A Study on Low-Complexity Audio Encryption Methods for Digital Rights Management

Submitted to
Graduate School of Information Science and Technology
Osaka University

July 2015

Twe Ta Oo

Publication List

Journal Papers

1. Twe Ta Oo, T. Onoye, and K. Shin, "Partial encryption method that enhances MP3 security,'' in *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98-A, no. 8, Aug. 2015, to be published.

Conference Papers with Referee

1. Twe Ta Oo, T. Onoye, and K. Shin, "A partial encryption scheme for compressed audio based on amplitude scaling," in *Proceedings of International Workshop on Smart Info-Media Systems in Asia (SISA)*, pp. 73-77, Sept. 2013.
2. Twe Ta Oo and T. Onoye, "Progressive audio scrambling via wavelet transform," in *Proceedings of IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 97-100, Nov. 2014.
3. Twe Ta Oo and T. Onoye, "Progressive audio scrambling via complete binary tree's traversal and wavelet transform," in *Proceedings of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Dec. 2014.

Conference Papers without Referee

1. Twe Ta Oo, Takao Onoye, and Kilho Shin, "An approach to amplitude scaling partial encryption for compressed audio," in *IEICE Technical Report*, vol. 113, no. 135, pp. 239-244, July 2013.

Abstract

This thesis discusses low-complexity audio encryption methods for Digital Rights Management (DRM). Recent years have seen the rapid growth of Internet traffic and the proliferated distribution of digitalized audio products such as music, audio books, and spoken news. As a result, secure distribution and copyright protection of those products have been increasingly important. The DRM technologies have been intensively studied in this area, and encryption plays an important role to protect the contents from unauthorized accesses. Although encryption ensures content security, the naive method of encrypting an audio file would destroy compliance with the audio standard so the resulting encrypted file could not be rendered by existing standard media players. This thesis focuses on low-complexity audio encryption methods that keep compliance with the media standard and achieve the following DRM requirements: 1) providing the confidentiality in audio distribution, 2) controllably degrading the audio quality by adjusting the percentage of encryption, and 3) realizing the try-before-purchase model, which is one of the important business models of DRM, in which the encrypted audio files are published for commercial purpose; users can render those files for trial without decryption and enjoy the contents in original quality by purchasing the decryption keys.

Firstly, this thesis presents a low-complexity partial encryption method for compressed audio (MP3). Unlike conventional encryption which encrypts the whole file, partial encryption can provide some interesting features such as yielding low-quality signals, reducing execution time, and coexistence with the media standards. The main idea of partial encryption is to protect the entire content by encrypting only the perceptually important parts. This thesis discusses how to choose the perceptually important parts during the MP3 encoding process in accordance with the concept of the Human Auditory System (HAS). Experimental results show that encrypting the whole MP3 file renders the audio signal meaningless while encrypting 2-10% of the file degrades the audio quality but not completely destroys the signal so it can be used as trial music. That trial MP3 keeps compatibility with the standard so it can be rendered by any existing MP3 players without need to decrypt. Under the access of the correct decryption keys and specific MP3 players, the full-quality MP3 can be successfully recovered. In addition, this thesis discusses the invalid amplitude problem regarding audio encryption: in any kinds of audio format, there are valid amplitude ranges for audio samples, which differ based on the supported bit-depth of an audio coder. If the audio samples to be coded are not within the valid range, they are clipped. This becomes a problem when the encrypted audio samples are beyond the valid ranges and get clipped because the data losses introduced by clipping deter the decryption process from successfully recovering the signal. This thesis also presents a solution for this problem.

Secondly, this thesis presents two low-complexity audio scrambling methods, which are kinds but not direct applications of audio encryption. Unlike audio encryption that renders an audio signal meaningless by changing both the values and positions of the contents, audio scrambling degrades the residual intelligibility of an audio signal by breaking the coherence between data contents. They neither inject any new values nor change the values of the existing contents. Due to this feature, they are more preferable to usual audio encryption to be used as pre- and post-processing of data hiding methods. This thesis presents two effective audio scrambling methods in the time domain: one based on the pre-order traversal of a complete binary tree and the other on a pseudorandom number generation algorithm called Mersenne Twister (MT). Experimental results show that the proposed methods are very effective in terms of time and space complexity and scrambling effect. However, their cryptographic security is limited because of the only use of permutation operations.

Thirdly, with the aim of strengthening the cryptographic security of the proposed audio scrambling methods, this thesis presents two new schemes in the wavelet domain. First, an audio signal is wavelet decomposed. Then, the layers of wavelet coefficients are separately scrambled by considering not only the pre-order but also in-/post-order based scrambling methods in the first scheme and using the MT based scrambling method with a series of keys in the second scheme. Experimental results show that anyone without knowledge of the correct wavelet decomposition parameters and the correct method/key used for each layer will not be able to successfully descramble the signal. The new schemes also achieve progressive scrambling that enables the audio outputs with different quality levels to be generated by controlling the scrambling degree on the basis of the system requirement: slightly distorted ones for the try-before-purchase model of the DRM systems and severely distorted ones for the systems with strong security needs.

As a conclusion, this thesis presents low-complexity audio encryption methods for both compressed and uncompressed audios with detailed discussions on 1) how to solve the invalid amplitude problem regarding audio encryption, 2) how to effectively choose the perceptually important parts for partial encryption, 3) how to conduct encryption while keeping compliance with the media format, and 4) how to strengthen the cryptographic security of audio scrambling. In addition to providing confidential audio distribution, the proposed methods can also be used to realize the try-before-purchase model. Thus, these proposals strongly contribute to the development of efficient DRM systems.

Acknowledgements

First of all, I would like to express my deep and sincere gratitude to my supervisor, Professor Takao Onoye of Osaka University, for providing me a precious opportunity and an excellent environment to study as a doctoral student in his laboratory. His encouragement, detailed and constructive comments, and professional guidance have led me to this achievement.

I would like to express my heartfelt appreciation to Professor Yoshihiro Kilho Shin, University of Hyogo, for his dedication of a considerable amount of time from his busy schedule to guide and help me on my learning in the field of cryptography. His interesting discussions and excellent advices on cryptographic technologies presented in this thesis have greatly contributed to the success of this work.

I am deeply grateful to Professor Tatsuhiro Tsuchiya and Associate Professor Masanori Hashimoto of Osaka University for agreeing to be on my dissertation committee. They were more than generous with their expertise and precious time for reviewing my thesis.

I would like to take this opportunity to express my gratitude to all of the professors from Information Systems Engineering Department of Osaka University. I am deeply thankful to Professor Masaharu Imai, Professor Koji Nakamae, Professor Haruo Takemura, and Professor Makoto Nakamura for their useful advices.

I am also thankful to Assistant Professor Masahide Hatanaka of Osaka University for technical and other supports in the laboratory.

I am also thankful to all of my friends who kindly and actively helped me during the subjective audio tests conducted to evaluate this work. I will never forget their dedication and enthusiasm.

I also owe my gratitude to Ms. Mayuko Nakamura, Ms. Yuki Yoshida, and Ms. Tomomi Kondo for all the things they have done for me. They kindly and willingly helped me whenever I had difficulties in doing office procedures because of my language deficiency.

My appreciation also goes to all members of Information Systems Synthesis Laboratory of Osaka University, especially to Mr. Zhao Wen Jun, for their understandings and various supports throughout my student life.

I would like to dedicate this thesis to all of my teachers who mentored me throughout my career. They installed in me sources of knowledge and firm determination which led to this success.

Last but not least, I also dedicate this thesis to my parents and my uncles and aunts. I am deeply indebted to them for their love and continuous support. They never failed to encourage me whenever I had a hard time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter describes the background and objectives of this thesis. This thesis focuses on developing low-complexity audio encryption methods, for both uncompressed and compressed audio formats, to be applied in Digital Rights Management (DRM) environment. The two basic problems which make most of the usual encryption methods to be less preferable for audio protection are discussed. Then, the concept of partial audio encryption and its desirable features over usual encryption of the whole file are presented. The theories behind uncompressed and compressed audio file formats are also briefly discussed. Finally, the objectives and contributions of this thesis are explained.

1.1 Background

Nowadays, the multimedia technology has been widely used in various fields of industrial production, scientific research, and daily life [1-2]. The proliferated distribution of digitalized audio products has been seen in recent years due to the advents of multimedia compression standards and high-speed Internet and communication networks. These latest technologies have made our daily lives easier and more comfortable, but on the other hand they have been raising serious issues of copyright violation and unauthorized access of digital contents on a scale never before imagined. Many producers from music industries fear that online distribution of their works, on which the revenue of these industries partly or wholly depends, will speed up illegal distribution on the Internet or the Darknet [3]. For solving those issues, DRM [4-5] technologies have been intensively studied by researchers from both academia and industry.

The DRM is a set of access-control techniques used by manufacturers, publishers, and copyright holders to limit the usage of digital devices or information [6-7]. The most commonly used DRM technologies are:

- **Restrictive licensing agreements**: The access to digital materials, copyright and public domain is controlled. Some restrictive licenses are imposed on consumers as a condition of entering a website or when downloading software [8].
- **Encryption, scrambling of expressive material, and embedding of a tag**: This technology is designed to control access and reproduction of information.

Generally, a DRM system can be realized as shown in Fig. 1.1.

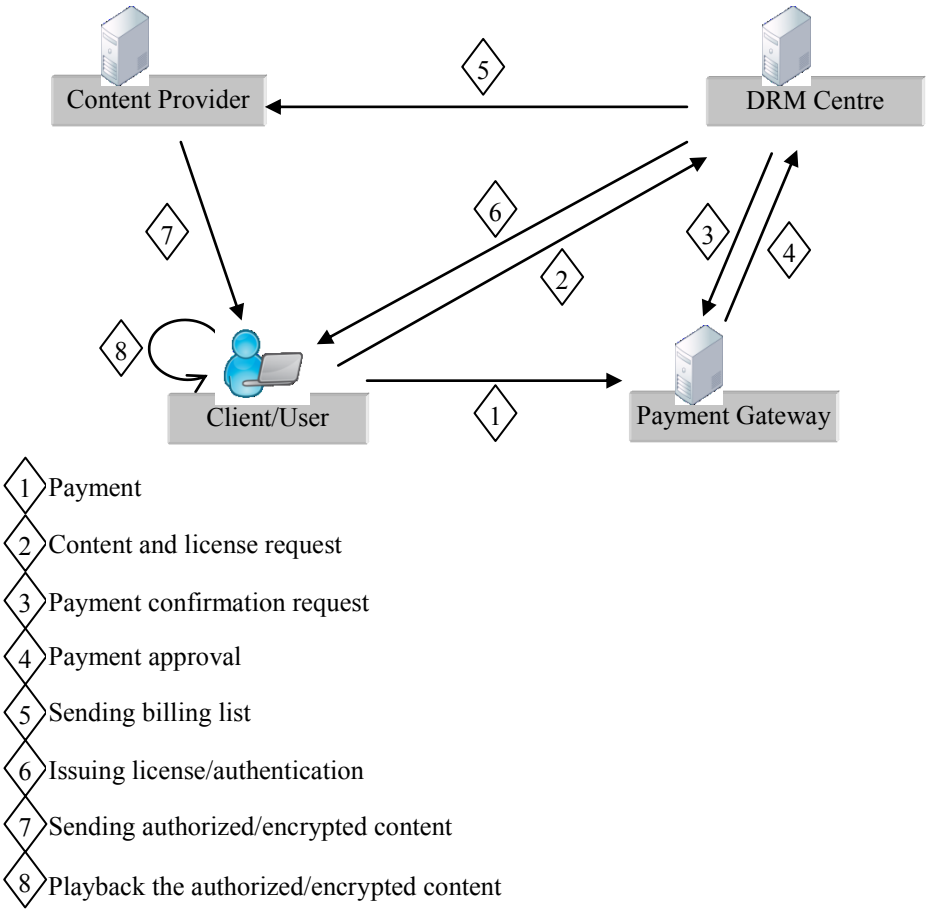


Figure 1.1: An example of a typical DRM system.

- The user pays for the content and gains it through the provider.
- The content provider or Internet service provider (ISP) takes the billing list from the DRM centre and sends the authorized or encrypted content to the user.
- The payment gateway takes payment request from the DRM center and sends an approval of payment to it.
- The DRM center takes content request from the user after payment approval and then sends the billing list to the content provider or ISP and authentication (e.g. license or decryption key) to the user.

Nowadays, the DRM technology has been adopted by the entertainment industry, most e-book publishers and online music stores. Electronic books read on a personal computer or an e-book reader typically use the DRM technology to limit copying, printing, and sharing of e-books. Examples of the e-book DRM schemes commonly used today are Adobe's ADEPT DRM [9] and Apple's FairPlay DRM [10-11]. As for the multimedia industry, with the aim of restricting usage of the media content purchased and downloaded, they employ the DRM technology in various kinds of business scenarios that include [12]:

- **Online scenarios**: These scenarios require users to be online while they play back the media content:

  - **Live streaming:** Live streaming sends the content directly to the computer or device without saving the file to a hard disk. A live stream is only available while it is being broadcasted. Internet television and radio are examples of live streaming.
  - **Progressive download**: Progressive download lets users play back the media while it is downloading. The main difference between progressive download and live streaming, from a user's point of view, is that progressively downloaded content is stored on the user's computer or device, at least temporarily.
- **Offline scenarios**: These scenarios allow users to be offline while they play the content.
  - **Download file offline (onetime purchase)**: Users download the content from the Internet and play it later whenever they want. The DRM software restricts redistribution of the content to one or more devices.
  - **Subscription**: It enables the customers to play back the content based on a subscription model. For example, customers of the online television channel pay a monthly fee to watch up to 100 hours of television content online. In order to renew their subscription, they need to pay the monthly fee and connect to the service at least once a month because the subscription license expires every 45 days.

The followings are examples of the online music stores that employ the DRM technology. Prior to 2009, Apple's iTunes store utilized the FairPlay DRM system [10-11] that enables playback of the iTunes music only on Apple devices and Apple's QuickTime media player. In recent years, music tracks with DRM-free iTunes Plus format are available at a higher price point. Napster music store [13] offers a subscription-based DRM approach alongside permanent purchases. Users of the subscription service can download and stream an unlimited amount of music during the subscription period. However, when the subscription period lapses, all the downloaded music is unplayable until the user renews his or her subscription. Prior to 2008, Sony operated a music download service called Connect [14] which used Sony's proprietary OpenMG DRM technology. Music sold at Sony Connect was only playable on Sony hardware and computers running Microsoft Windows.

Although DRM is prevalent for Internet music, it also has an undesirable side of frustrating the authorized customers. From the DRM regulations used by the above mentioned online music stores, we can see that most of the DRM techniques restrict copying or viewing of the contents regardless of whether such copying or other use is legally considered a "fair use" by authorized customers. Under that circumstance, DRM has come under fire. Those opposed to DRM contend there is no evidence that DRM helps prevent copyright infringement, arguing instead that it serves only inconvenience to legitimate customers, and that DRM helps big business stifle innovation and competition [15]. Thus, it has been very important to develop the DRM technologies that ensure not to lock out the legitimate users.

1.2 Problems of encryption on audio data

Among the DRM techniques, encryption plays an important role to protect the contents from unauthorized accesses.

A lot of audio encryption methods in different domains have been proposed in the literature. In the time domain approaches, audio signals are generally encrypted by permuting

the audio samples [16-17], blocks of the samples [16], or the bits which made up of each sample [18]. In the transform domain approaches, the audio signal is first transformed into specified domains (e.g. Fast Fourier Transform (FFT) [19-21]) and then encryption is performed on the transformation results. Basically, any kinds of traditional encryption algorithms, including symmetric or asymmetric ones, can be used for audio protection. However, the naive method of encrypting the entire audio file would destroy compliance with the media standard due to changing data format and increasing file size after encryption, especially in the time-domain approaches. Then, the encrypted audio files could not be played back by existing standard music players and this would hinder the convenient use for legitimate customers. In addition, the problems that will be discussed in the following sections reduce the effectiveness of usual encryption methods to be used for audio protection.

1.2.1 Invalid amplitude problem

In any kinds of audio format, there are valid amplitude ranges for audio samples [22-23], which differ on the basis of the supported bit-depth of an audio coder and data type of the audio samples. As shown in Table 1.1, integer-type audio samples can be encoded in 8-bit, 16-bit, or 24-bit bit-depth, and whereas floating point audio samples can be encoded in up to 64-bit. The higher the bit-depth, the better the audio resolution but the larger file size is yielded.

Before writing an audio file, the audio encoder checks if the audio samples are within the valid amplitude ranges and if they are not, the exceeding parts are clipped. This clipping process becomes a problem for usual encryption algorithms to be directly used for audio protection. Generally, encryption algorithms try to render an audio signal meaningless by applying a series of mathematical operations on the audio samples such as shifting, adding or XOR-ing with key sequences, substituting the samples in accordance with a look-up table, etc. After those operations, it is highly unlikely to keep the audio samples in their original format or size and there is a possibility that the amplitudes of the samples become beyond the valid ranges. Thus, the above clipping process is inevitable and the data losses introduced by clipping will definitely deter the decryption process from successfully recovering the audio signal. Thus, any kinds of audio encryption methods need to pay attention to how to handle this problem. One of the proposals in this thesis presents a solution for this problem.

1.2.2 Operational conflict problem with data hiding methods

Encryption and data hiding are widely used DRM techniques in which encryption mechanism protects the content from unauthorized accesses and data hiding mechanism ensures copyright protection by embedding the watermark (e.g. copyright related messages or signatures) into the audio signal without sacrificing the audio quality. However, encryption techniques protect the data just before decryption. Once the data is decrypted, there is no more protection. On the other hand, data hiding techniques cannot provide secure distribution. For those reasons, employing encryption and data hiding techniques individually is not fully desirable for today's DRM systems. There has been a lot of interest in developing a combined scheme of encryption and data hiding for ensuring both confidential distribution and copyright protection.

Table 1.1: Valid audio amplitude ranges for different bit-depth systems.

Bit-Depth	Valid Range for Respective Data Type	
	Integer	Floating Point
8-bit	[0, 255]	[-1.0, +1.0)
16-bit	[-32768, +32767]	[-1.0, +1.0)
24-bit	[-2^23, 2^23-1]	[-1.0, +1.0)
Higher bit-depth		[-1.0, +1.0]

Kothamasu and Sehgal [24] proposed a system that uses both encryption and data hiding techniques in which a watermark image is embedded into an audio signal on the basis of the energy of the signal in the discrete wavelet and cosine transform domains. Before embedding, the watermark image is encrypted by using the Arnold transformation [25], which is a common image transformation technique in the two-dimensional domain. A similar scheme was also proposed by Mondal and Mandal [26] in which data hiding is done in the time domain and the watermark (here it is a speech signal) is encrypted by using the chaos based encryption [27]. The main purpose of these methods is to secure the watermark information in case the data hiding methods break but not for secure distribution of the host audio signal.

However, if the main purpose of such a combined scheme is to provide both secure distribution and copyright protection, encryption must be done on the host audio signal, not on the watermark information. In that case, usual encryption methods are not appropriate for combining with data hiding methods. It is because both encryption and data hiding try to change the values of the audio contents and thus cause operational conflicts with each other. Let us consider two example scenarios of combining the methods in which encryption is to perform bitwise XOR between audio samples and keys and data hiding is to replace the least significant bit (LSB) of each audio sample with the watermark bit. For an audio signal s, the first scenario is as follows:

1) The watermark information is embedded on s: $m(s)$
2) The marked file is encrypted: $e(m(s))$

To detect the watermark in this scenario, the above steps must be done in exact reverse order. That is, the watermark information cannot be detected from the encrypted signal $e(m(s))$.

