it to be highly resistant to MP3 compression - Not destroyed by the MP3 compression scheme. Also, the original paper [Gruhl et al. 1996] cited a bit rate of at least 16 bps, higher than other similarly robust methods.

This method, proposed by Lie and Chang [Lie and Chang 2006], seeks to preserve the time-domain waveform envelope as much as possible by exploiting the relations between consecutive segments of audio and examining them as related entities rather than separate samples. Their results showed good resistance to MP3 compression and a data rate similar to that obtained through echo hiding, making it another good candidate for implementation.

4. Error-Correcting Codes

With an active warden, it is important for the methods to be as robust as possible to transformations such as lossy encoding and band-pass filtering. Whilst the embedding method is the key factor here, not all distortion of the data can be prevented, so it is necessary to use error-correcting codes to aid recovery of the original data.

Hamming codes are perfect single-error-correcting codes; they have a minimum *distance* (number of places where two code words differ) of exactly 3, thus any one-bit error can be corrected, as it will result in a pattern closer to the original code word than any other [Lin and Costello 1983]. For any positive integer $m \geq 3$ there exists a Hamming code with code length $n = 2^m - 1$, encoding $k = 2^m - m - 1$ bits of data. The generating matrix is given as $\mathbf{G} = [\mathbf{Q}^T \ \mathbf{I}_{2^m-m-1}]$, where $\mathbf{Q}$ is the sub-matrix consisting of $2^m - m - 1$ columns, each of which is a unique m-tuple of weight 2 or more (the order is unimportant) and $\mathbf{I_{2^m-m-1}}$ is the $(2^m - m - 1) \times (2^m - m - 1)$ identity matrix. The parity check matrix is given by $\mathbf{H} = [\mathbf{I}_m \ \mathbf{Q}]$. The syndrome gives the position of any one-bit error for decoding.

Cyclic or *polynomial-generated* codes are an important subclass of linear block codes with the property that any cyclic shifting of a code word is also a code word. A polynomial of degree $(n - k)$ known as the *generator polynomial* defines such a code; parity bits are generated by dividing the k-bits of information by the generator polynomial using polynomial arithmetic [Lin and Costello 1983, p. 89]. The cyclic Golay codes are the only other non-trivial perfect codes known to exist besides the Hamming codes [Huffman and Pless 2003, p. 49]. The (23, 12) binary Golay code has a minimum distance of 7, thus any error of up to 3-bits may be corrected within a 23-bit code word. The code is generated by either:

$$\mathbf{g}_1 = 1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11} \tag{1}$$

or

$$\mathbf{g}_2 = 1 + X + X^5 + X^6 + X^7 + X^9 + X^{11} \quad (2)$$

Decoding Golay codes is rather more complicated. For software implementation, the Systematic Search Decoder [Lin and Costello 1983, p. 128] is reasonably efficient.

5. Program Structure

There are two main sections within the program: the methods that deal with reading in audio and embedding or extracting data, and the data pipeline, concerned with opening files and encrypting, compressing and adding error-correcting codes as required. It was important to specify the interface between these two areas early in the development process so they could be written independently; this is the role of the `BitStream` abstract class, described in §6.

6. Bit Streams

The `BitStream` class abstracts away the details of the embedded data from the embedding methods. The class itself is abstract, with three methods declared pure virtual. Stego-algorithm implementations only interact with the `BitStream` interface; they never have to know anything about the concrete implementations, giving loose coupling between these two areas of code.

The `FileBitStream` class exists to allow reading and writing of files through the `BitStream` interface. It was originally written using the `iostream` C++ library, but when it came to integrating the compression feature, it was found easier to make this part of the same class, just adding an optional 'use compression' argument to the constructor. The bzip2 [2] library was an obvious choice, as it is fast, freely available and offers an excellent compression rate.

There are several free high-quality encryption libraries available in C/C++. OpenSSL[3] and Crypto++[4] were considered before settling on Botan [5], partly as (unlike OpenSSL) it is written in C++ so was easier to integrate, but more importantly because it has better documentation. The Botan library contains a useful random-number generating class, which automatically seeds itself with entropy gathered from several places in the system. To make the task of getting and setting the key easier a wrapper class, `Cipher`, was created that provides access to the required cryptographic functions and includes extra key-management methods. It also provides a centralised place in which the choice of cipher is made. If AES were ever found to be broken and another block cipher were more suitable, only the single file need be changed for the whole program to switch ciphers.