For the second scenario,

1) The audio signal is encrypted: $e(s)$
2) The watermark information is embedded on the encrypted signal: $m(e(s))$

In this scenario, decryption before completely removing the watermark would result in random decrypted values and the signal with nearly original quality could not be successfully recovered. Moreover, most of the audio watermarking methods proposed in the literature are irremovable methods in which removing the watermark will surely degrade the audio quality.

Let us consider how the above conflicts reduce the effectiveness of the DRM systems in the application scenario shown in Fig. 1.2: in DRM, the media is usually encrypted for confidentiality before distribution. The encrypted media may then be transmitted from owners to

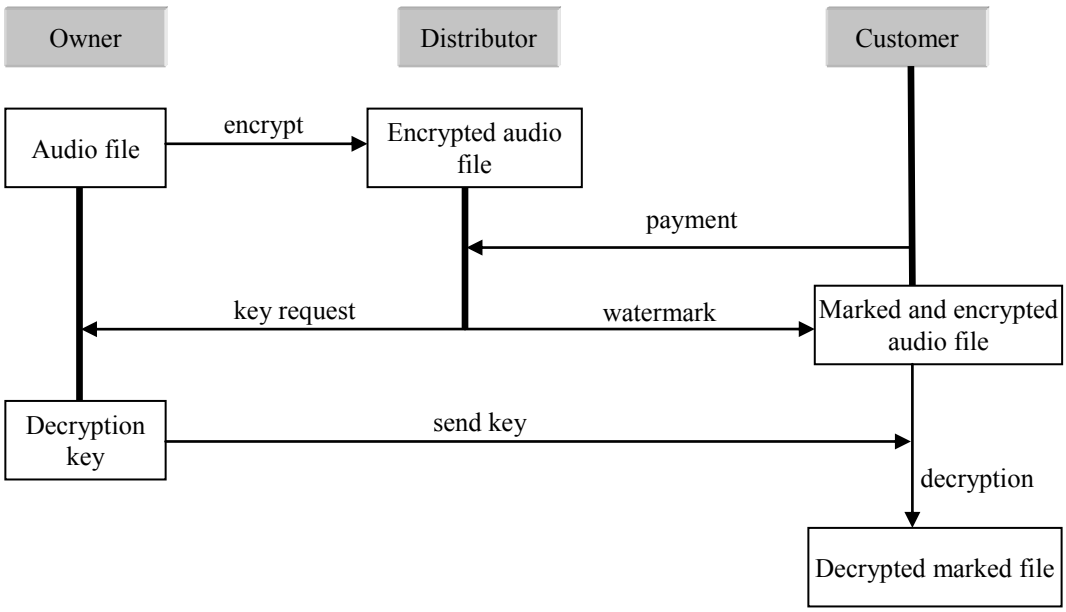


Figure 1.2: Online audio distribution scenario in DRM environment.

customers through distributors. Those distributors have no right to access the unencrypted media and are only entitled to distribute the encrypted media to end users and request the license server in the DRM system to issue the license containing the decryption key to end users. However, they sometimes need to watermark the media for copyright violation detection, distribution control, proof of distributorship, or traitor tracing, and must embed the watermark on the encrypted content. As for customers, they can get the decryption keys from owners for a fee and then enjoy the high-quality content after decryption. Note here that decryption must be done on the marked content. If the usual encryption methods were applied in this scenario, they would completely destroy the objective of data hiding because decryption would never be successful without removing the watermark.

Thus, it is very important to develop effective audio encryption methods that can be used together with the data hiding process without causing any conflicts. This thesis proposes two audio scrambling methods, which are kinds but not direct applications of audio encryption, which can be effectively applied in the above application scenario.

1.3 Audio file formats

This section discusses the uncompressed and compressed audio file formats. An audio file format is a file format for storing digital audio data on a computer system. The bit layout of the audio data (excluding metadata) is called the audio coding format and can be uncompressed or compressed to reduce the file size by often using the lossy compression. There are three major groups of audio file formats:

- Uncompressed audio formats, e.g. waveform audio file format (WAV) [28-29], audio interchange file format (AIFF) [30]
- Compressed audio formats (lossless), e.g. WavPack [31]

- Compressed audio formats (lossy), e.g. advanced audio coding (AAC) [32], MPEG-1 layer III (MP3) [33-35]

As this thesis proposes audio encryption methods for WAV and MP3 audio formats, the following sections explain those file formats in detail.

1.3.1 Waveform audio file format (WAV)

Waveform audio file format (commonly known as WAV) [28-29] is a Microsoft and IBM audio file format standard for storing an audio bitstream on personal computers (PCs). It is the main format used on Windows operating systems for raw and typically uncompressed audio. The usual bitstream encoding is the pulse code modulation (PCM) format. Even though an uncompressed WAV file is large and thus not appropriate for file sharing over the Internet, it is a commonly used file type suitable for retaining audio files of high quality, for using in applications like audio editing where the time needed in compressing and decompressing data is a concern or on a system where disk space is not a constraint.

The WAV is an application of the Resource Interchange File Format (RIFF) bitstream format method for storing data in chunks. Fig. 1.3 shows the WAV file format. Every chunk (including the header) starts with the chunk ID that defines what is included in the chunk, e.g. “RIFF” chunk, “fmt” chunk, and “data” chunk. The other fields in the RIFF chunk define the size of the overall file and file format (here it is “WAVE”). The format chunk is the metadata chunk which describes the necessary information to decode the WAV file such as the sampling rate, the bit-depth, the number of channels, etc. The data chunk contains the audio sample data whose data type and range vary depending on the chosen bit-depth, as discussed in Sect. 1.2.1.

	Field Name	Value
The “RIFF” chunk	ChunkID	“RIFF”
	ChunkSize	varies
	Format	“WAVE”
The “fmt” sub-chunk	Subchunk1ID	“fmt”
	Subchunk1Size	16 for PCM
	AudioFormat	1 for PCM, others for compressed formats
	NumChannels	1 for Mono, 2 for Stereo, etc
	SampleRate	8000, 44100, etc
	ByteRate	SampleRate*NumChannels*BitDepth/8
	BlockAlign	NumChannels*BitDepth/8
	BitDepth	8, 16, etc
The “data” sub-chunk	Subchunk2ID	“data”
	Subchunk2Size	varies
	Data	sample data

Figure 1.3: The canonical WAV file format.

1.3.2 MPEG-1 layer III audio file format (MP3)

The MP3 is a compressed audio format which is designed to greatly reduce the amount of data required to represent the audio recording and still sound like a faithful reproduction of the original uncompressed audio for most listeners [33-35]. Nowadays, the MP3 is a de facto standard of digital audio compression for the transfer and playback of music on most digital audio players as well as a common format for consumer audio streaming or storage.

All MP3 files are divided into smaller fragments called frames. Each frame stores 1,152 samples and lasts for 26 ms. As shown in Fig. 1.4, which is the structure of an MP3 frame, a frame consists of five parts. The header contains important information such as the synchronization word, the sampling rate, the bitrate, and so on. The synchronization word found at the beginning of each frame enables the MP3 decoders to lock onto the signal at any point in the stream [33]. This makes it possible to broadcast an MP3 file. A receiver tuning in at any point of the broadcast just has to search for the synchronization word and then start playing.

The Cyclic Redundancy Check (CRC) field is used to check if there are transmission errors in the header. According to the standard, bits 16 to 31 in both the header and the side information field are very sensitive to errors to such an extent that it can corrupt the whole frame, whereas an error in the main data only distorts a part of the frame. A corrupted frame can either be muted or replaced by the previous frame [33].

The side information field carries the necessary information to decode the MP3 main data (scalefactors and Huffman codes). For instance, it tells the decoder where to find the start of the main data in a certain frame and which Huffman code tables to be used to decode that frame. In the main data part, the purpose of scalefactors is to reduce the quantization noise. If the samples in a particular scalefactor band are scaled in the right way, the quantization noise will be completely masked. The second part of the main data consists of Huffman encoded bits. Finally, the ancillary data part is optional.

Figure 1.5 shows a series of the MP3 encoding processes. First, the input PCM signal is transformed from time-domain samples to frequency lines by passing through a filterbank cascaded by a windowed Modified Discrete Cosine Transform (MDCT). The input PCM also passes through the FFT process to yield the input for psychoacoustic model which provides information to control the MDCT windows type: short windows for adjacent frequency spectra with certain changes and long windows for others. This model also provides a set of masking thresholds used to discard audio redundancy. That is, frequency components under those thresholds cannot be perceived by human ear and thus the MDCT frequency lines can be non-linearly quantized by using the lowest possible bitrate as long as the quantization noise is kept under those thresholds. Then the quantized values are divided into different regions and encoded with different Huffman tables that are tuned for the statistics of particular region. Finally, an MP3 frame is constructed by appending a header as shown in Fig 1.4.

For decoding an MP3 bitstream, as shown in Fig. 1.6, the decoder firstly identifies every frame in the bitstream by searching for the synchronization word. If the protection bit in the header is set, the CRC field exists and checks the most sensitive data for transmission errors. Then the MP3 main data (scalefactors and Huffman coded data) are decoded by using the decoding parameters specified in the side information part. The decoded scalefactors are later

used when re-quantizing the Huffman decoded values (quantized values). Then, the Inverse MDCT (IMDCT) process is carried out on the re-quantization results (frequency lines). Finally, the synthesis filterbank reconstructs the signal.

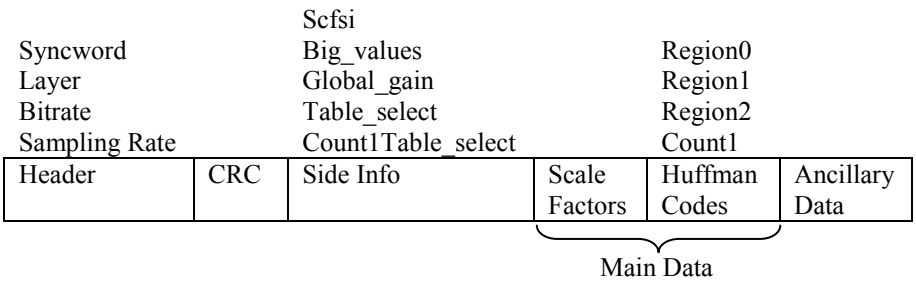


Figure 1.4: MP3 frame format.

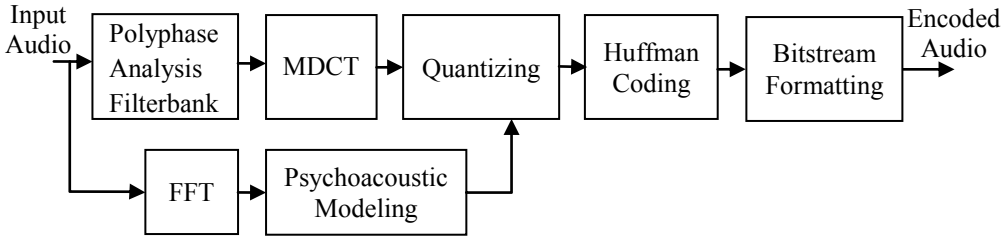


Figure 1.5: MP3 encoding process.

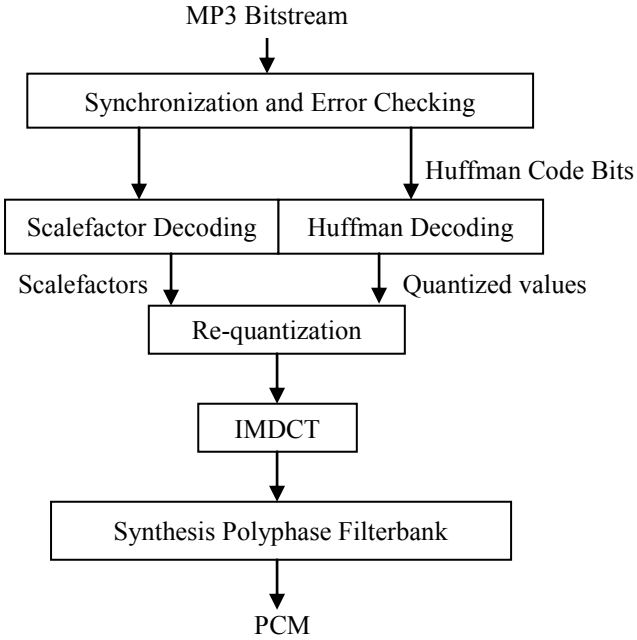


Figure 1.6: MP3 decoding process.

1.4 Partial encryption

Regarding audio protection, partial encryption methods have more desirable features than usual encryption of the whole file. The main concept of partial encryption is to protect the entire content by encrypting only the perceptually important parts which have smaller size. Unlike encrypting the whole file, partial encryption just degrades the audio quality but not completely destroys the file. Thus, if partial encryption is carefully carried out not to destroy the media format, the encrypted file can be played back by existing music players without need to decrypt. This feature is very attractive for realizing the try-before-purchase model, which is one of the important business models of DRM.

Let us consider a real world audio distribution scenario. A customer walks up to the music counter in the store and pick up an interesting CD. By using the CD player and headphone provided by the store, the customer checks the taste of the music. If the customer thinks it is worth a buy, he/she walks up to the counter and pays for it. As for online music stores, they should also provide preview music files for commercial purpose. Customers should be able to listen to those files via online streaming or by downloading to their own devices. The simplest scenario for providing the preview music is as follows: a short part of the music file is provided for preview and any customers can freely access it; for a fee, the customers are allowed to download the whole music file. Regarding this scenario, there are three undesirable facts. Firstly, it consumes the network bandwidth because of the need for two times of file transfer (for downloading the preview file and the whole file). Secondly, providing the whole unprotected music file for downloading after payment is not desirable. During data transfer between owner and customer, there might be an adversary who would illegally capture the file. Thirdly, providing a short part of the music file for trial may not be a good idea because it has a tendency to reduce the sales of mobile ringtones or ring-back tones. According to the reports over the Internet, sales of ringtones make up only 63 percent of the overall mobile music market in recent years, whereas in 2007, they made up 80 percent of the market. The main reasons for such a drop are:

- Due to the development of high-tech smart phones, it is very easy to create customized ringtones;
- Musical ringtones could be costly. For instance, the 20- to 30-second snippets were often pricier than buying the whole song. Someone who updated their ringtones frequently could easily pay $20 a month or more.

Thus, using a short part of the music file for trial may add up fire to that “ringtones sales drop” problem. In addition, a short part of the music file, e.g. chorus part of the music, may not be good enough for listeners to taste the music. Even though that short part is good, it does not reflect the whole song is good.

This thesis proposes partial audio encryption methods which provide more effective alternative to generate preview music than the above one. The possible application scenario is as follows: the whole music file whose quality is degraded by partial encryption is freely distributed for preview. That file can be played back by any existing music players without need

to decrypt. Even though the quality is degraded, customers can listen to the whole music file and thus they will be able to easily guess what the original music will sound like. After payment, they can get the decryption key (no need to download the music file again) and no doubt a high-quality music file can be enjoyed under the access of the correct decryption key and specific media player. In addition, as the preview file is protected by partial encryption, it is not needed to worry about illegal access during transmission.

Partial encryption on audio data also has other desirable properties of the followings:

- Lower complexity, which is suitable for real time applications;
- Adjustable security level based on the security requirement of the underlying system;
- Restraining synchronization error [36], which occurs when directly encrypted MP3 files are Huffman decoded, to specific frames;

This thesis presents a low-complexity partial encryption method for MP3 security. Thus, the basic concept of partial encryption on MP3 is introduced here. In most cases of partial encryption on compressed audio data, encryption comes after compression process as stated below.

An uncompressed audio signal M passes through the compression process and yields the compressed audio signal Y.

$$Y = Compressed(M). \tag{1.1}$$

While generating Y, the perceptually important parts Y_e for encryption are identified apart from the perceptually less important parts Y_c to be left as plain.

$$Y = Y_e \cup Y_c. \tag{1.2}$$

After encryption,

$$Y_e' = E(Y_e, key), \tag{1.3}$$

where $E(.)$ represents an encryption process in general and Y_e' is the encrypted signal. After bitstream formatting, the compressed-encrypted audio signal M' is obtained.

$$M' = Y_e' \cup Y_c. \tag{1.4}$$

The M' must be decodable by any MPEG standard decoders without decrypting Y_e'. The effectiveness of partial encryption depends on the choice of perceptually important parts for encryption. Chapter 2 of this thesis discusses how to effectively choose the perceptually important parts during the MP3 encoding process.

1.5 Advanced Encryption Standard (AES)

The methods proposed in Chapter 2 of this thesis use the Advanced Encryption Standard (AES) algorithm for audio encryption and thus the AES algorithm is briefly described in this section.

The AES is based on the Rijndael cipher [37] designed by Vincent Rijmen and Joan Daemen and proposed to the National Institute of Standards and Technology (NIST) during the AES selection process. It was announced as the Federal Information Processing Standards (FIPS)-approved cryptographic algorithm by the NIST in 2001 [38].

The AES is based on a design principle known as a substitution-permutation network. It is a symmetric key block cipher [37-38]. As shown in Table 1.2, the lengths of the input block

and the output block are 128 bits. The length of the cipher key is 128, 192, or 256 bits. The number of the rounds to be performed during the execution of the algorithm is dependent on the key length. Internally, the operations in the AES algorithm are performed on a two-dimensional (4×4) array of bytes called the state. Each individual byte in the state has two indices and can be indexed as either $s_{r,c}$ or $s[r,c]$, for $0 \le r < 4$ and $0 \le c < 4$. For executing the cipher or decipher process, the input array of bytes $[in_0, in_1, ..., in_{15}]$ is copied into the state array, then the ciphering and deciphering operations are conducted on this state array, and the final results are copied to the output array of bytes $[out_0, out_1, ..., out_{15}]$ as shown in Fig. 1.7. The copying from the input array and to the output array is carried out according to the following schemes:

$$s[r,c] = in[r+4c] \quad \text{for } 0 \le r < 4 \text{ and } 0 \le c < 4. \tag{1.5}$$

$$out[r+4c] = s[r,c] \quad \text{for } 0 \le r < 4 \text{ and } 0 \le c < 4. \tag{1.6}$$

As described in the high level description of the AES algorithm, the cipher and decipher processes use a round function that is composed of four different byte-oriented transformations: 1) SubBytes, 2) ShiftRows, 3) MixColumns, and 4) AddRoundKey. In the AddRoundKey step, each byte of the state is combined with a block of the round key using bitwise XOR.

Table 1.2: Algorithm specification of the AES.

	Key Length (bits)	Block Size (bits)	Number of Rounds
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

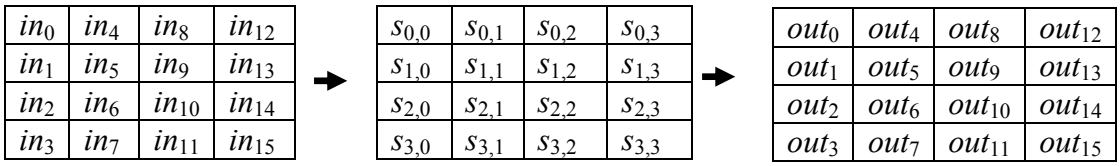


Figure 1.7: The state array input and output in the AES.