The `ECBitStream` class is another example of the decorator pattern [Gamma et al. 1995, pp. 175–184], this time adding error-correcting redundant bits to the stream. Early in the implementation phase it was discovered some methods often introduced errors, so first the (7, 4) Hamming code was implemented. Further analysis of the errors, however, found they were more likely to occur in small bursts, thus the Golay code seemed a better choice. When re-factoring the class the *strategy* design pattern [Gamma et al. 1995, pp. 315–323] was used to separate the error-correcting encode/decode functions entirely, allowing the block code to be replaced with ease.

[2] http://www.bzip.org/
[3] http://www.openssl.org/
[4] http://www.cryptopp.com/
[5] http://botan.randombit.net/

7. StegAudioFile

Representing the stego-audio file, this abstract class encapsulates the libsndfile library[6] calls required to manipulate an uncompressed audio file, and provides external clients a simple interface for embedding and extracting data. Some code is common to all the embedding algorithms, such as setting up the `BitStream` and creating an output audio file when embedding.

Implementing the most basic form of least significant bit substitution is fairly trivial; reading the frames of audio into a buffer, masking off the lsb of each frame and adding the next bit in the data stream would suffice. In our implementation, however, we make use of the Botan cryptographic functions to embed data into the parity of the lsbs of a pseudo-random number of frames, between 32 and 63; the number is generated using the AES algorithm in output feedback mode, making use of the `Cipher` class. The extraction function requires the stego-key to recreate the same sequence and recover the data. If the parity of the section does not correspond to the parity of the bit to be embedded, the least significant bit of a random frame is flipped.

Embedding using echo hiding requires implementation of a FIR filter. To add both echoes and pre-echoes, an audio input buffer is required to hold enough frames either side of the section of audio currently being processed. Each section consists of a transition window used for smoothly mixing between echo delays and the segment where a constant echo encodes a single bit; this is illustrated in Figure 1.

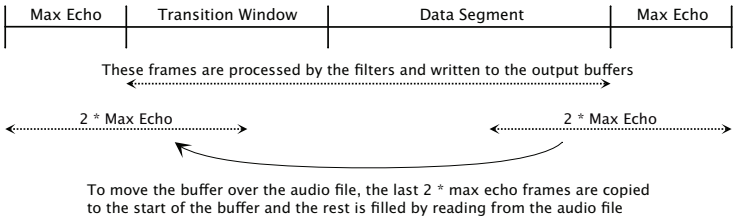


Figure 1. Input audio buffer for echo hiding

The first frames of audio up to the maximum echo delay cannot have an echo added, so are written to the output without modification. After this, the input buffer moves over the entire file; to look at the next section (transition window plus data segment), we copy over the last `2 * MAX_ECHO` frames to the beginning of the buffer then fill the rest by reading from the audio file. On each iteration both the 'one' and 'zero' echo kernels are applied, with the results written into the two output buffers.

There are a number of parameters that may be adjusted. The length of the transition window affects the unobtrusiveness of the steganography; if it is too short then the changes in resonance become noticeable. The segment length used to embed each bit of data impacts the recovery rate; longer segments are better. Echoes around the 2 ms mark work well in terms of not aversely affecting the sound quality and being detectable; we used the delays given by Kim and Choi [Kim and Choi 2003]. The final parameter is the decay rate: what proportion of the amplitude of the original signal is added back in the echo.

[6]http://www.mega-nerd.com/libsndfile/

To extract data, the cepstrum must be found for each segment of audio in which a single bit has been embedded. First, a Hamming window is applied and the Fourier transform taken. Next, we find the complex logarithm of each value using the `cmath` and `complex` libraries then perform the inverse Fourier transform, resulting in the cepstrum. We extract the bit by determining whether the magnitude of the cepstrum is greater at the delays used for hiding a '0' or for those used embedding a '1'.

The implementation of amplitude-modification steganography is essentially a direct translation of the algorithm into code. The most interesting part of the code is the sorting of the mean amplitudes. As there are only ever three members in the array holding them, the use of a separate sorting function would have been unnecessary overhead; only three comparisons are required. As well as sorting the means though, the algorithm requires one to keep track of which section corresponds to which mean, so that the amplitudes can be scaled later.