AES Algorithm: High Level Description

1. KeyExpansions - round keys are generated on the basis of the cipher key. AES requires a separate 128-bit round key block for each round including the initial round;
2. InitialRound
   - AddRoundKey – adding a round key to the state;
3. Rounds
   - SubBytes – byte substitution using a lookup table;
   - ShiftRows – shifting rows of the state array by different offset;
   - MixColumns - mixing data within each column of the state array;
   - AddRoundKey
4. Final Round (no MixColumns)
   - SubBytes
   - ShiftRows
   - AddRoundKey

The SubBytes transformation shown in Fig. 1.8 is a non-linear byte substitution that independently operates on each byte of the state using a substitution table (S-box). For instance, if $s_{1,1}$={53}, then the substitution value ($s'_{1,1}$) would be the one in the S-box determined by the intersection of the row with index 5 and the column with index 3.

In the ShiftRows transformation, as shown in Fig. 1.9, the bytes in the last three rows of the state are cyclically shifted over different number of bytes (offsets). The first row, r=0, is unchanged. Specifically, the shifting proceeds as follows:

$$s'_{r,c} = s_{r,(c+shift\_offset)} \text{ for } 0 < r < 4 \text{ and } 0 \le c < 4, \tag{1.7}$$

where *shift_offset* depends on the row and it is 1 for r=1, 2 for r=2, and 3 for r=3. Thus, each byte in the second row is shifted one to the left. Similarly, the bytes in the third and fourth rows are shifted to the left by offsets of two and three, respectively. The importance of this step is to avoid the columns being linearly independent, in which case, the AES degenerates into four independent block ciphers.

Like the ShiftRows, the MixColumns transformation provides diffusion in the cipher. As shown in Fig. 1.10, this transformation operates on the state column-by-column, treating each column as a four-term polynomial. The transformation rule is a matrix multiplication as follows:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \text{ for } 0 \le c < 4. \tag{1.8}$$

To perform the decipher process in the AES algorithm, the above cipher transformations are inverted and then implemented in reverse order. The transformations used in the deciphering process are: 1) InvShiftRows - each byte of the second, third, and fourth rows of the state is cyclically shifted to the right by offsets of 1, 2, and 3, respectively; 2) InvSubBytes - each byte of the state is independently substituted using an inverse S-box; 3) AddRoundKey; and 4) InvMixColumns - each column of the state is transformed by multiplying with a matrix.

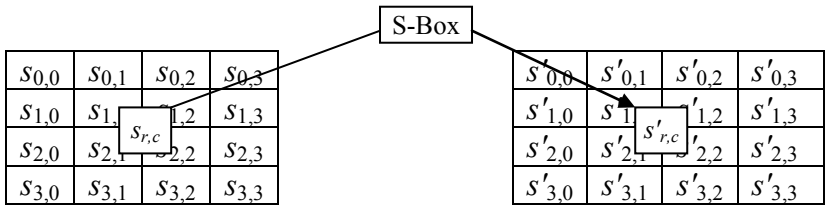


Figure 1.8: The SubBytes transformation in the AES.

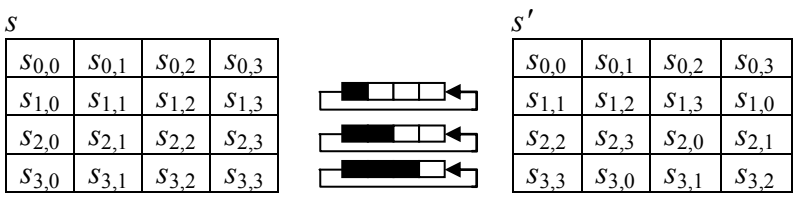


Figure 1.9: The ShiftRows transformation in the AES.

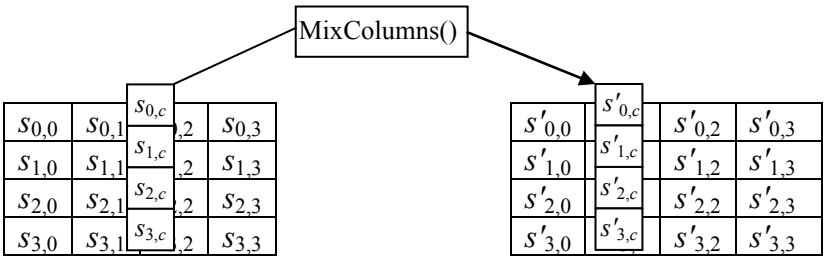


Figure 1.10: The MixColumns transformation in the AES.

Regarding the security of the AES, the U.S. Government announced that the AES could be used to protect classified information [39]. In addition, many papers have been published on the cryptanalysis of the AES and it has survived numerous cryptanalytic efforts [40-42]. It has approved that the 128-bit key size used in the AES is resistant against the known cryptanalysis attacks including the brute force attack. Moreover, the computational complexity of the exhaustive key search for the AES depends on the length of the cipher key:

- For a 128-bit key, the first key recovery attack has computational complexity of $2^{126.1}$;
- For a 192-bit key, the first key recovery attack has computational complexity of $2^{189.7}$;
  For a 256-bit key, the first key recovery attack has computational complexity of $2^{254.4}$;

As for the implementation, the AES is fast in both software and hardware [43-48]. It can be implemented to run at speeds unusually fast for a block cipher on a Pentium (Pro) [37].

In this thesis, although there is no restriction on the choice of encryption algorithms to be used for the proposed methods, we choose the AES due to its speed, high level security, and ease of implementation.

1.6 Audio scrambling

Nowadays, audio scrambling methods [16-17, 49-52] are widely used to provide confidentiality in audio distribution. Transposition-based audio scrambling methods are kinds but not direct applications of audio encryption. They try to reduce the residual intelligibility of an audio signal by breaking the coherence between data contents so that the signal is unintelligible to unintended recipients of the communication. As they neither inject any new values nor change the values of the existing contents, they do not stop compliance with the audio format or neither increase file size nor change data format after scrambling. In addition, they do not create the operational conflict problem discussed in Sect. 1.2.2. They enable us to embed the watermark on the scrambled signal and to extract it from either the scrambled or descrambled version of the signal as well. Thus, they can be used more effectively as pre- and post-processing of the data hiding process rather than the usual encryption methods.

A scrambling algorithm can be evaluated in many aspects, where the most important of which are as follows:

- **Scrambling effect**: even for one-time scrambling, the scrambling degree should be achieved to a high extent;
- **Efficiency**: the scrambling and descrambling processes should not be too complex;
- **Security/anti-decryption capability**: even if the algorithm is made public, descrambling should still be difficult or even impossible;
- **Perceptual quality**: the scrambled signal should achieve very low or even zero residual intelligibility and the descrambling process must also be able to recover the signal with nearly original quality;
- **Execution time**: as long as there is no harm to security, the shorter the time taken for scrambling/descrambling, the more efficient the algorithm;

In this thesis, with the purpose of applying together with data hiding methods, two low-complexity audio scrambling methods are proposed. Their performances are also thoroughly evaluated in terms of the above mentioned aspects.

1.7 Objective of this thesis

As described in the previous sections, the development of audio encryption methods that keep compliance with the media standard and that can be used together with data hiding is very important for today's DRM systems. In addition, those methods should not create inconvenience for the authorized users to access and enjoy the audio files. This thesis proposes low-complexity audio encryption methods that satisfy those requirements.

Firstly, Chapter 2 introduces the concept of partial encryption on uncompressed audio data together with a solution for invalid amplitude problem discussed in Sect. 1.2.1. Then, it focuses on a low-complexity partial encryption method for MP3. The proposed method is embedded in the MP3 encoding process in which only the quantized values that are perceptually important according to the Human Auditory System (HAS) are encrypted. It reduces execution time by encrypting only the parts of an MP3 file rather than the whole file. The resulting encrypted file is still compatible with the MPEG standard so it can be rendered by any existing MP3 players. For full-quality rendering, decryption using the appropriate cryptographic key is necessary. Moreover, the effect of encryption on audio quality can be flexibly controlled by adjusting the percentage of encryption. On the basis of this feature, we can realize the try-before-purchase model discussed in Sect. 1.4: users can render the encrypted MP3 files for trial and enjoy the contents in original quality by purchasing the decryption keys. From our experiments, it turns out that encrypting 2-10% of the MP3 data suffices to generate trial music, and furthermore file size increase after encryption is subtle.

Secondly, Chapter 3 proposes two fast and simple transposition-based audio scrambling methods: one is based on the pre-order traversal of a complete binary tree and the other on a pseudorandom number generator (PRNG) called Mersenne Twister (MT). Since the methods only affect the positions of the audio samples but not the values, they can be used together with data hiding methods without the operational conflict problem discussed in Sect. 1.2.2. According to the experimental results, both methods are very effective in terms of time and space complexity and the scrambling effect. However, their cryptographic securities are limited.

Thirdly, with the aim of strengthening the cryptographic security of the methods proposed in Chapter 3, Chapter 4 proposed two new schemes that combine each of those methods and the Discrete Wavelet Transform (DWT). First, an audio signal is wavelet decomposed and the layers of wavelet coefficients are retrieved. Then the coefficients in each layer are separately scrambled by considering not only the pre-order but also the in-/post-order based scrambling methods in the first scheme and by using the MT based scrambling method with a series of keys in the second scheme. Anyone who does not know the correct wavelet decomposition parameters and the correct method/key used for each layer will not be able to successfully descramble the signal. In addition, the new schemes also achieve the progressive scrambling effect in which the audio outputs with different quality levels can be generated by scrambling layer-after-layer. During the descrambling process, the audio signals with low-to-high quality can be progressively recovered by descrambling layer-after-layer. This feature is very attractive because it enables the audio quality to be controllably degraded on the basis of the system requirement: slightly distorted ones for the try-before purchase model of the DRM systems and severely distorted ones for the systems with strong security needs.

The rest of this thesis is organized as follows. Chapter 2 discusses two low-complexity partial encryption methods for uncompressed (WAV) and compressed (MP3) audios along with the experimental results. Chapter 3 proposes two transposition-based audio scrambling methods. The evaluation results of their scrambling effect, execution time, and cryptographic security are also described. Chapter 4 discusses how to enhance the cryptographic security of the methods proposed in Chapter 3 with the detailed evaluation results. Finally, Chapter 5 concludes and summarizes this thesis.

Chapter 2

Partial Encryption Methods that Enhance Audio Security

This chapter introduces the concept of partial encryption on raw audio (WAV) with a solution for invalid amplitude problem discussed in Sect. 1.2.1 as well. Additionally, Sect. 2.3 presents a low-complexity partial encryption method for MP3 as well as a detailed suggestion on how to effectively choose the data for encryption during the MP3 encoding process.

2.1 Introduction

As discussed in Sect. 1.4, partial encryption has more desirable features than usual encryption of the whole file such as reducing complexity and yielding low-quality audio signals. A lot of works on partial/selective encryption of compressed audio (MP3) have been proposed in the literature. All of those works are different based on the fact that how and which data are chosen for encryption.

During the MP3 encoding process, the input PCM data transformed into several variants: frequency lines, quantization values, etc. Thorwirth et al. [53] proposed a partial encryption scheme that encrypts some selected frequency regions of the MP3 main data. The audio quality was degraded on the basis of the regions encrypted. However, neither the detailed procedure of encryption nor which frequency regions were selected for encryption was discussed.

Servetti et al. [54] proposed a more detailed selective encryption scheme on frequency regions that yields the encrypted MP3s with low-pass filter quality. It was realized as follows: the selected frequency regions are limited by encrypting whose frequency lines and ordering the MP3 decoders not to decode them by setting their associated Huffman table indices to not-used values (4 and 14). That system needed to modify some fields in the MP3 header so that the decoder could successfully skip the encrypted parts. Modifying the header and forcing the decoders to behave in exceptional manners might cause problems of compatibility: not all decoders might function as expected.

Torrubia and Mora [55] introduced the term “perceptual cipher” that means encryption that keeps the encrypted data to be perceptually acceptable (albeit lower quality). In that system, encryption was done on two variants of the MP3 data: selected scalefactors and Huffman codes. The selected scalefactors were encrypted by performing the XOR operation with the output of a

PRNG, whereas the Huffman codes were encrypted by replacing the selected codewords with another codewords of the same size from the same codebook. In this system, the encryption on Huffman codes poses a limitation on the choice of encryption algorithm, and only the ones that do not affect file size and data format are suitable.

Steinebach et al. [52] also proposed a partial encryption method on the scalefactor values in which the scalefactors of an MP3 frame are randomly formed into groups in accordance with a key and permuted within their corresponding group without changing the values. That method provided only a weak security because the number of possible permutations was not large enough.

A selective encryption method on quantization values was proposed in [56] in which only the quantized values positioning at even indices are encrypted and the ones at odd indices are left unencrypted. Although the system was simple and easy to implement, the effect of encryption on the perceptual quality was limited.

This chapter proposes a low-complexity partial encryption method for MP3 that can be carried out without extensive computation, impact on compression ratio, and significant modification on MP3 headers. It is also very effective in terms of the perceptual quality because it chooses the data for encryption in accordance with the concept of the HAS. Moreover, the method does not impose any limitation on the choice of cipher algorithms although using the state-of-the-art AES will be a practical advantage.

2.2 Partial encryption on raw audio

As discussed in Sect. 1.3.1, in spite of its large size, the uncompressed/raw audio format (WAV) is a commonly used file type for storing high-fidelity audio files on a system where disk space is not a constraint. The uncompressed WAV files are sometimes used by some radio broadcasters, especially those that have adopted a tapeless system. For instance, BBC Radio in the UK uses the 44.1 kHz 16-bit two-channel WAV audio as standard in their VCS system (system for managing playout including media stores, playout systems and editing systems).

In this section, an effective partial encryption method for uncompressed audio (WAV) is proposed in which the selected parts of a WAV file are encrypted by using the AES algorithm. Regarding the AES, there are some facts which we should pay attention to. Because it is a block cipher, the length of input needs to be multiple of 16 bytes and if they are not, some padding schemes are needed and this may increase file size after encryption. In our system, we choose the size of Y_e (the data to encrypt) to be satisfied multiple of 16 bytes so that the AES can be used with no-padding scheme. The Electronic Code Book (ECB) mode among the AES modes of operations is chosen for the following reasons:

- ECB mode does not have error propagation feature;
- Other block chaining modes need the connected blocks to be decrypted in order and impose limitations in online streaming applications;
- Other modes increase size after encryption due to the use of initialization vector (IV);

For stronger security, the AES/CTR mode can be used instead of the ECB. Like the ECB, the CTR mode independently encrypts and decrypts each input block and hence avoids error

propagation. The CTR mode satisfies all of the above mentioned properties of the ECB except the use of the IV. The IV is used to randomize the encryption and hence it produces distinct ciphertexts even if the same plaintext is encrypted multiple times. Even though it strengthens the security, the IV must be somehow passed to the decryption process for successful decryption. The most common way is to concatenate the IV at the start of the ciphertext for the price of an increase in file size. If the CTR mode is to be applied in the proposed method, we can avoid increasing the file size by appending an additional file header that carries the encrypted IV information at the beginning of an audio track without violating the media standard. We can use the same key or different keys for encrypting the IV and the audio content. In this research, the ECB mode is still chosen because of its simplicity. In addition, although the ECB mode is vulnerable to dictionary attacks, that mode on audio bitstreams makes such attacks very hard due to large alphabet size.

Although high-level security can be achieved by applying the AES, unfortunately, the encrypted results by the AES inevitably face the invalid amplitude problem that was discussed in Sect. 1.2.1 [22-23]. Thus, a solution for that problem is firstly proposed in the following section.

2.2.1 Solution for invalid amplitude problem

As discussed in Sect. 1.2.1, the invalid amplitude problem occurs when the encrypted audio samples are beyond the valid range and thus clipped by the audio coder. In our system, the MATLAB software that we use for simulation deals with any data as double (64-bit) format regardless of the original format and thus the valid amplitude range is [-1, +1]. As the AES algorithm used for ciphering the audio files works on array of bytes, the resulting encrypted audio samples are in the range of [0, 255] and hence need to be scaled to keep compliance with the original format. The proposed scaling scheme is as follows:
After encryption,

$$Y_e'' = Y_e' / 256, \tag{2.1}$$

where Y_e' is the block with encrypted samples. Before decryption,

$$Y_e' = Y_e'' \times 256. \tag{2.2}$$

The only and most important thing is to keep high arithmetic precision accuracy when using Eq. (2.1) and Eq. (2.2) [57]. If we failed to keep high precision accuracy after applying Eq. (2.1) and let the result round off, undesirable quality distortion might still occur in the decrypted audio file.

2.2.2 Procedure

Figures 2.1(a) and (b) show the block diagrams of partial encryption and decryption on raw audio data, respectively. The detailed procedure is as follows [57]:

Parameters: user-defined key (*u_key*), encryption percentage (*p*), and PCM data (*M*);

Step 1: To facilitate the AES's block size restriction, *M* is first divided into 128-sample blocks *S*;

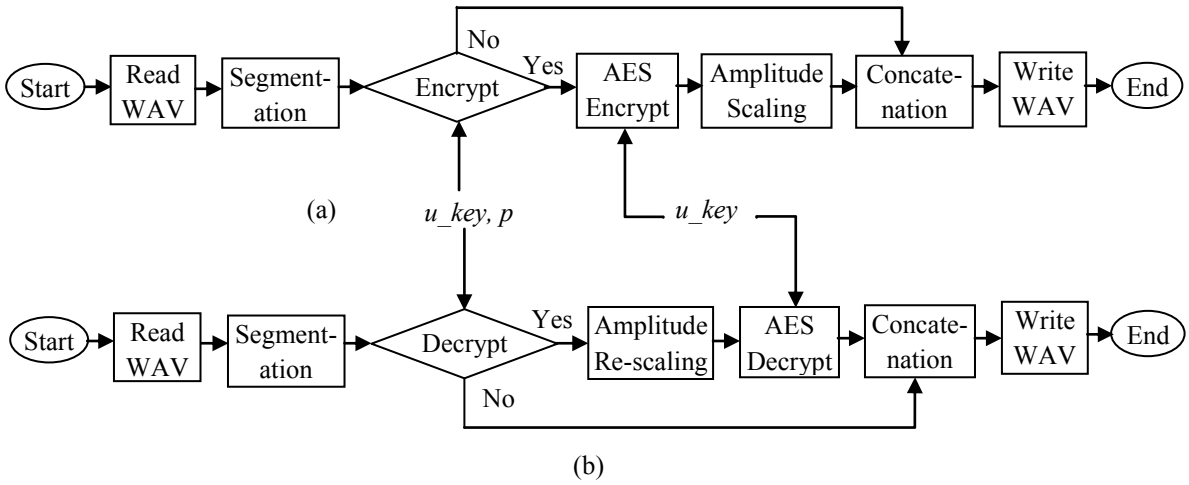


Figure 2.1: System flow of (a) partial encryption and (b) partial decryption on raw audio.

$$S = \bigcup_{i=1}^{l} S_i \text{ where } l = \lfloor length(M)/128 \rfloor \text{and} \qquad (2.3)$$
$$S_i = \{s_{i,j} \in M \mid \text{sample } j \text{ for block } i, 1 \le j \le 128\}.$$

$$M = S \cup (M \setminus S). \qquad (2.4)$$

Step 2: On the basis of p, the number of blocks (B_e) to be encrypted is determined;

$$B_e = \lfloor (p/100) * l \rfloor, \qquad (2.5)$$

where $l = \lfloor length(M)/128 \rfloor$.

Step 3: The blocks for encryption $\{Y_{e,1}, Y_{e,2}, \cdots, Y_{e,B_e}\}$ are randomly chosen from S; for high security, the pseudorandom number generator is seeded with *u_key*.

Step 4: Each Y_e is encrypted and amplitude scaling defined by Eq. (2.1) is applied on the encrypted results of Y_e to make sure compliance with the original audio format.

$$Y'_e = \bigcup_{k=1}^{B_e} \left(E(Y_{e,k}, u\_key) \Big/ 256 \right). \qquad (2.6)$$

Hereafter, the *E(.)* represents the formal AES encryption process. Then the encrypted PCM is,

$$M' = Y'_e \cup Y_c, \qquad (2.7)$$

where Y_c are the samples in M left unencrypted. By restraining the size of Y_e to 128, the AES/ECB mode with no-padding scheme can be used without any trouble and thus M' will also be the same size as M. After writing the M' as a WAV file, the quality-degraded audio file that can be securely distributed is obtained.

When writing the M' (floating point) as a WAV file by using default 16-bit bit-depth, it has to be converted to integer (int16 type). Instead of simple rounding which leads to data loss, the "wavwrite" and "wavread" of MATLAB use Eq. (2.8) and Eq. (2.9) to change data type between floating point and integer [57]. Those functions achieve nearly perfect reconstruction with very small precision loss which has no effect on the decryption process. Thus by decrypting with the authorized use of the *u_key* and *p*, the file with nearly original quality can be successfully recovered.

To convert from floating point x to integer y (int16 type),

$$y = round(x \times 32768). \tag{2.8}$$

To convert from integer (int16 type) to floating point,

$$x = y / 32768. \tag{2.9}$$

2.2.3 Experimental results

In this thesis, we use the MATLAB software to simulate the proposed method on audio signals belonging to different categories: pop, jazz, classical, and speech, which are encoded in 16-bit PCM format with sampling frequency of 48 kHz. The results are thoroughly analysed in terms of perceptual quality, execution time, and security.

The perceptual quality assessment is done by measuring the objective difference grade (ODG) between the encrypted WAVs and the reference unencrypted WAVs. Table 2.1 shows the grading scales of ODG [58]. The ODG, which is an objective measure of the perceived audio quality, is calculated by the perceptual evaluation of the audio quality (PEAQ) algorithm specified in the ITU-R BS 1387-1 [58].

Figure 2.2 shows the ODG values after applying the proposed method on the audio signals with varying p (% of encryption). The p is calculated on the basis of the total number of blocks (each with size of 128 samples) consisting in the audio file. From Fig. 2.2, we can see that the qualities of the encrypted audios are getting worse along with the increasing p. For $p \geq 20\%$, the resulting ODG values are less than -3.2 for all signals. This shows that the proposed method badly destroys the signals to such an extent that the resulting audio qualities are very annoying. For p=2-10%, the ODG values are between -2.2 and -3.6 in general, which mean fine-to-poor audio quality. Thus, it can be concluded that p=2-10% is suitable for generating the preview-quality audio files to realize the try-before-purchase model of DRM.

To test and verify execution time of the method, we simulate the system on a PC with 2.4 GHz, Intel ® Core™ i5-2430M Processor. Figures 2.3(a) and (b) show the execution time results of the proposed method for encryption and decryption respectively. The results are the time taken not only for encryption and decryption but also for pre and post processing (i.e. segmentation and restructuring of an audio frame). For both encryption and decryption processes, we can see from Fig. 2.3 that execution time increases along with the increasing p for all music items. Note that execution time drops for p=100% because pre and post processing are not needed. For all music items, the proposed method only takes about one-fifth of the duration of the original signal even for encrypting half of that signal. For p=2-10%, which we assume that it is suitable for generating trial music, it takes less than one second for encryption/decryption and it is much shorter than the duration of the original music. Therefore, it can be concluded that the proposed method is possibly applied in real time applications.

The security of the proposed method depends on the security of the AES algorithm, whose security has been widely researched in the literature. The longer the key length is, the higher the security that the AES can offer. In our system, we use the AES with 128-bit-long key which is known to give strong resistance against the known cryptanalytic methods including the brute force analysis. In addition, a pseudorandom sequence used to select the blocks of audio samples for encryption/decryption is generated based on the user-defined key. Thus, it is

computationally impossible to generate the same sequence without knowledge of the key used.

2.3 Partial encryption on MP3

As we discussed in Sect. 1.3.2, the MP3 is the most widely used compressed audio format for the transfer and playback of music on most digital audio players. However, it is not equipped with security features to protect the content from unauthorized accesses. This section proposes a low-complexity partial encryption method for MP3 together with a detailed performance evaluation on execution time, perceptual quality of the encrypted signals, etc. Additionally, this section presents a suggestion on how to choose data for encryption in accordance with the concept of the HAS, which is very important for the effective use of partial encryption.

2.3.1 Choice of data to encrypt

The effectiveness of partial encryption on MP3 depends on how to select Y_e (the data to encrypt). The Y_e can be chosen from

Table 2.1: Grading scales of ODG.

ODG	Impairment Description	Audio Quality
0	Imperceptible	Excellent
-1	Perceptible but not annoying	Good
-2	Slightly annoying	Fine
-3	Annoying	Poor
-4	Very annoying	Bad

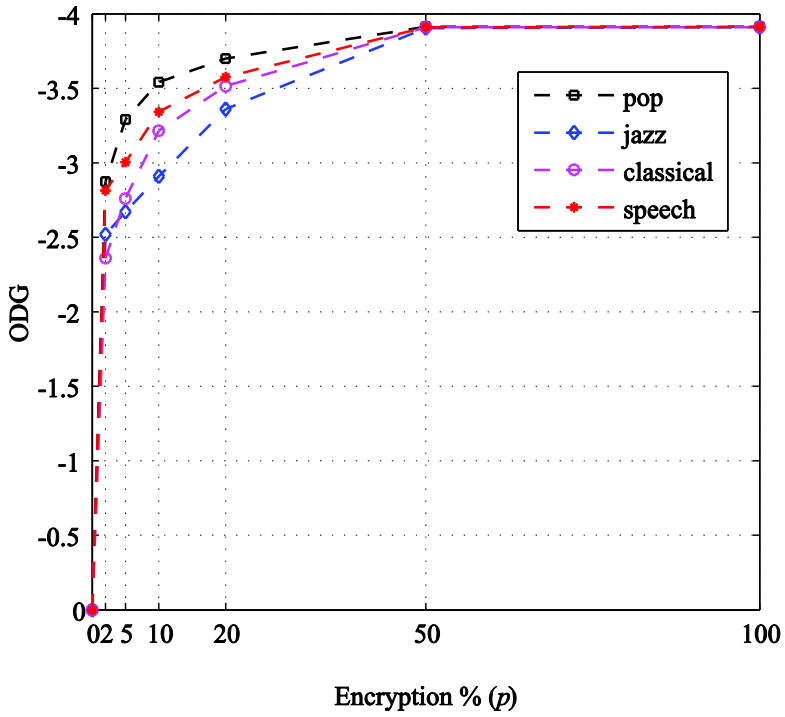


Figure 2.2: ODG vs p of the encrypted WAVs.

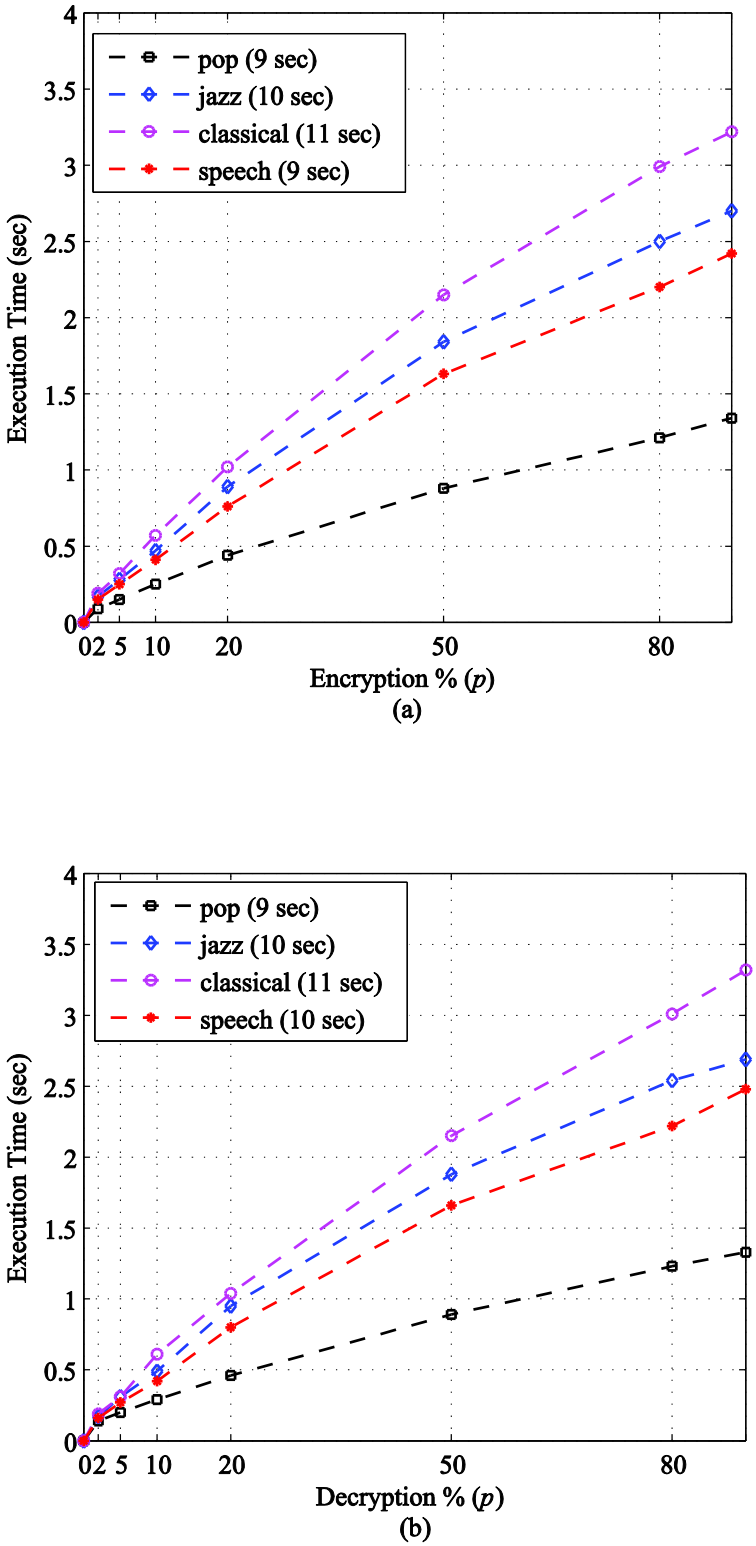


Figure 2.3: Execution time of the proposed method: (a) encryption (b) decryption.

- Header-like information (e.g. side information, CRC, and Huffman tables) or
- Variant of the input PCM and its supplements (e.g. MDCT frequency lines, quantized values, Huffman codes, and scalefactors);

The former are the information needed by the decoder to decode an MP3 file. Encryption on them makes the decoder to decrypt them first before decoding and thus it is not suitable for real time applications and music trial services. The latter are appropriate to be used in all kinds of applications and among them, the proposed method chooses the quantized values as Y_e candidates for the following reasons [57]:

- Scalefactors and Huffman codes are not directly related to perceptibility. We cannot decide which part of them is perceptually more important to choose as Y_e;
- Huffman codes have error avalanche effect [36], and encryption on them leads to synchronization problem;
- Huffman codes are sensitive to encryption algorithms. Only the algorithms which yield the encrypted results with invariable format and length are suitable, e.g. the one that replaces the codewords with other codewords of the same length;
- Although the MDCT frequency lines can be chosen on the basis of perceptibility, encryption on them has a significant impact on compression efficiency of MP3 encoders;

2.3.2 Procedure

As shown in Fig. 2.4, the proposed method can be applied for both online and offline applications. Online applications, e.g. live broadcast, are time critical and thus the encrypted MP3 files should be generated by simultaneously compressing and encrypting raw PCM. In the case of offline applications, encryption is done on already encoded MP3 files and thus partial decoding is needed for obtaining data to encrypt. To understand the details behind the proposed method, we need to have a closer look at the MP3 encoding process.

As discussed in Sect. 1.3.2, each MP3 frame carries 1,152 samples and lasts for 26 ms [33]. After passing through the analysis filterbank and the MDCT process, the end result is subdivided into two granules (each with 576 spectral lines sorted in order of low to high frequencies). The non-uniform quantizer used in the MP3 encoding process makes use of the nature of the HAS to effectively quantize those frequency lines. According to the HAS, most human cannot sense frequencies below 20 Hz nor above 20 kHz. More precisely, frequencies ranging from 2 kHz to 4 kHz are the easiest to perceive, they are detectable at a relatively low volume [33].

To achieve the highest quality with the lowest possible bitrate, the non-uniform quantizer uses a large quantization step size for high frequency lines (leads to small quantized values) and a small step size for low frequency lines. The quantized values are then divided into five regions as shown in Fig. 2.5 and encoded by using different Huffman tables to enhance the performance of Huffman encoder.

Among the five regions, most of the spectral energy of an audio signal accumulates in *big_values* regions, the size of which is indicated by *big_values* field in the side information part of an MP3 frame hence the maximum value is 288. Most common region boundaries, for a 44.1

kHz sampled signal, are 0-2 kHz for region0, 2-5 kHz for region1, and 5-14 kHz for region2. These frequencies are very sensitive to human ear and thus the *big_values* regions are the most suitable for partial encryption.

In the proposed method, Y_e is chosen from the *big_values* regions, and the detailed procedure is as follows:

Parameters: user-defined key (*u_key*), encryption percentage (*p*), and *big_values* (*M*);

Step 1: First the number of frames carried by an MP3 bitstream is calculated. As each MP3 frame is 26 ms long, for a *t*-second long signal, it generally consists of

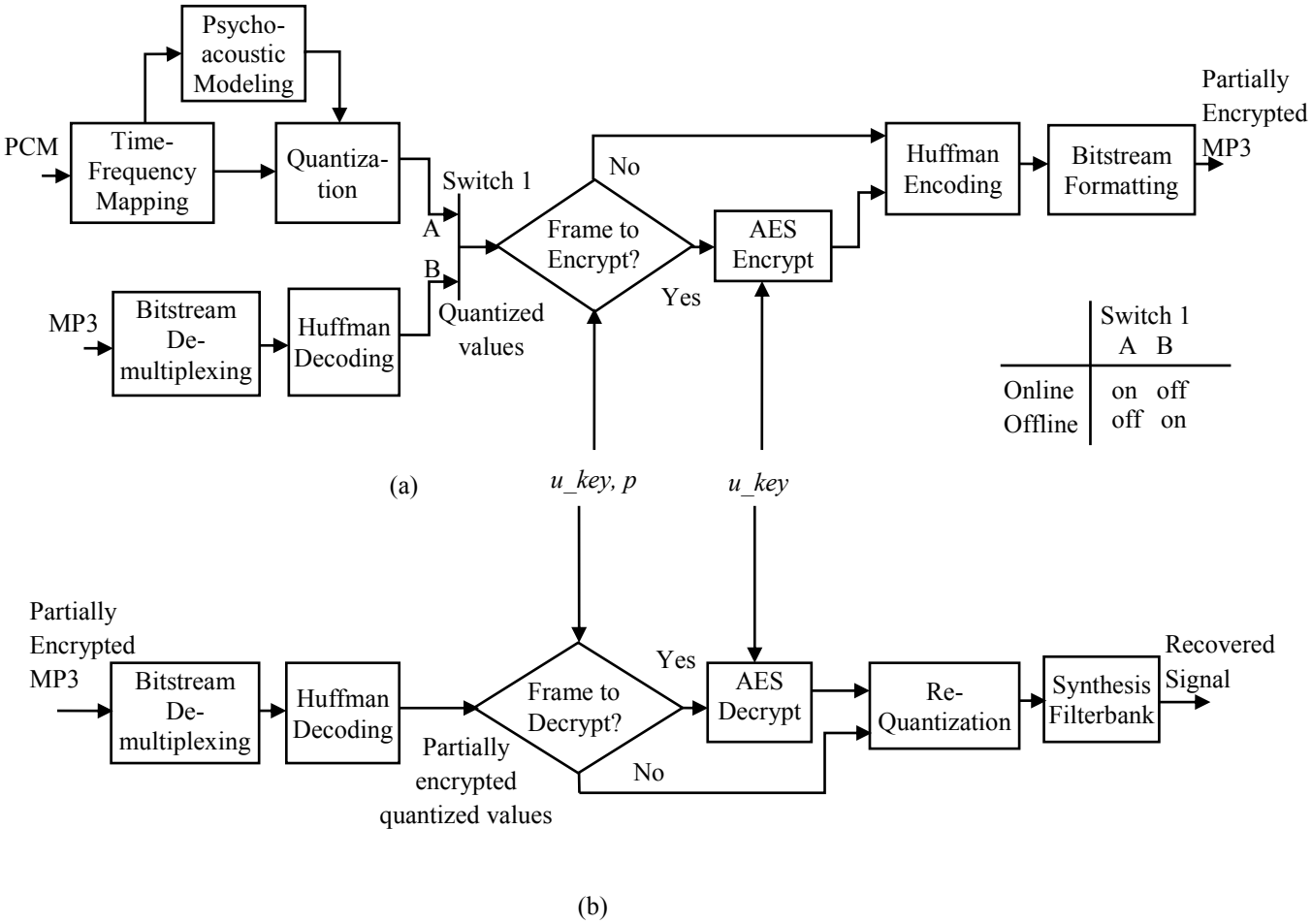


Figure 2.4: Proposed method (online/offline): (a) encryption (b) decryption.

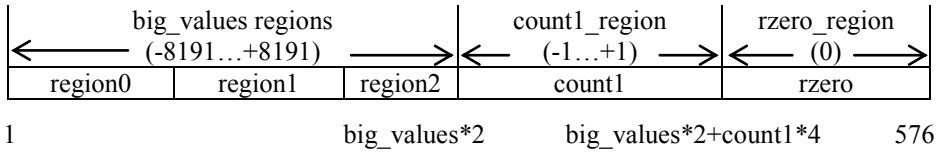


Figure 2.5: Partitioning of quantized values.

$$F = 38 * t \text{ frames.} \tag{2.10}$$

Step 2: On the basis of p and F, the number of frames for encryption (F_e) is determined.

$$F_e = \lfloor (p/100) * F \rfloor. \tag{2.11}$$

Step 3: The F_e frames for encryption are pseudo-randomly selected. Note that each frame is made up of 2 granules (sub-frames) and the following procedure is applied on both granules of the selected frames though we just refer to as frame for simplicity. For each selected frame j, to make the length of the part to encrypt facilitate with the AES block size, the $Y_{e,j}$ is chosen from M_j as follows:

$$\begin{aligned} Y_{e,j} = \{ & Q_{i,j} \in M_j \mid i\text{-}th \text{ quantized value from} \\ & big\_values \text{ of frame } j, 1 \le i \le l * 128\}, \end{aligned} \tag{2.12}$$

where $l = \lfloor length(M_j)/128 \rfloor$.