8. MP3 Encoding and Decoding

For MP3 compression the open-source LAME[7] library is both fast and high-fidelity. Since we only need the most basic of features, we abstract all the LAME specifics of buffering the audio frames and setting the compression options away into a single function. As an MDCT[8] based encoder, MP3 files introduce padding to the beginning and end of each file. All of the embedding methods rely on the start frame used for embedding remaining in the same place, so to compensate for this, when embedding or extracting from an MP3 we skip the appropriate number of frames at the beginning of the file before passing it to the extraction functions.

9. Resistance to Detection

To determine the impact of the three embedding methods on audio quality a controlled listening test was conducted following the 'double-blind triple-stimulus with hidden reference' method laid down in the ITU BS.1116 standard [Union 1997].

All trials were conducted in a quiet, studio environment using high-quality reference monitor headphones We used six tracks selected to represent a wide range of audio types. Fifteen student volunteers were recruited to take the test. We present the mean difference score (score assigned to stego-audio minus score assigned to original) and the sample standard deviation for each track and embedding method in Table 1.

	LSB		**Echo**		**Amplitude**	
Track	$\overline{x}$	s	$\overline{x}$	s	$\overline{x}$	s
Classical	-0.25	0.89	-0.64	0.94-0.83		1.29
Folk	0.15	0.54	-0.23	1.48-1.14		0.96
Hip Hop	-0.05	0.34	-0.43	0.89-3.39		1.00
Pop	0.02	0.45	-0.78	1.10-1.11		1.03
Rock	-0.04	1.22	-0.92	1.00-2.59		1.39
Spoken	-0.08	0.86	-0.13	0.64-0.85		1.03

Table 1. Mean difference and sample standard deviation

The hypothesis that the stego-audio is imperceptibly different from the original is proven by a mean difference score of 0.0.

[7]http://lame.sourceforge.net/

[8]Modified discrete cosine transform.

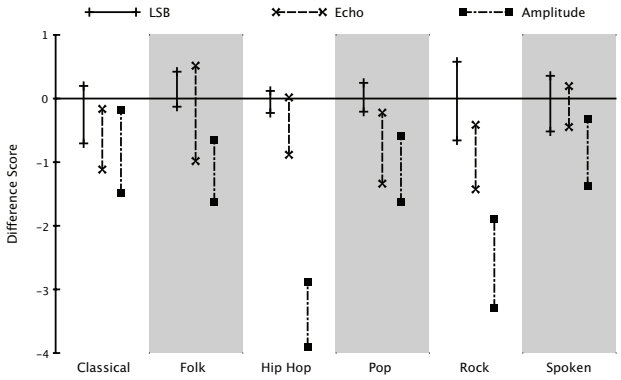


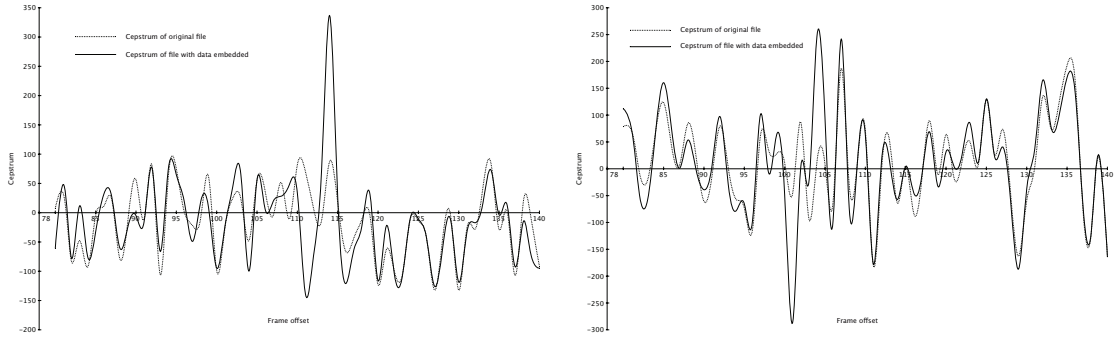
Figure 2. 95% confidence intervals for the mean difference scores

Looking at the 95% confidence intervals[9] for the means in Figure 2, it is clear that the lsb substitution method makes an imperceptible change to the audio. The mean difference scores are all very close to 0.0 and the low standard deviations make the confidence intervals small.