Step 4: After encrypting the $Y_{e,j}$ and putting back in their original positions, the encrypted M'_j is obtained.

$$Y'_{e,j} = E(Y_{e,j}, u\_key). \tag{2.13}$$

$$M'_j = Y'_{e,j} \cup Y_{c,j} \text{ where } Y_{c,j} = M_j \setminus Y_{e,j}. \tag{2.14}$$

The above procedure is done on all granules of the selected frames. Finally, after Huffman-encoding and bitstream formatting, the quality-degraded MP3 file is obtained. For the offline case where the input is an already encoded MP3 file, partial decoding is needed to obtain M. For decoding a frame and extracting M from it, the decoder has to read the fields in the header and the side information part, the most important of which are the followings:

- **Sync**: points the start of a frame;
- **main_data_begin**: indicates where the main data of a certain frame begins;
- **part2_3_length**: states the number of bits allocated in the main data part of the frame for scalefactors (part2) and Huffman encoded data (part3);
- **big_values**: indicates size of the *big_values* partition;
- **table_select**: specifies the Huffman tables that must be used for decoding the *big_values* partition;

Note here that the quantized values also have their own value range as shown in Fig. 2.5. If the encrypted results do not follow those specified ranges, it will destroy compliance with the MP3 standard and needs to be solved as the invalid amplitude problem discussed in Sect. 1.2.1 and Sect. 2.2.1. In the proposed method, we avoid this problem by encrypting only the *big_values* regions. Because those regions carry the quantized values in [-8191, +8191] range, the encrypted results which are in [0, 255] range are valid.

2.3.3 Experimental results

Using the same music items used in Sect. 2.2.3, performance of the proposed method is evaluated in terms of perceptual quality, execution time, security, and the effect on compression ratio. The LAME encoder [59], which is open-source software, with default bit rate setting (128

Kbits per second) is used for encoding MP3 files. The perceptual quality assessment is done by measuring both ODG [58] and subjective difference grade (SDG) [60] between the original WAVs and the encrypted MP3s. The grading scales of SDG are shown in Table 2.2.

Figure 2.6 shows the ODG values of applying the proposed method on MP3 audios with varying p (% of encryption). The p is calculated on the basis of the total number of frames (each with size of 1,152 samples) that make up of an MP3 file. As shown in Fig. 2.6, the ODG values are getting smaller along with the increasing p. This means that the qualities of the encrypted MP3s are getting worse when the encryption percentage is getting larger. Note that the ODGs are around -0.3 even for p=0% because of the lossy MP3 compression. For all music items, the ODG values are less than -3.6 for $p \geq 20\%$ and show that the qualities of the encrypted MP3s are very annoying. For p=2-10%, the resulting ODG values are generally between -2.7 and -3.6 for all signals. These ODG values refer to fine-to-poor audio quality and hence it can be concluded that encrypting 2% to 10% of an MP3 file is suitable for generating a preview-quality file to be used in the try-before-purchase model of DRM.

Table 2.2: Grading scales of SDG.

SDG	Impairment Description	Audio Quality
5	Imperceptible	Excellent
4	Perceptible but not annoying	Good
3	Slightly annoying	Fine
2	Annoying	Poor
1	Very annoying	Bad

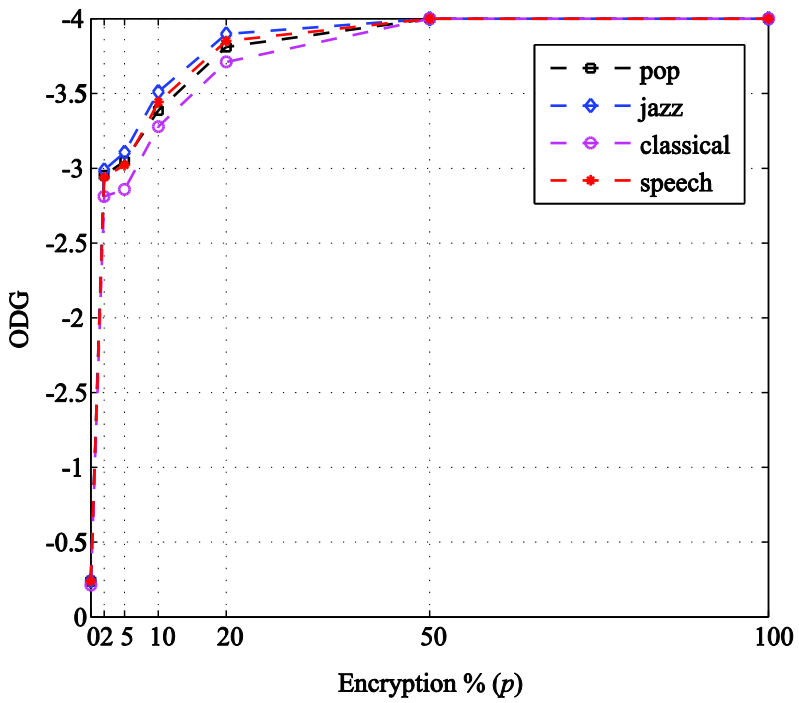


Figure 2.6: ODG vs p of the encrypted MP3s.

To make the above assumption concrete, we also conducted subjective tests in which ten untrained subjects were given seven different audio files for each test without mentioning anything about encryption. One is the original WAV file and the others are the encrypted MP3 files. Firstly, they were asked to compare the qualities of the WAV and the other files. Then, using the WAV file as a reference, they were asked to grade the quality impairment of the MP3 files in accordance with the subjective grading scales shown in Table 2.2. The final grading results (G) calculated by using Eq. (2.15) are shown in Table 2.3. All of the listeners could correctly identify 100% encrypted files because they sound like a white noise signal. Most of them could perceive the gradual decrease in audio quality along with the increasing p starting from 5% to 100%. Some could even correctly differentiate between 2% and 5% encrypted files even though the quality impairment difference is very small. All of them confirmed that the quality impairment caused by p>10% is too annoying to use as a trial music.

$$G = \frac{1}{N_E} \sum_{k=1}^{N_E} SDG_k, \tag{2.15}$$

where N_E is the number of evaluators and SDG_k is the subjective measure graded by k^{th} evaluator.

Figure 2.7(a) shows the execution time results of the method for both online and offline cases. For online processing, the results are the total time taken for both encryption and MP3 encoding processes, whereas the offline case results are the time taken for all partial decoding, encryption, and re-encoding processes. Generally, the more the file is encrypted, the longer it takes for processing. However, even for p=100%, it only takes less than 2 seconds for 9-11 seconds long signals, and the time taken for p≤10% is found to be very short.

For authorized users, an MP3 file with nearly original quality can be successfully recovered by decrypting with the use of the correct decryption key and p. Figure 2.7(b) shows the time taken for both decoding and decryption processes. For all signals, it takes less than 2.5 seconds for p=100% and less than one second for p≤10%. Thus, it can be concluded that the proposed method is effective so that it can be applied in real time applications.

As for the security of the method, it depends on the security of the AES algorithm as discussed in Sect. 2.2.3. Although the AES/ECB mode is vulnerable to dictionary attacks, the ECB mode on MP3 bitstreams makes such attacks very hard due to large alphabet size. For instance, 128 bits per block result in 2^{128} possible input combinations, making the ECB block ciphers too large to be handled using today's dictionary attacks. In addition, since the pseudorandom sequence used to select the MP3 frames for encryption/decryption is generated based on the user-defined key, it is computationally impossible to generate the same sequence without knowledge of the key used. In addition, the fact that encrypted bits are diffused after decompression also enhances the security. If far stronger security is desired, the AES/CTR mode instead of the ECB can be applied as discussed in Sect. 2.2.

Table 2.3: Subjective measures of audio quality for the encrypted MP3s.

Music	Grading on Varying p					
	2%	5%	10%	20%	50%	100%
Pop	3.40	3.30	2.30	1.60	1.00	1.00
Jazz	3.30	3.20	2.20	1.40	1.00	1.00
Classical	3.50	3.30	2.50	1.60	1.00	1.00
Speech	3.40	3.30	2.30	1.60	1.00	1.00

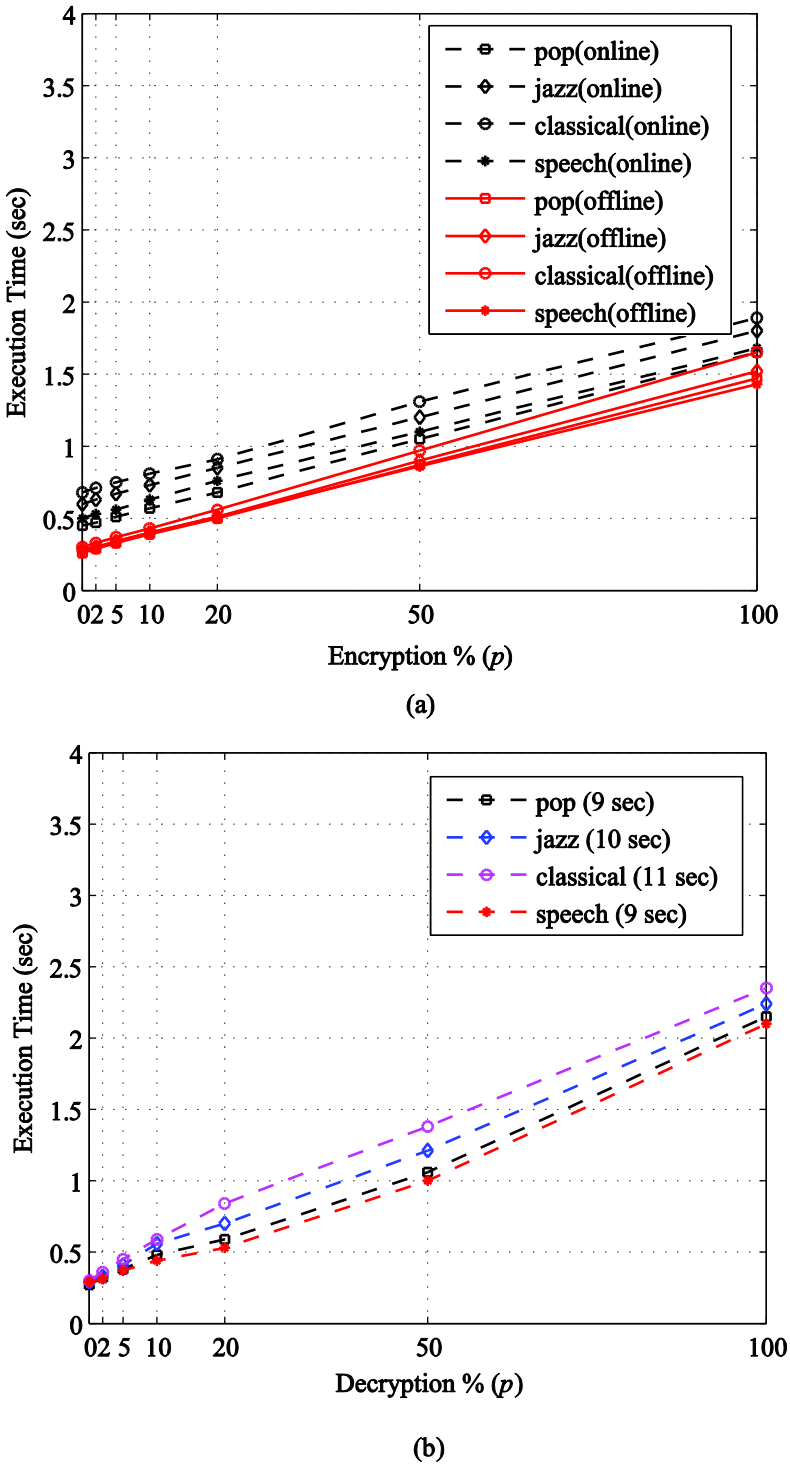


Figure 2.7: Execution time of the proposed method: (a) encryption (b) decryption.

Furthermore, we tried to correlate the encrypted-decoded MP3 with the unencrypted-decoded MP3. A correlation coefficient below 0.35 gives a security level comparable to that obtained by complete encryption of a frame [61-62]. In our system, for p=2-10% which we assume that it is effective enough to generate trial music, the resulting correlation coefficients vary from 0.45 to 0.62 for p=2%, 0.3 to 0.37 for p=5%, and 0.21 to 0.25 for p=10%. Thus in the

proposed method, the security level can be adjusted by varying *p* on the basis of the confidentiality level of the prospective customer to whom the trial music has to be distributed.

We also analyze the effect of the proposed method on compression ratio. Table 2.4 shows the size of the raw file (WAV) and the sizes of the compressed files (MP3s) with and without encryption as well. The AES algorithm has a tendency to increase the file size after encryption because the encrypted results are larger in value than the original ones. The original quantized values in the *big_values* partitions are [-8191, +8191] range but most are up to 15. The encrypted results are in the range of [0, 255] and thus most values change to larger ones after encryption and need more bits to code them. As shown in Table 2.4, there may be an increase in the file size up to 2KB for *p*=100%. However, it turns out that encrypting up to 10% of an MP3 file, which we suggest for generating trial music, has no effect on compression ratio.

2.3.4 Implementation necessity

In this section, we discuss the implementation requirement for the proposed partial encryption method for MP3. The proposed method needs to be incorporated into existing encoders or it is needed to develop new encoders that will embed it as a module. In our simulation, we use the LAME encoder to encode the MP3 files and apply our proposed encryption method as an external module.

As for the try-before-purchase model of DRM, the resulting encrypted MP3 files can be rendered by any standard MP3 players without need to decrypt as the usual unencrypted ones. For enjoying the full-quality, however, it is needed to decrypt the file with the authorized use of the encryption % (*p*) and the key. In order to provide those decryption parameters, we can append an additional file header that carries them at the beginning of an audio track without violating the MP3 standard [53-54]. That header needs to be protected from unauthorized accesses by using some public-key cryptographic algorithms. More strictly, although appending an additional header does not comply with the MPEG standard, any player implemented according to the standard will ignore that header while successfully playing the encrypted file. Without doubt, the file in the original quality can be recovered under the use of the correct decryption parameters and specific MP3 decoders.

Table 2.4: Effect of encryption on file size.

Music	File Size (WAV)	File Size (MP3 before Encryption)	File Size (MP3 after Encryption)	
			p=2-10%	*p*=100%
Pop	918 KB	78 KB	78 KB	80 KB
Jazz	1.85 MB	159 KB	159 KB	160 KB
Classical	2.01 MB	173 KB	173 KB	175 KB
Speech	1.7 MB	146 KB	146 KB	148 KB

2.4 Conclusion

This chapter discussed effective partial encryption methods for both uncompressed (WAV) and compressed (MP3) audio formats. The solution for the invalid amplitude problem was also presented. For partial encryption on MP3, this chapter presented how to choose the perceptually important parts for encryption in accordance with the HAS. Experimental results showed that the quality of the encrypted MP3s can be controllably degraded by adjusting the encryption percentage in which encrypting 2-10% of an MP3 file suffices to generate a preview-quality file. Those files can be played back by existing standard MP3 players without decryption. In addition, the proposed method can be carried out without extensive computation and significant impact on compression efficiency.

Chapter 3

Transposition-Based Audio Scrambling Methods in Time Domain

With the aim of possibly applying together with data hiding methods, this chapter discusses two transposition-based audio scrambling methods developed in the time domain: one is based on the pre-order traversal of a complete binary tree and the other on a PRNG called Mersenne Twister. The evaluation results of their scrambling effect, time and space complexity, and cryptographic security are also discussed.

3.1 Introduction

As discussed in Sect. 1.2.2, usual encryption methods [61, 63-64] are not appropriate for using together with the data hiding process because they cause the operational conflict problem. That problem can be avoided on the condition that those encryption methods provide the homomorphic property. Exactly speaking, homomorphic encryption is a form of encryption that allows computations to be carried out on the ciphertext and generates an encrypted result which, when decrypted, matches the result of those computations performed on the plaintext. Subramanyam and Emmanuel [61] proposed a combined scheme of homomorphic encryption and data hiding. In that system, the watermark information was embedded on the ciphertext and it could be detected in either the encrypted or decrypted domain. However, a cryptosystem which is unintentionally malleable can be subjected to attacks on this basis.

Datta and Gupta [63] proposed another combined scheme in which the conflict between encryption and data hiding operations is avoided by firstly segmenting the audio signal and then performing the methods on disjointed segments. In that system, the marked segments could not be encrypted and the encrypted segments could not carry the watermark information. Thus, the data embedding rate (the amount of watermarking bits that can be hidden in the signal) was reduced.

On the other hand, transposition-based audio scrambling methods are highly preferable for such a combined scheme. As discussed in Sect. 1.6, they do not affect the values of the audio contents. Thus, they provide operational transparency when using together with the data hiding process in which the watermark information embedded on the scrambled audio signal can be extracted from either the scrambled or descrambled signal.

A lot of transposition-based scrambling methods have been proposed in the literature and they can be classified as time domain (spatial domain) and transform domain methods.

Jayant et al. [16] presented comparison results of four speech scramblers which are based on sample permutation (S), block permutation (B), frequency inversion (F), and a combination of B and F, respectively. Their performances are thoroughly compared in terms of residual intelligibility, bandwidth expansion, and encoding delay. Among those scramblers, S achieves lower residual intelligibility than B and F due to the availability of the more possible permutations, however it increases encoding delay. In addition, the time domain scramblers like S and B are simpler to perform than F because the scrambling process can be directly done on the time domain samples. However, in terms of the bandwidth, F is more efficient because it ensures the bandwidth keep unchanged based on the frequency bands scrambled.

Chen and Hu [17] proposed two audio scrambling methods in the time domain in which the audio samples are reordered in accordance with the indices generated on the basis of a key and on the basis of the in-order traversal scrambling transformation, respectively. Then, the methods were combined by introducing a new parameter. In comparison with the individual methods, it was stated that the combined scheme strengthens the security even though there was no specific analysis on how much it was strengthened. In addition, according to the reproduction results, it is found out that the methods are relatively slow to execute. They take about half of the duration of the music file to scramble/descramble.

In the transform domain approaches, scrambling methods based on Fibonacci Transformation [49], digital chaotic ciphers [50], and Euler Transformation [51] have also been subjects of study.

In this chapter, we propose two transposition-based audio scrambling methods developed in the time domain together with the detailed evaluation results. The first method is based on the pre-order traversal of a complete binary tree. That method is effective in terms of time and space complexity and scrambling effect; however, its cryptographic security is not strong enough because there is no secret key control in that method [65]. According to Kerckhoffs's principle, the security of a cryptosystem should depend solely on the secrecy of the key and the private randomizer, not on the algorithm. Thus, the second method which is cryptographically more secure is proposed. That method is based on a PRNG called Mersenne Twister in which the state of the PRNG is controlled via a secret key [66].

3.2 Audio scrambling method based on the pre-order traversal of a complete binary tree

This section presents the proposed method that is based on the pre-order traversal of a complete binary tree. It is faster to execute than the in-order traversal based method proposed in [17].

First, let us introduce the traversal methods of a complete binary tree. A complete binary tree can be traversed in either breadth-first or depth-first traversal. Most simply, the breadth-first traverses a tree by visiting the node closest to the root it has not already visited (first child, then second child before grandchildren) whereas the depth-first traverses a tree by recursively visiting each node in the left and right subtrees of the root (first child, then grandchildren before second

child). The depth-first traversal can further be classified as pre-order, in-order, and post-order traversals in accordance with the position of the root with regard to the right and left nodes.

- Pre-order visits the root, traverses the left subtree, and then traverses the right subtree.
- In-order traverses the left subtree, visits the root, and then traverses the right subtree.
- Post-order traverses the left subtree, traverses the right subtree, and finally visits the root.