Echo hiding also fares very well, although it cannot be said to be imperceptible on all tracks. The confidence intervals for the folk, spoken and hip hop tracks all contain 0.0, and the one for the spoken word track is particularly narrow, due to a low variance. The other tracks generally fell in the range down to -1.0, meaning the change was perceptible but not annoying.

Amplitude hiding is the weakest method. The intervals in general are around the -0.5 to -1.5 range, making the change probably just perceptible enough for a warden to detect. The rock and especially the hip hop track were much worse; this was due to *clipping* introduced by the amplitude changes. Music in these genres is often highly compressed, meaning the dynamic range is reduced and the music is pushed to the peak of the uncompressed format's decibel range for the majority of the track.

9.1. Non-Audible Factors



(a) A segment with the echo clearly visible in the cepstrum

(b) A segment with the echo peaks similar to surrounding peaks

Figure 3. Non-Audible Factors

[9]Calculated using the normal distribution; the central limit theorem states that the distribution of the sample means should tend to this.

To be completely undetectable a stego-audio file must resist more than just human listening tests. Least significant bit substitution is probably the least detectable non-audibly as well as audibly. As explained in §3, by embedding into the parity of a group of least significant bits, the impact of this method on the statistics of the audio is minimised, probably below any reasonable hope of detection.

Whilst hard to audibly detect, echo hiding is a lot easier to distinguish using other methods. The issue is that the peak in the cepstrum is often quite visible. For example see Figure 3(a), which graphs the significant portion of the cepstrum for a random segment of the pop song before and after embedding data; the large peak here is obvious.

However, not every segment is so obvious. In Figure 3(b) a segment from the same song is shown where peaks from echoes are clearly visible, but not much greater than nearby peaks. An attacker examining a number of segments would probably find enough segments with significant spikes to conclude that the audio contains hidden data.

We think that the only way to defend against this attack would be to calculate the power of the echo for each segment individually based on the cepstrum of the original audio file. The peak could then be made sufficiently higher than the peak at the other delays to ensure that the correct bit is decoded, but otherwise as small as possible so that it does not stand out in the cepstrum. Erfani et al. [Erfani et al. 2007] have outlined a possible algorithm.

The amplitude-modification algorithm is promising, as detecting subtle shifts in the mean amplitude of audio is a potentially difficult task; music tends to have quite a high dynamic range. However, the clipping would need to be eliminated and more work done to reduce the general impact first.

10. Robustness

To evaluate robustness we created a new `BitStream` implementation to supply a simple alternating '101010...' pattern 4096 bits long. We then transformed the clips in a variety of ways before attempting to extract the data. The mean percentage errors are shown in Table 2.

	LSB		**Echo**		**Amplitude**	
Transformation	$\overline{x}$	s	$\overline{x}$	s	$\overline{x}$	s
Original	0.00	0.00	1.44	1.171.37	1.92	
Resampling	50.0	0.686	1.88	1.6416.3	6.10	
Bandpass Filter	45.9	5.30	0.838	1.1343.8	3.11	
Conversion to Mono	49.9	0.330	7.45	3.8350.4	0.629	
AAC 96 kbps	45.7	5.19	1.39	1.126.10	4.78	
MP3 160 kbps	50.2	0.556	1.72	1.6429.3	2.29	

Table 2. Extraction: mean percentage error and sample standard deviation

Whilst allowing for perfect recovery when no attack has been made, LSB substitution is clearly extremely fragile. The 95% confidence intervals for all transformations other than AAC conversion contained 50%, the expected mean value if one were to just generate random bits.

Echo hiding is the most robust, being almost unaffected by everything except conversion to monophonic sound, where it still stayed within the recovery domain of the

error-correction codes. In fact, the lossy compression in AAC marginally increased the recovery rate.

Amplitude modification fares well in the face of resampling or compression, but the data is destroyed by conversion to mono. It is interesting to note that the resistance to AAC compression is much higher than to MP3, despite the much lower bit rate used for the former. AAC is a newer, higher quality codec, and appears to do a better job of maintaining the exact dynamics of the original.

Synchronisation is probably the biggest weakness of all the methods. An extra frame or two in the audio will prevent recovery of all subsequent lsb data. Synchronisation codes inserted periodically into the bitstream would provide resistance against this form of attack at the expense of data rate. Interesting possibilities arise when more than one embedding method can be used simultaneously. LSB substitution is too fragile to embed on top of, but embedding again into existing echo or amplitude stego-audio may be possible. LSB substitution may be used after either of the other techniques. Amplitude modification may be done after echo hiding whilst maintaining a high recovery rate for both, but the reverse order of transformations results in damage to the carefully adjusted amplitudes, reducing the recovery rate.