If we consider the result of the breadth-first traversal as an original data array/sequence, the results of the depth-first traversals will be like scrambling that sequence. As an example, assume that $S = [s_0, s_1, s_2, s_3, s_4]$ is an array of audio samples. That array can be represented as a complete binary tree by assuming the first sample as root node and filling the samples starting from the left in both branches, as shown in Fig. 3.1. As the breadth-first traverses a tree level-by-level, its traversal result is exactly the same as the given array, as shown in Fig. 3.2. As for the depth-first traversals, the pre-order traverses each and every subtree in root-left-right order which yields the result shown in Fig. 3.3. Similarly, the in-order (left-root-right) and the post-order (left-right-root) traversal results are shown in Fig. 3.4 and Fig. 3.5 respectively. All of the depth-first traversal results are different from *S*. Note that a tree is always traversed starting from the root node; however, the arrows in Fig. 3.2 to Fig. 3.5 just show the order of the result nodes.

On the basis of the above feature, an audio scrambling method based on the pre-order traversal is proposed in this thesis. For the sake of simplicity, the method is hereinafter referred to as *pre*.

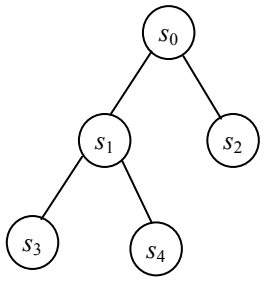


Figure 3.1: A tree representation of an array of audio samples (*S*).

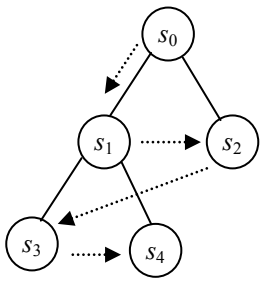


Figure 3.2: The breadth-first traversal result: $S' = [s_0, s_1, s_2, s_3, s_4]$.

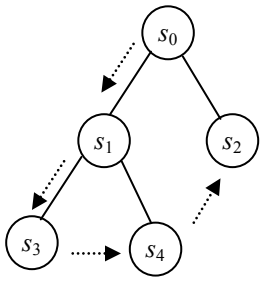


Figure 3.3: The pre-order traversal result: $S' = [s_0, s_1, s_3, s_4, s_2]$.

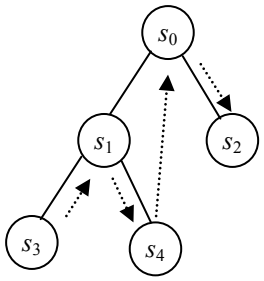


Figure 3.4: The in-order traversal result: $S' = [s_3, s_1, s_4, s_0, s_2]$.

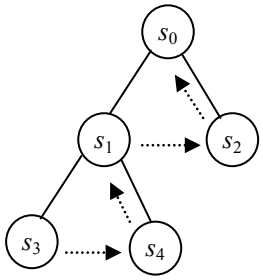


Figure 3.5: The post-order traversal result: $S' = [s_3, s_4, s_1, s_2, s_0]$.

3.2.1 Algorithmic detail

Algorithm 1 describes the algorithmic detail of *pre*. As in the above example, assume that each sample in an audio signal represents a node in a complete binary tree and that they are organized in breadth-first order. The complete binary tree in breadth-first order can be stored in an array instead of a linked list and thus the entire tree can be traversed by simply indexing like this: for a node at index i, its parent is at $\lfloor (i-1)/2 \rfloor$ (assuming the root has index 0) and its children are at $2*i+1$ (left child) and $2*i+2$ (right child). Avoiding the use of a pointer makes the algorithm highly efficient. In addition, a stack is used for storing some intermediate nodes for a later visit during traversal and it also increases the efficiency by avoiding recursion.

According to Algorithm 1, the sample at index 0 (the root) of the original audio is also the first sample/output in the scrambled audio. Then, if there is a sample at index 1 (left child), it is the next output, and the right child (if there is a sample at index 2) is stored in the stack for a later visit. If the left child is not the last node (if there are the samples left unvisited), it is assumed as the root of a subtree and the above steps of outputting the root, finding the left and right children, and outputting the left child are repeated. When there are no more left children to traverse down the tree, an element from the stack is retrieved as the next output. If that element is also the root of a subtree, that subtree is traversed again in the same manner mentioned above. This process continues until no more elements are in the stack. Figure 3.6 shows the step-by-step tracing of the *pre* on an example array according to Algorithm 1.

Algorithm 1 Pre-order Based Audio Scrambling Method

[1] Given an audio signal s with n samples $[s_0, s_1, ..., s_{n-1}]$, a scrambled audio signal $s' = [s_0', s_1', ..., s_{n-1}']$ is generated; Assume that each sample in s represents a node in a complete binary tree; Initialize *pos*=0 as index of root;
[2] **for** $i = 0, 1, ..., n-1$ **do**
[3] Construct the scrambled signal s' with samples from s indexed by *pos*:
$s_i' = s_{pos}$
[4] Calculate index of the left child:
lchild=2**pos*+1
[5] Calculate index of the right child:
rchild=2**pos*+2
[6] **if** rchild<n **then**
[7] Save the right child's index in stack:
push(rchild)
[8] Assign *pos*=lchild
[9] **else if** lchild<n **then**
[10] Assign *pos*=lchild
[11] **else if** stack is not empty **then**
[12] Output an element from the stack:
pos=pop()
[13] **end if**
[14] **end for**

Index	0 1 2 3 4 5 6 7	n=8
output root;	A B C D E F G H s ↓ A s'	Stack
pos=0;		
output lchild(1); save rchild(2) in stack;	A B s'	C
pos=1;		

output lchild(3); save rchild(4) in stack;	A B D	*s′* C E
pos=3;		
output lchild(7);	A B D H	*s′* C E
pos=7;		
no more lchild and rchild; output an element from stack;	A H D H E	*s′* C ~~E~~
pos=4;		
no more lchild and rchild; output an element from stack;	A H D H E C	*s′* ~~C~~
pos=2;		
output lchild(5); save rchild(6) in stack;	A H D H E C F	*s′* G
pos=5;		
no more lchild and rchild; output an element from stack;	A H D H E C F G	*s′* ~~G~~
pos=6;		

Figure 3.6: The step-by-step tracing of the *pre* on an example array.

3.2.2 Experimental results

In this section, the proposed method is thoroughly analysed in terms of its scrambling effect, time and space complexity, and security. We use the MATLAB software to simulate the method on a machine with an Intel(R) Core™ i5-2430M CPU @ 2.40GHz processor. The music items described in Table 3.1, which are sampled at 48-kHz and coded in 16-bit WAV format, are used in the experiments.

To verify the scrambling effect, the *pre* is applied on the “Pop” audio file and then the waveforms before and after scrambling are compared. Figures 3.7(a) and (b) show the waveforms of the original and scrambled audio signals respectively. From (b), it is difficult to guess the structure of the original audio and this shows that the *pre* has good scrambling effect. When listened to the scrambled audio, it has very low residual intelligibility.

The scrambling effect is also inspected in the frequency domain by applying the FFT algorithm on an analysis frame of 1,024 samples. From Fig. 3.7(c) and (d), which show the frequency spectra of the original and scrambled signals, we can see that the frequency spectrum becomes flat in the average sense after scrambling. This flatness ensures a decrease in residual intelligibility.

Let us consider the efficiency of the *pre*. For a single with n samples, it can be seen from Algorithm 1 that the *pre* needs the stack space plus $2n$ memory spaces for storing the original and scrambled samples. Thus, its space complexity is $O(n)$. Time complexity also depends on the length of the signal and it is $O(n)$ as each sample is processed sequentially and independently. More specifically, Table 3.2 shows the execution time results of the *pre*. For all signals, it takes less than one-third of the duration of the original signal and thus the *pre* is possibly applied in real time applications.

Table 3.1: Music items used in experiment.

Music Category	Track Name	Duration (sec)	Length (samples)
Pop	Bee Gees- Words	9	432651
Jazz	Dave Brubeck- Take Five	10	481244
Classical	Beethoven- Symphony No.5 in C Minor	18	867594

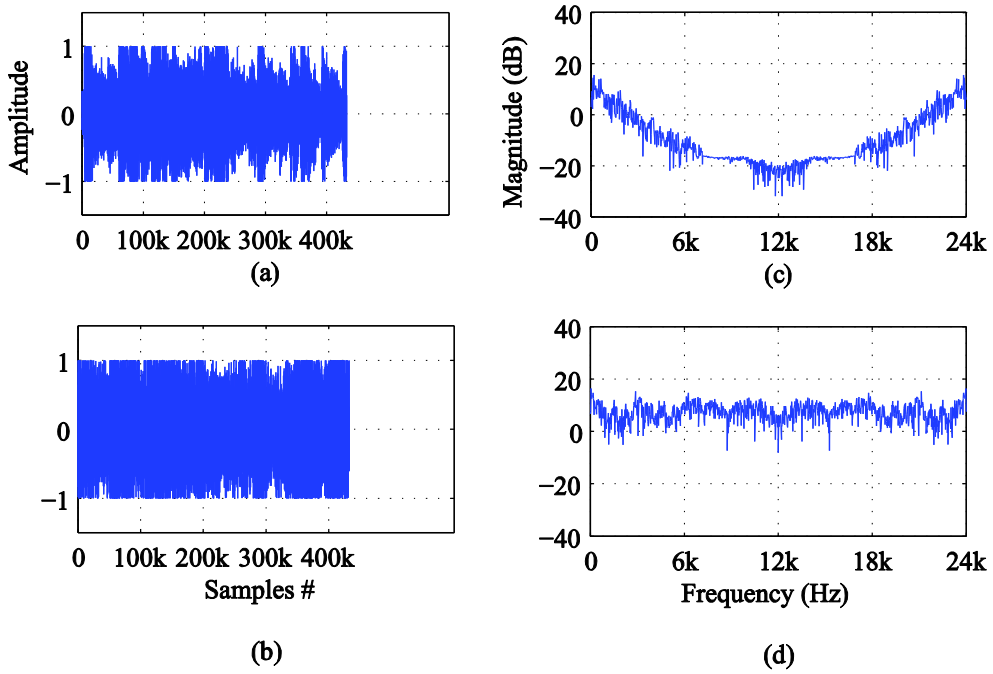


Figure 3.7: Waveforms of (a) the original signal and (b) the scrambled signal by *pre*; (c)-(d) their respective frequency spectra.

Table 3.2: Execution time results for the *pre*.

Category	Duration (sec)	Execution Time (sec)	
		Scrambling	Descrambling
Pop	9	1.99	2.70
Jazz	10	2.22	3.08
Classical	18	4.31	5.96

As for its cryptographic security, we need to firstly discuss the descrambling process. In transposition-based algorithms, descrambling can be done by tracing the scrambling procedure backward. For instance, if s= {*d, c, a, b*} is a sequence scrambled according to the scrambling indices {*4, 3, 1, 2*}, then it means that the first sample in s comes from the 4th position of the original sequence, the second one from the 3rd position, the third one from the 1st position, and the last one from the 2nd position, respectively. Then, by putting the samples back in their original positions, descrambling is successful. Thus, cryptographic security of the transposition-based scrambling methods like the *pre* depends on the concealment of the algorithm. If the algorithm is made public, there is no more security.

However, if the algorithms are in secrecy, an adversary needs to try a brute force attack to generate all possible permutations. As for the *pre*, the number of possible permutations is $n!$ where n is the length of the signal (the number of samples). For instance, the 9-second long "Pop" audio signal in Table 3.1 consists of 432,651 samples. The number of possible permutations for that signal is 432,651!, which is obviously impractical to try a brute force attack. Thus, the cryptographic security of the *pre* is strong enough if the algorithm is in secrecy.

With the aim of enhancing the cryptographic security of the *pre* in case the algorithm is made public, we briefly present a solution by introducing a new parameter k where a signal is divided into two parts at index k and the *pre* is then applied separately. For instance, for a signal s with n samples, the two parts $\{s_1, ..., s_k\}$ and $\{s_{k+1}, ..., s_n\}$ for $1 \le k \le n$, are separately scrambled. The idea is to enhance the security with the control of an unknown parameter. If an adversary wants to descramble the signal, he/she must try a brute force attack to guess k by running the *pre* $2n$-1 times with different k as the worst case. Thus, the longer the signal is, the possible k is more varied and the security is getting higher. To verify if the above solution has undesirable side effects although the security is strengthened, we also conducted some experiments on the music items in Table 3.1 for $k=n$, $3n/4$, $n/2$, $n/4$ where n is the length of the music item. Table 3.3 shows the scrambling effect for different k, which is measured in terms of the signal-to-noise ratio (SNR) and normalized cross-correlation (NCR) between the original and scrambled signals. For all signals, we can see from Table 3.3 that scrambling for different k achieves the same scrambling effect (almost the same SNRs and NCRs) like scrambling the whole signal. By comparing the results of Table 3.2 and Table 3.4, we can also see that there is no significant difference in execution time for different k although the tree is built in two parts.

Table 3.3: SNR and NCR of the scrambled signals for different k.

Category	Value of k	SNR (dB)	NCR
Pop	$n/4$	-2.99	0.03
	$n/2$	-3.03	0.03
	$3n/4$	-3.00	0.03
	n	-2.98	0.03
Jazz	$n/4$	-3.00	0.02
	$n/2$	-3.03	0.01
	$3n/4$	-3.01	0.01
	n	-2.99	0.01
Classical	$n/4$	-2.98	0.07
	$n/2$	-3.01	0.07
	$3n/4$	-3.07	0.07
	n	-3.01	0.07

Table 3.4: Execution time results for different *k*.

Category	Value of *k*	Execution Time (sec)	
		Scrambling	Descrambling
Pop	*n*/4	2.62	2.81
	n/2	2.54	2.80
	3*n*/4	2.27	2.82
Jazz	*n*/4	2.94	3.20
	n/2	2.44	3.21
	3*n*/4	2.49	3.22
Classical	*n*/4	5.42	5.74
	n/2	5.30	5.98
	3*n*/4	5.16	6.00

3.3 Audio scrambling method based on a pseudo-random number generator

As discussed in the previous section, cryptographic security of the *pre* depends on the secrecy of the algorithm. However, according to Kerckhoffs's principle, a cryptosystem should be secure even if everything about the system, except the key, is public knowledge. In other words, the security of a cryptosystem should depend solely on the secrecy of the key and the private randomizer. Thus, this section presents an audio scrambling method which is cryptographically more secure than the *pre*.

The proposed method is based on a pseudorandom number generator called Mersenne Twister (MT). Even though any kinds of PRNG can be applied in this proposal, the MT was chosen because it is by far the most widely used PRNG. The MT was developed in 1997 by Makoto Matsumoto and Takuji Nishimura [67]. It was designed to rectify most of the flaws found in older PRNGs and was the first PRNG to provide fast generation of high-quality pseudorandom integers. It is also the default PRNG for most software systems, e.g. Python [68], Ruby [69], PHP [70], and MATLAB [71]. The most commonly used version, MT19937, uses a 32-bit word length and has the following properties:

- A super astronomical period of 2^{19937}-1;
- 623 dimensional equidistribution up to 32-bit accuracy while consuming a working area of only 624 words;
- Coded in C language and faster than the other PRNGs;
- Passes numerous tests for statistical randomness including the Diehard tests [72-74] and most, but not all, of the stringent TestU01 Crush [75-78] randomness tests;

3.3.1 Algorithmic detail

The proposed method uses the MT to generate the indices for the scrambled list, on the basis of which the samples in the original audio signal are reordered. To ensure strong security, the MT is seeded with a random seed which is generated by hashing a user-defined key with a

cryptographic hash function (e.g. the message digest 5 (MD5) or the secure hash algorithm (SHA)). Hereafter, the method is simply referred to as the *AS-MT*.

The algorithmic detail of the *AS-MT* is as follows [66].

Input: An audio signal with n samples $[s_0, s_1, ..., s_{n-1}]$ and a user-defined key (*key*);

Step 1: The *key* is hashed by a cryptographic hash function. The resulting hash value is used as a *seed* for seeding the MT; the use of a cryptographic hash function ensures the randomness of the seed;

Step 2: Call the MT by passing the generated *seed* and n, and generate random integers (indices, z_k);

Step 3: On the basis of the z_k, the samples in the given audio signal are reordered;

$$s'_k = s_{z_k} \text{ for } k = 0, 1, ..., n\text{-}1. \tag{3.1}$$

Output: The scrambled audio signal with the reordered samples $[s'_0, s'_1, ..., s'_{n-1}]$;

Below is the procedure of the MT (Step 2) in detail.

- On the basis of the *seed*, initialize the generator in an array of length j $[x_0, x_1, ..., x_{j-1}]$;
- For a word x with w-bit width,

$$x_{k+j} \coloneqq x_{k+m} \oplus \left(x_k^u \mid x_{k+1}^l\right)A, \; k = 0, 1, ..., n-1, \tag{3.2}$$

where $\oplus$ is *bitwise XOR*, | is *bitwise or*, x^u and x^l are upper u-bits and lower l-bits shifts of x, respectively; A is a twisted transformation matrix in rational normal form

$$A = R = \begin{bmatrix} 0 & I_{w-1} \\ a_{w-1} & \left(a_{w-2}, ..., 0\right) \end{bmatrix}, \tag{3.3}$$

so that

$$xA = \begin{cases} x >> 1 & x_0 = 0 \\ \left(x >> 1\right) \oplus a & x_0 = 1, \end{cases} \tag{3.4}$$

where I_{w-1} is an (w-1)*(w-1) identity matrix and >> is *bitwise right shift*.

To compensate for the reduced dimensionality of equidistribution because of the choice of A as rational normal form, the MT is cascaded with tempering transforms as follows:

$$y_k \coloneqq x_{k+j} \oplus \left(x_{k+j} >> u\right), k = 0, 1, ..., n-1, \tag{3.5}$$

$$y_k \coloneqq y_k \oplus \left(\left(y_k << s\right) \& b\right), \tag{3.6}$$

$$y_k \coloneqq y_k \oplus \left(\left(y_k << t\right) \& c\right), \tag{3.7}$$

$$z_k \coloneqq y_k \oplus \left(y_k >> l\right), \tag{3.8}$$

where << is *bitwise left shift*, & is *bitwise and*. The coefficients of MT19937 are: (w, j, m) = (32, 624, 397); a = 9908B0DF$_{16}$; (u, l) = (11, 18); (s, b, t, c) = (7, 9D2C5680$_{16}$, 15, EFC60000$_{16}$). Note that we do not change the above settings of the MT19937 and we only set the MT to make sure $z_k \leq len$ and does not duplicate.

3.3.2 Experimental results

Like the *pre* in Sect. 3.2, the *AS-MT* is evaluated in terms of its scrambling effect, time and space complexity, and cryptographic security by using the music items in Table 3.1.

To test and verify the scrambling effect, we apply the *AS-MT* with user-defined key of "*secret*" on the "Pop" audio file and compare the waveforms before and after scrambling. Figures 3.8(a) and (b) show the waveforms of the original and scrambled audio signals respectively. Comparing (a) and (b), we can see that the *AS-MT* completely destroys the signal to such an extent that the envelopes of the waveforms are completely different. The NCR between the original and scrambled signals is 0.01 and the SNR is -3.01. When listened to the scrambled audio, it sounds like a white noise with zero residual intelligibility. Thus, it can be concluded that the *AS-MT* has good scrambling effect.

Figures 3.8(c) and (d) show the frequency spectra of the original and scrambled signals generated by applying the FFT algorithm on the signals for an analysis frame of 1,024 samples. Like in the *pre*, we can see that the frequency spectra become flat in the average sense after scrambling. This flatness ensures a decrease in residual intelligibility.

As for the efficiency, the *AS-MT* consumes 624 memory spaces for storing the initial state of the generator in addition to $2n$ spaces for storing the samples before and after scrambling for a signal with n samples. Thus, its space complexity is $O(n)$. The time complexity is generally $O(n)$ as well. However, from Table 3.5 which shows the execution time results of the *AS-MT*, we can see that the *AS-MT* is very fast so that its execution time can be negligible. This occurs because the main MT algorithm (Step 2 of the *AS-MT*) is a MATLAB built-in function written in C language for fast execution. In that regard, the *AS-MT* is more preferable to the *pre* to be applied in real time applications.

As for the cryptographic security, unlike the *pre*, the security of the *AS-MT* depends on the secrecy of the key which is used to seed the MT, not the algorithm itself. Even if the algorithm is made public, one will find it difficult to descramble without knowing that key. If the algorithm is in secrecy, an adversary needs to try a brute force attack to generate all possible permutations. Like in the *pre*, the number of possible permutations is $n!$ where n is the length of the signal (the number of samples), which is obviously impractical to try a brute force attack.

3.4 Conclusion

This chapter proposed two low-complexity audio scrambling methods together with the detailed evaluation results of their performances. As both methods affect only the positions of the audio samples, they are appropriate to be applied together with data hiding. Experimental results showed that their scrambling effect and time and space complexity are good enough to be applied in real time applications. If the methods were compared, the *AS-MT* was far superior to the *pre* in all aspects, especially for execution time and cryptographic security. Thus, this chapter also presented a solution for how to improve the cryptographic security of the *pre*.

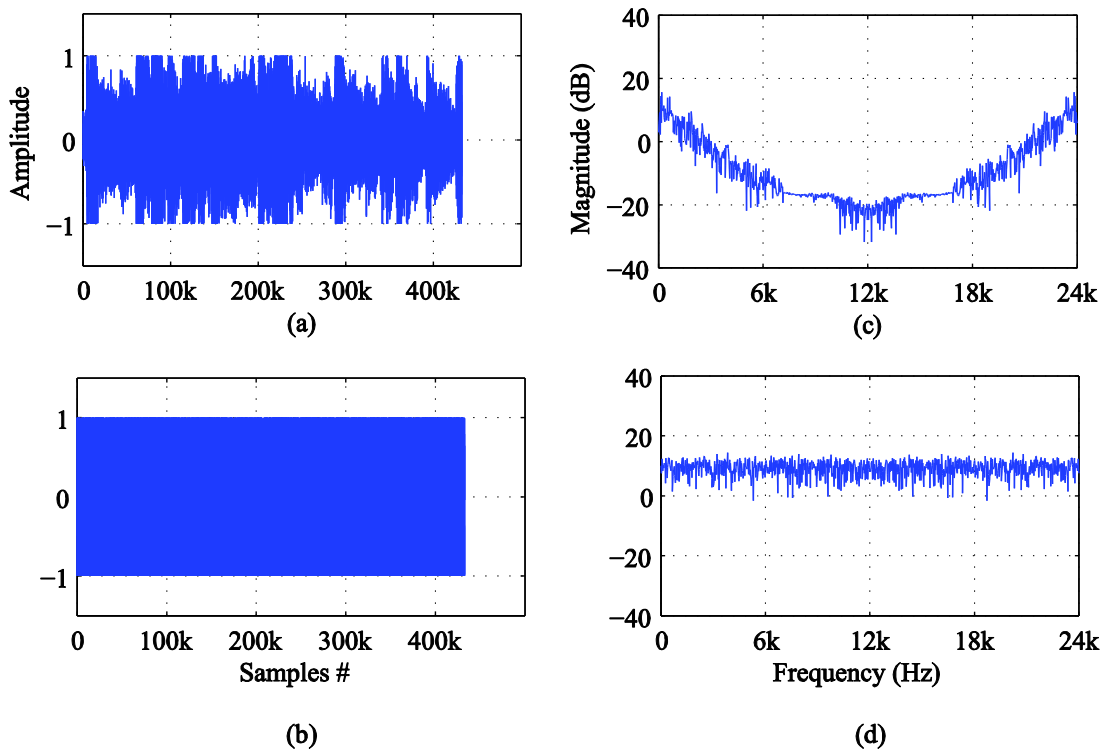


Figure 3.8: Waveforms of (a) the original signal and (b) the scrambled signal by *AS-MT*; (c)-(d) their respective frequency spectra.

Table 3.5: Execution time results for the *AS-MT*.

Category	Duration (sec)	Execution Time (sec)	
		Scrambling	Descrambling
Pop	9	0.07	0.07
Jazz	10	0.07	0.07
Classical	18	0.14	0.12

Chapter 4

Progressive Audio Scrambling Schemes in Wavelet Domain

This chapter discusses how the cryptographic security of the audio scrambling methods (the *pre* and the *AS-MT*) proposed in Chapter 3 can be strengthened in the wavelet domain. The two schemes of combining the *pre* and the *AS-MT* with the discrete wavelet transform (DWT) are proposed along with the detailed evaluation results. In addition to enhancing the security, the new schemes also achieve the progressive scrambling effect which is a desirable feature for DRM. This chapter also gives a detailed discussion on that matter.

4.1 Introduction

As discussed in Chapter 3, the pre-order based audio scrambling method has good scrambling effect and efficient time and space complexity as well. However, its cryptographic security is limited because it depends on the secrecy of the algorithm. Thus, in order to enhance its security, this chapter presents a new scheme in the wavelet domain. In this scheme, an audio signal is firstly decomposed into different frequency sub-bands by using the DWT with user-defined parameters. Then, scrambling is performed independently on each subband. Any user without knowledge of the wavelet decomposition parameters will not be able to successfully descramble the signal. The main idea is to enhance the cryptographic security with the control of unknown parameters.

In addition, the *pre* and the *AS-MT* in Chapter 3 performed scrambling on the whole signal. In those time-domain based methods, the scrambling effect on audio quality could not be controlled. Thus, the resulting scrambled signals became like noise and could not be used to realize the try-before-purchase model of DRM. In the new wavelet based scheme of the *pre*, the scrambling effect on audio quality can be controlled based on the sub-bands scrambled. This is called the progressive scrambling effect, which will be discussed in detail later in this chapter.

Thus, with the aim of realizing the try-before-purchase model of DRM, the *AS-MT* is also applied in the wavelet domain like the *pre*. No doubt, with the control of unknown wavelet parameters, the cryptographic security of the new scheme will be stronger than the security of the *AS-MT* in the time domain.

The main reason of using the DWT is to obtain the progressive (different frequency sub-bands) representation of an audio signal [66]. To decompose a signal into frequency sub-bands, we can also use another method such as the FFT filterbank instead of the wavelet transform; however, the wavelet representation is more efficient for analysis and reconstruction because of its multiresolution property [79-80]. More detail about the DWT is explained in the following section.

4.2 Discrete Wavelet Transform (DWT)

For better understanding of the proposed schemes, we briefly explain the DWT [81-86]. The DWT has been widely utilized in diversified applications including digital communications and image/speech/video compression [79-80, 86]. It was developed to overcome the shortcoming of the Short Time Fourier Transform (STFT) [79-80] which can be used to analyze non-stationary signals. Unlike the STFT that uses a constant window size for all frequencies, the DWT uses a window size that varies the frequency scale and is thus advantageous for analysis of the signals containing both discontinuities and smooth components.

The DWT decomposes the host audio signal into several multiresolution sub-bands [81]. That is, it provides good time resolution for high frequency sub-bands and good frequency resolution for low frequency sub-bands. The processes of DWT (decomposition) and IDWT (reconstruction) are shown in Fig. 4.1(a) and (b), respectively. In DWT, a discrete time-domain signal S is analysed by successive lowpass and highpass analysis filters, denoted by G_0 and H_0, respectively. Given S of length n, the DWT consists of $\log_2 n$ levels at most. At the first level, H_0 and G_0 followed by downsampling produce detail information (high frequency) dc_1 and coarse approximations (low frequency) ac_1, respectively. Downsampling by 2 doubles the frequency resolution as the uncertainty in frequency is reduced by half and halves the time resolution as the entire signal is now represented by only half of the number of samples. The second level repeats the same scheme, replacing S by ac_1 and yielding dc_2 and ac_2. This process

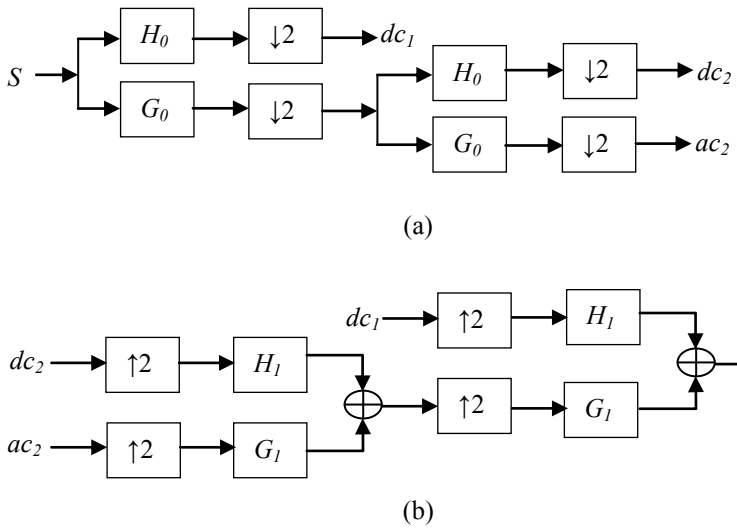


Figure 4.1: Two-level DWT: (a) decomposition (b) reconstruction.

is continued until the desired level is reached. The final decomposition result at level N consists of $[ac_N, dc_N, ..., dc_2, dc_1]$. With this approach, time resolution becomes arbitrarily good at high frequencies and frequency resolution becomes arbitrarily good at low frequencies.

In IDWT, the original signal can be successfully reconstructed from the wavelet coefficients. The *ac* and *dc* at every level are up-sampled by 2 and passed through the lowpass and highpass synthesis filters, denoted by G_1 and H_1 respectively, and then added. This process is continued through the same number of levels as in the decomposition process to obtain the original signal.

4.3 The proposed schemes

This section proposes two schemes of combining the *pre* and the *AS-MT* with the DWT with the aim of not only strengthening the security but also controllably degrading the audio quality. In these schemes, scrambling is performed independently on different frequency sub-bands of an audio signal in the wavelet domain. The detailed procedures of the proposed schemes are as follows [65-66]:

Step 1: An audio signal is wavelet decomposed. Then, the layers of wavelet coefficients [ac_N, dc_N, ..., dc_1], ordered in low-to-high frequency, are retrieved for N-level decomposition. Different layers have different frequency ranges.

Step 2: The coefficients in each layer are scrambled by:
scheme_1: considering not only the *pre* but also the in-/post-order based scrambling methods and applying one method randomly chosen out of the three, as shown in Fig. 4.2;
scheme_2: applying the *AS-MT* with different keys as shown in Fig. 4.3;

Step 3: Reconstruct the signal with the scrambled coefficients;

In order to descramble the signal successfully, the user must know the correct wavelet decomposition parameters such as the wavelet family and the decomposition level in addition to the correct method/key used in each layer. Decomposition with different wavelet parameters yields the layers of different coefficients and thus descrambling on them is similar to scrambling the signal in another way even if the correct method/key is used for each layer. Thus, without knowing the correct parameters, it is not possible to recover a high-quality audio signal. The experimental results in the following section prove that the proposed schemes can strengthen the cryptographic security of the *pre* and the *AS-MT* rather than in the time domain with the control of unknown parameters.

Moreover, scrambling on different layers degrades the audio quality differently. This feature ensures the progressive scrambling in which the audio signals with different quality levels can be generated by scrambling layer after layer. During descrambling, low-to-high quality can be progressively recovered by descrambling layer after layer as well, which will be discussed in detail in Sect. 4.4.2.

As the in-order and post-order based scrambling methods are applied together with the *pre* in *scheme_1*, their algorithmic details are described in Algorithms 2 and 3, respectively. Hereafter, we simply refer to them as *in* and *post* respectively.

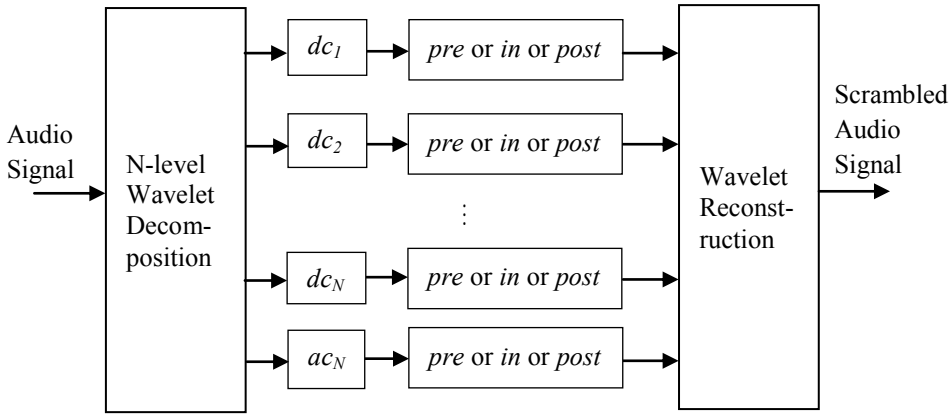


Figure 4.2: System flow of the *scheme_1*.

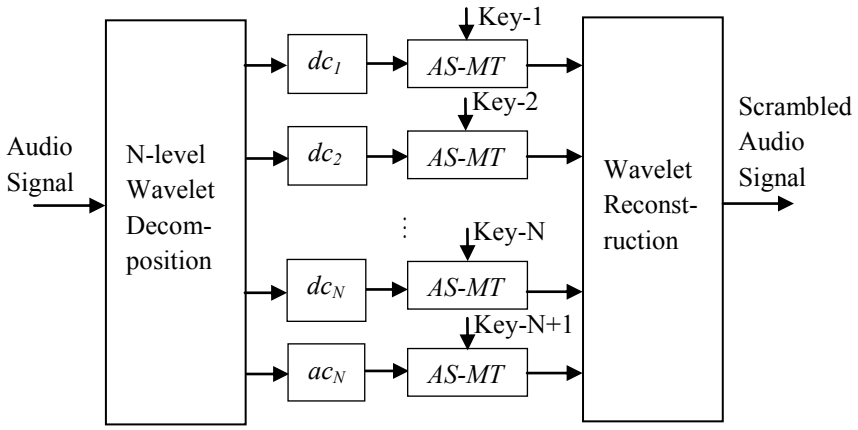


Figure 4.3: System flow of the *scheme_2*.

Like in the *pre*, assume that each sample in an audio signal represents a node in a complete binary tree and that they are organized in breadth-first order. For in-order, as described in Algorithm 2, the sample at index 0 (the root) of the original audio is stored in the stack for a later visit. Then, if it has a left child (sample at index 1) and that left child is also the root of a subtree, it will also be stored in the stack and the next left child (sample at index 3) is searched. These steps of finding the left child and storing it in the stack are repeated until there are no more left children. Then, an element from the stack is retrieved as the output/sample in the scrambled audio. The retrieved element may have a right child. If it has, assuming that right child is also the root of a subtree, all of the previous steps starting from step 1 of storing the root in the stack are repeated again. Otherwise, the steps of outputting an element from the stack and finding its right child are repeated. This process continues until no more elements are in the stack. Figure 4.4 shows the step-by-step tracing of Algorithm 2 on an example array.

For post-order, as described in Algorithm 3, the sample at index 0 (the root) of the original audio is stored in the stack. If it has both right and left children (samples at index 2 and index 1 respectively), firstly the right child and then the left child are stored in the stack. Then,

assuming the left child is the root of a subtree, the above steps are repeated. Otherwise, if the root has only the left child, it is stored in the stack and then an element from the stack is retrieved as the output/sample of the scrambled audio. If that retrieved element is a left child (note that all of the left children are at odd indices as the root is at index 0), it may have its neighbor, right child (subsequent element at the same level of a tree). If it has the subsequent right child, assume that this right child as the root of a subtree and repeat all of the previous steps starting from step 1 of storing the root in the stack. Otherwise, the steps of outputting an element from the stack and checking its right child neighbor are repeated. This process continues until no more elements are in the stack. Figure 4.5 shows the step-by-step tracing of Algorithm 3 on an example array.

Algorithm 2 In-order Based Audio Scrambling Method

[1] Given an audio signal s with n samples $[s_0, s_1, \ldots, s_{n-1}]$, a scrambled audio signal $s'=[s_0', s_1', \ldots, s_{n-1}']$ is generated; Assume that each sample in s represents a node in a complete binary tree; Initialize pos=0 as index of root and i=0;
[2] Save the root's index in stack:
push(pos)
[3] **while** true **do**
[4] Calculate index of the left child:
lchild=2**pos*+1
[5] **while** lchild<n **do**
[6] Save the left child's index in the stack:
push(lchild)
[7] Assign pos=lchild
[8] Calculate index of the left child:
lchild=2**pos*+1;
[9] **end while**
[10] Output an element from the stack:
pos=pop()
[11] Construct the scrambled signal s' with samples from s indexed by pos:
$s_i'=s_{pos}$
[12] Increment i by 1:
i++
[13] Calculate index of the right child:
rchild=2**pos*+2
[14] **while** stack is not empty **and** rchild>=n **do**
[15] Do Step 10-13
[16] **end while**
[17] **if** rchild<n **then**
[18] Save the right child's index in the stack:
push(rchild)
[19] Assign pos=rchild
[20] **else**
[21] break
[22] **end if**
[23] **end while**

Algorithm 3 Post-order Based Audio Scrambling Method

[1] Given an audio signal s with n samples $[s_0, s_1, \ldots, s_{n-1}]$, a scrambled audio signal $s'=[s_0', s_1', \ldots, s_{n-1}']$ is generated; Assume that each sample in s represents a node in a complete binary tree; Initialize *pos*=0 as index of root;
[2] Save the root's index in stack:
push(*pos*)
[3] **for** $i = 0, 1, \ldots, n-1$ **do**
[4] Calculate index of the left child:
lchild=2**pos*+1
[5] Calculate index of the right child:
rchild=2**pos*+2
[6] **while** rchild<*n* **do**
[7] Save the right child's index in the stack:
push(rchild)
[8] Save the left child's index in the stack:
push(lchild)
[9] Assign *pos*=lchild
[10] Calculate index of the left child:
lchild=2**pos*+1
[11] Calculate index of the right child:
rchild=2**pos*+2
[12] **end while**
[13] **if** lchild<*n* **then**
[14] Save the left child's index in the stack:
push(lchild)
[15] **end if**
[16] **while** stack is not empty **do**
[17] Output an element from the stack:
pos=pop()
[18] Construct the scrambled signal s' with samples from s indexed by *pos*:
$s_i'=s_{pos}$
[19] **if** !(*pos%2=0*) **and** *pos*+1<*n*/2 **then**
[20] *pos*++
[21] break
[22] **end if**
[23] **end while**
[24] **end for**

Index	0 1 2 3 4 5 6 7	*n*=8
save root in stack;	A B C D E F G H s s'	Stack A
pos=0;		
save lchild(1) in stack;	s'	A B
pos=1;		
save lchild(3) in stack;	s'	A B D
pos=3;		

save lchild(7) in stack;		*s′* A B D H
pos=7;		
no more lchild and output an element from stack;	H	*s′* A B D ~~H~~
pos=7;		
no rchild and output an element from stack;	H D	*s′* A B ~~D~~
pos=3;		
no rchild and output an element from stack;	H D B	*s′* A ~~B~~
pos=1;		
save rchild(4) in stack;	H D B	*s′* A E
pos=4;		
no more lchild and output an element from stack;	H D B E	*s′* A ~~E~~
pos=4;		
no rchild and output an element from stack;	H D B E A	*s′* ~~A~~
pos=0;		
save rchild(2) in stack;	H D B E A	*s′* C
pos=2;		
save lchild(5) in stack;	H D B E A	*s′* C F
pos=5;		
no more lchild and output an element from stack;	H D B E A F	*s′* C ~~F~~
pos=5;		
no rchild and output an element from stack;	H D B E A F C	*s′* ~~C~~
pos=2;		
save rchild(6) in stack;	H D B E A F C	*s′* G
pos=6;		
no more lchild and output an element from stack;	H D B E A F C G	*s′* ~~G~~
pos=6;		

Figure 4.4: The step-by-step tracing of the *in* on an example array.