11. Data Rate

The final consideration in any steganography algorithm is the data rate achievable. For least significant bit substitution, each bit is embedded into the parity of a random number of frames, between 32 and 63. The average is therefore 47.5 frames per bit. Since it embeds across all channels the mean bit rate for a 44,100 Hz stereo WAV file will be:

$$\frac{44100 \times 2}{47.5} = 1856.8 \text{ bps}$$

Echo hiding embeds the bits into all channels, taking 2048 frames for each one (a data segment of length 1792 frames and a transition window of 256 frames) thus giving a data rate of:

$$\frac{44100}{2048} = 21.5 \text{ bps}$$

The amplitude modification method embeds into each channel separately, using three 512-frame sections to embed each bit. This gives a data rate of :

$$\frac{44100 \times 2}{3 \times 512} = 57.4 \text{ bps}$$

Echo hiding obviously uses a lot less as a percentage of a raw WAV file, but as we have demonstrated, it is very robust in the face of lossy compression algorithms. If it is transmitted in 96 kbps AAC then one second takes only 96,000 bits of space, giving an embedding capacity of 0.0224%—not as good as LSB but only a factor of 6 out; a reasonable price for robustness.

The echo hiding method had difficulty with the Folk track, with a 3.45% error recorded extracting data from the unaltered stego-audio file. Amplitude showed an anomalously high error in the hip hop track, which was probably due to the clipping that also lowered the subjects' scores in the listening test. This should be corrected but if one file is not appropriate the sender may simply choose another.

12. Final Considerations

There are situations where *secure* communication is insufficient; it must also be *covert*. In this paper we have looked at the use of audio as a carrier for hidden data, successfully implementing three methods that overcome the precision of the human auditory system. The resulting application allows for effective data hiding within uncompressed and compressed audio.

Improvements could still be made to all three algorithms. Psycho-acoustic models, such as the open-source GPSYCHO, could be used to embed data into the areas of audio with the least chance of detection. The amplitude modification method is currently the weakest in terms of audibility, but shows a lot of potential. Some simple checks and volume reduction to eliminate clipping should make this fairly imperceptible and using cryptographically generated random variable segment lengths could help make statistical detection even harder.

References

Bender, W., Gruhl, D., Morimoto, N., and Lu, A. (1996). Techniques for data hiding. *IBM Systems Journal*, 35(3&4).

Erfani, Y., Moin, M. S., and Parviz, M. (2007). New methods for transparent and accurate echo hiding by using the original audio cepstral content. In *6th IEEE/ACIS International Conference on Computer and Information Science*.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Orientated Software*. Addison-Wesley.

Gruhl, D., Lu, A., and Bender, W. (1996). Echo hiding. In *Proceedings of the First International Workshop on Information Hiding*, pages 293–315. Springer.

Huffman, W. C. and Pless, V. (2003). *Fundamentals of Error-Correcting Codes*. Cambridge University Press.

Kim, H. J. and Choi, Y. H. (2003). A novel echo-hiding scheme with backward and forward kernels. *IEEE Trans on Circuits and Systems for Video Technology*, 13(8).

Lie, W.-N. and Chang, L.-C. (2006). Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modulation.

Lin, S. and Costello, D. J. (1983). *Error Control Coding: Fundamentals and Applications*. Prentice Hall.

Oh, H. O., Seok, J. W., Hong, J. W., and Youn, D. H. (2001). New echo embedding technique for robust and imperceptible watermarking. In *Proceedings of the IEEE ICASSP*, volume 3, pages 1341–1344.

Petitcolas, F. A., Anderson, R. J., and Kuhn, M. G. (1999). Information hiding—a survey. In *Proceedings of the IEEE*.

Regulation of Investigatory Powers Act 2000. Regulation of investigatory powers act 2000. `http://www.opsi.gov.uk/acts/acts2000/ukpga_20000023_en_1`. Part III, Section 49.

Union, I. T. (1997). Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems. Technical report, Recommendation ITU-R BS.1116-1.

Westfeld, A. (2003). Detecting low embedding rates. In Petitcolas, F. A., editor, *Information Hiding: 5th International Workshop*. Springer.