Index	0 1 2 3 4 5 6 7	*n*=8
save root in stack;	A B C D E F G H *s* *s′*	Stack A
pos=0;		
save rchild(2) in stack; save lchild(1) in stack;	*s′*	A C B
pos=1;		
save rchild(4) in stack; save lchild(3) in stack;	*s′*	A C B E D
pos=3;		
save lchild(7) in stack;	*s′*	A C B E D H
pos=3;		
output an element from stack;	H *s′*	A C B E D ~~H~~
pos=7;		
output an element from stack;	H D *s′*	A C B E ~~D~~
pos=3; *pos*=4;		
output an element from stack;	H D E *s′*	A C B ~~E~~
pos=4;		
output an element from stack;	H D E B *s′*	A C ~~B~~
pos=1; *pos*=2;		
save rchild(6) in stack; save lchild(5) in stack;	H D E B *s′*	A C G F
pos=5;		
output an element from stack;	H D E B F *s′*	A C G ~~F~~
pos=5;		
output an element from stack;	H D E B F G *s′*	A C ~~G~~
pos=6;		
output an element from stack;	H D E B F G C *s′*	A ~~C~~
pos=2;		
output an element from stack;	H D E B F G C A *s′*	~~A~~
pos=0;		

Figure 4.5: The step-by-step tracing of the *post* on an example array.

4.4 Experimental results

In this section, by using the music items in Table 3.1, performances of the proposed schemes are thoroughly evaluated in terms of cryptographic security, effect of scrambling on audio quality, etc. In the experiment, an audio signal is two-level wavelet-decomposed by using the Daubechies wavelet family (db4). All of the obtained layers of coefficients are scrambled in such a way that [*pre, in, post*] in *scheme_1* and the *AS-MT* with user-defined keys of [*key1*, *key2*, *key3*] in *scheme_2* are applied on [ac_2, dc_2, dc_1] respectively. There is no specific reason in the choices of the parameters; it is just random.

4.4.1 Cryptographic security/anti-decryption capability

We mentioned that, in the proposed schemes, it is difficult to successfully descramble a signal without knowing the correct wavelet decomposition parameters and the method/key used for each layer.

To prove that fact, the "Classical" audio file is firstly scrambled by *scheme_1* and then descrambling is tried by using wrong wavelet parameters and wrong methods on the layers. Whether the signal is successfully recovered or not is determined by measuring the ODG, whose grading scales are described in Table 2.1.

Table 4.1 shows the ODG values resulting from descrambling with the wrong wavelet family where the other parameters such as the decomposition level and the methods/key used for each layer are correctly used. Different wavelet families use different filter-banks with different cutoff frequencies and hence it is impossible to generate the same wavelet coefficients. Descrambling on them is like scrambling the signal in another way. Thus, even though the other parameters except the wavelet family are correct, all of the ODG values are less than -3.9. It shows that the signal could not be successfully recovered and the resulting quality impairment was very annoying.

The results of descrambling with wrong decomposition levels are shown in Table 4.2. In this case, the signal is decomposed by using the correct wavelet family. As shown in the table, decomposing a signal at a different decomposition level yields different coefficients layers. For *N*-level decomposition, there are *N*+1 layers of coefficients. In addition, the sizes of the layers yielded by the use of different decomposition levels are also different, which cause different possible permutations on those layers. Thus, there is no way to successfully recover the signal with good quality. The resulting ODG values which are less than -3.8 show that the recovered audio quality is bad.

Then, the signal is decomposed at the correct decomposition level by using the correct wavelet family but the wrong methods are used for descrambling the coefficients in each layer. Table 4.3 shows the ODG values resulting from the use of wrong methods on the layers. Most ODG values are less than -3.8 and show that the resulting quality impairment is very annoying. We can also see that some ODG values are up to -2 when the correct methods are applied on two out of three layers. More interestingly, these values always result from the use of the correct method on the *ac*-part. As long as the wrong method is applied on the *ac*-part, the quality impairment is annoying. This is because the *ac*-part carries the lowest frequency coefficients of

the signal, to which the HAS is very sensitive, and changes to them have a strong effect on perceptual quality. However, even in that case, the ODG of -2 means that the quality impairment is still annoying.

Thus, if an adversary wants to recover a signal with good quality, he/she must try a brute force attack to guess the correct descrambling parameters. More precisely, he/she must at most try

$$126 \times \sum_{i=1}^{N} 3^{i+1} \text{ times in } \textit{scheme\_1} \text{ and} \tag{4.1}$$

$$126 \times \sum_{i=1}^{N} i+1 \text{ times in } \textit{scheme\_2}, \tag{4.2}$$

Table 4.1: ODG after descrambling with wrong wavelet family.

Wavelet Family	ODG
Haar	-3.906
Coiflets	-3.903
Symlets	-3.901
Discrete Meyer	-3.904
Biorthogonal	-3.906
Reverse Biorthogonal	-3.906

Table 4.2: ODG after descrambling with wrong decomposition level.

Level	Methods on Layers	ODG
2-level [a]	[pre, in, post]=>[ac_2, dc_2, dc_1] [a]	-0.186
1-level	[pre, in]=>[ac_1, dc_1]	-3.878
3-level	[pre, in, post, -]=> [ac_3, dc_3, dc_2, dc_1]	-3.893
4-level	[pre, in, post, -, -]=> [ac_4, dc_4, dc_3, dc_2, dc_1]	-3.895

[a]Correct decomposition level and correct choice of methods;

Table 4.3: ODG after descrambling with wrong choice of methods.

Methods on Layers [ac_2, dc_2, dc_1]	ODG
[pre, in, post] [b]	-0.186
[post, post, post]	-3.891
[post, pre, in]	-3.897
[post, pre, pre]	-3.896
[post, in, in]	-3.906
[in, pre, post]	-3.890
[in, post, pre]	-3.897
[in, in, in]	-3.906
[in, post, post]	-3.890
[in, pre, pre]	-3.897
[post, in, pre]	-3.906
[pre, post, in]	-3.843
[pre, pre, pre]	-3.842
[pre, in, in]	-2.021

Methods on Layers [ac_2, dc_2, dc_1]	ODG
[pre, post, post]	-3.781
[pre, post, pre]	-3.842
[pre, pre, post]	-3.783
[pre, in, pre]	-2.016
[pre, pre, in]	-3.843
[post, pre, post]	-3.891
[post, post, pre]	-3.896
[post, in, post]	-3.903
[post, post, in]	-3.897
[in, post, in]	-3.897
[in, in, post]	-3.903
[in, pre, in]	-3.897
[in, in, pre]	-3.906

[b]Correct choice of methods;

where 126 is a total of wavelet families (we only count those supported by the MATLAB [87]) and N is the decomposition level. The maximum possible value of N varies on the basis of the length of the signal and the wavelet family used. For example, for a 9-second long signal, N may be up to 18.

Considering the worst case scenario for that signal, an adversary must at most try $126\times\sum_{i=1}^{18}3^{i+1}=2.2\times10\text{^}11$ times for *scheme_1* and $126\times\sum_{i=1}^{18}i+1=23{,}814$ times for *scheme_2*. Let us assume for *scheme_1* that the time taken for scrambling each layer is the time taken by the *pre* (note that the *pre* is the fastest among the three methods). According to the simulation results on a PC with Intel ® Core™ i5-2430M 2.4 GHz Processor, each trial in the above worst case scenario takes 2.32 seconds and thus the total time taken for a brute force attack is nearly 16,045 years. For *scheme_2*, as the *AS-MT* is very fast to execute, the time taken is not a matter. However, the adversary has to guess the correct key for each trial and thus a total of 23,814 keys must be guessed. Thus, it can be concluded that the proposed DWT based schemes achieve the stronger cryptographic security than the methods in the time domain proposed in Chapter 3.

In the above experiment, a single wavelet family is used to decompose an audio signal. The cryptographic security of the schemes can be more strengthened by using multiple wavelet families. For instance, for 1-level decomposition that yields two layers, we can use two different wavelet families for yielding each layer. In this way, the level of difficulty for a brute force attack is higher since the adversary needs to guess the correct wavelet family for each layer (not for each level as in our experiment). However, this approach can increase the execution time because we need to decompose the signal multiple times.

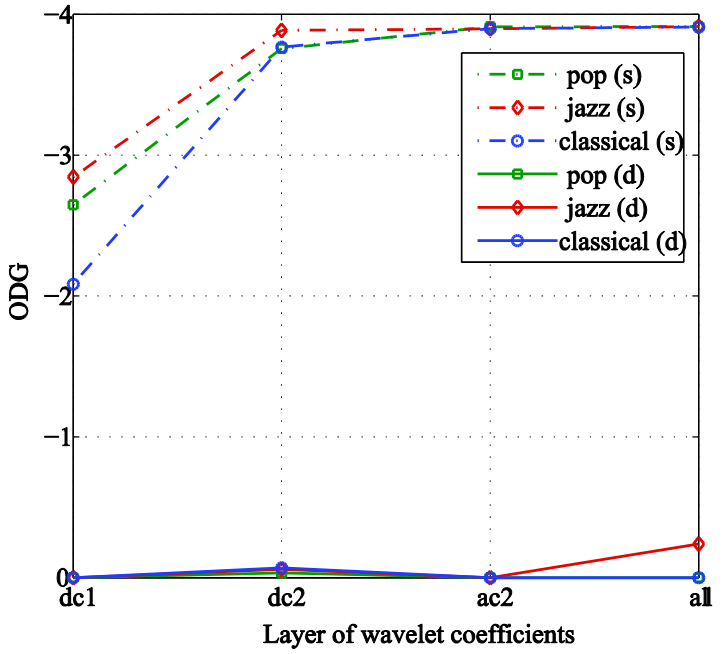
4.4.2 Scrambling effect

To test and verify the scrambling effect of the proposed schemes on the perceptual audio quality, the ODG values after scrambling and descrambling each layer as well as all layers are shown in Fig. 4.6.

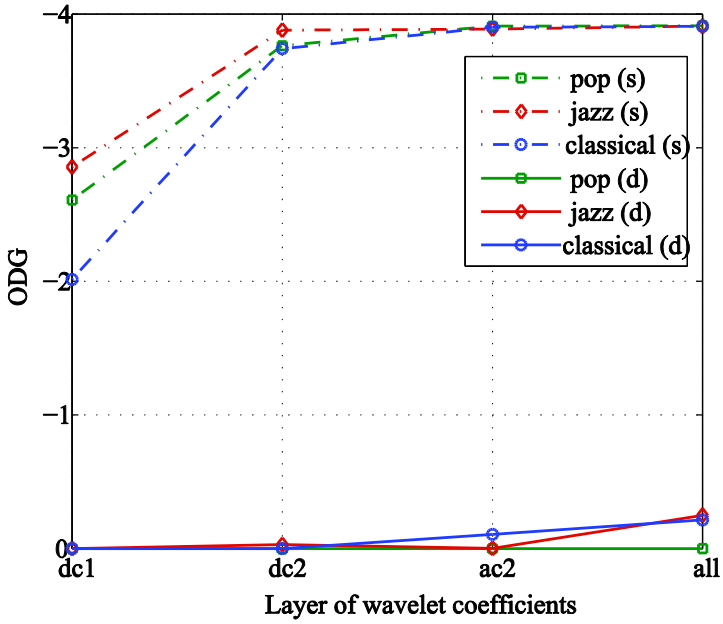
The ODG is calculated by using the original audio signals as reference. For both schemes, the ODGs after scrambling all layers are less than -3.9 for all test signals and show that the signals are severely distorted by scrambling. After descrambling, the ODG values are between 0 and -1 and indicate that the descrambling process can recover the signals back to their nearly original qualities. In addition, as discussed in Sect. 4.4.1, the coefficients in the *ac*-part are very important for perceptual quality and thus the ODGs after scrambling only ac_2 are almost comparable to those resulting from scrambling all layers.

Note here that the ODG values in Fig. 4.6 state a condition that we can call progressive scrambling: $SE(all) > SE(ac_2) > SE(dc_2) > SE(dc_1)$, where *SE* means the scrambling effect on audio quality. By progressively scrambling layer after layer, the resulting audio qualities are getting worse and worse. During descrambling, the audio signals with low to high quality can be progressively recovered by descrambling layer after layer. As for *scheme_2*, we can use a subset of keys for descrambling to obtain the lower quality audios, whereas nearly full-quality can be recovered by using all keys. The larger the decomposition level, the more layers are obtained and the more progressively we can control the scrambling degree.

The above feature also provides flexibility in the choice of layers to be scrambled on the basis of the system requirement: if the system requires high-level security, all layers are scrambled; if the system is not very security-centric but demands severe quality degradation, scrambling only the *ac*-part is enough; if the system demands low-quality audio files for realizing the try-before-purchase model of DRM, scrambling only the *dc*-parts, which yields fine-to-poor audio quality, is enough.



(a)



(b)

Figure 4.6: ODG after scrambling_(s)/descrambling_(d) on layer-by-layer basis: (a) *scheme_1* (b) *scheme_2*.

Table 4.4: Subjective measures of audio quality for the scrambled signals.

Music	SDG after Scrambling: Layer-by-Layer Basis			
	dc_1	dc_2	ac_2	all
Pop	4.3	2	1	1
Jazz	4.3	2	1	1
Classical	4.1	2	1	1

To verify the music quality after scrambling, we also conducted subjective tests. A total of five music files (the original and scrambled files for each layer) were distributed to six untrained listeners without mentioning anything about scrambling. Firstly, they were asked to pick out the original audio signal and all of them were able to correctly identify the original signal. Then, they were asked to feedback on the quality impairment of the scrambled signals in comparison with the original one. The grading results are shown in Table 4.4.

Although the listeners are not experts, they could all perceive the difference in the audio quality degradation based on the layer scrambled. Some could even identify the correct layer that was scrambled on the basis of the music quality they perceived. For all music items, all listeners confirmed that the quality impairment introduced by scrambling the *dc* parts, especially dc_2, is effective enough to use for trial music and scrambling the *ac*-part has almost exactly the same effect as scrambling all layers.

4.4.3 Execution time

Table 4.5 shows the execution time results of the new schemes for scrambling/descrambling each layer as well as all layers. Among the layers, the dc_2 is the largest in size and thus it takes the longest time for both scrambling and descrambling processes. Generally, for scrambling or descrambling all layers, the *scheme_1* takes at most $t/2$ seconds for a t-second long signal, and whereas the time taken for the *scheme_2* is short enough to be negligible. Considering this fact, it can be concluded that the *scheme_2* is more appropriate for real time applications. However, even for the *scheme_1*, if the underlying system is not very security-centric, we can significantly reduce execution time by scrambling only the *ac*-part that achieves the same scrambling effect as scrambling all layers.

4.4.4 Anti-attack capability

From Table 4.5, we can see that the proposed *scheme_2* is very fast to execute. In addition, since the scrambling process is carried out on different frequency sub-bands, the bandwidth of the signal can also be limited according to the sub-bands scrambled. This feature is important for speech scramblers which are applied to ensure privacy in real time speech transmission in telephone networks and radio communications [88-90]. Thus, with the possibility of applying the *scheme_2* in those applications, this section verifies that the *scheme_2* is fairly robust against channel distortions (common signal processing attacks) providing that the attacked signals are perfectly time-synchronized with the original ones.

The “Pop” audio signal in Table 3.1 is scrambled in accordance with the *scheme_2* in which all layers of the wavelet coefficients are scrambled by using the *AS-MT* with different

keys. Figure 4.7 shows the waveform of the original signal. For a noisy channel, the transmitted information may be partly or wholly lost during transmission. Thus, in the experiment, 60% of the scrambled signal is randomly cropped (cropped samples are replaced with 0) and then descrambling is tried. Figures 4.8(a) and (b) show the waveforms of the cropped signal and the recovered signal respectively. Comparing Fig. 4.8(b) with Fig. 4.7, we can see that most of the envelopes of the original audio still exist in the waveform recovered. It is because the lost samples are scattered throughout the entire signal and thus do not affect the basic structure of the audio. When we listened to the recovered audio, we can absolutely understand the music contents although some slight noise is introduced.

We also conducted noise addition attack in which additive white Gaussian noise is added to the scrambled signal by keeping SNR=10dB which indicates the noise is strong. Then, descrambling is tried. Waveforms of the attacked signal and the recovered signal are shown in Fig. 4.9(a) and (b) respectively. From Fig. 4.9(b), we can see that the envelopes of the recovered waveform are not clear anymore. When listened to the recovered audio, it sounds like the original audio which white noise is directly added to. However, we can still understand the music contents. We also tried to compress the scrambled signal by using the LAME MP3 encoder [59] at a bit-rate of 96 Kbps and then decoding and descrambling are tried. As shown in Fig. 4.10(b), the envelopes of the recovered waveforms are as clear as the original one. The ODG between the descrambled MP3-coded music and the original music is -1.75, which is fine quality. The audio quality is degraded due to data loss during compression but we can still understand the music. When listened to, it has better quality than the one white noise added to.

4.4.5 Performance comparison

A large number of image and speech scrambling methods have been proposed in the literature. However, there are few audio scrambling methods and even fewer well evaluated ones. Table 4.6 summarizes the performances of the proposed schemes including the methods proposed in Chapter 3 and shows their comparison results with the reference works [17] [52] [64].

Table 4.5: Execution time on layer-by-layer basis.

Music	Length	Layer	Exec. Time (sec)			
			Scrambling		Descrambling	
			scheme_1	scheme_2	scheme_1	scheme_2
Pop	9 sec	dc_1	2.49	0.12	2.33	0.12
		dc_2	1.27	0.11	1.26	0.11
		ac_2	0.83	0.11	0.83	0.11
		all	4.25	0.16	4.32	0.15
Jazz	10 sec	dc_1	2.68	0.14	2.70	0.14
		dc_2	1.42	0.12	1.45	0.12
		ac_2	0.96	0.12	0.95	0.12
		all	4.85	0.18	4.85	0.17
Classical	18 sec	dc_1	4.94	0.26	4.93	0.25
		dc_2	2.63	0.23	2.55	0.22
		ac_2	1.88	0.23	1.77	0.22
		all	8.67	0.34	8.71	0.30

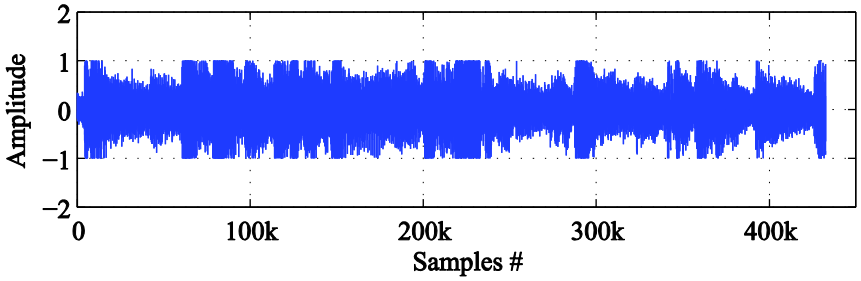


Figure 4.7: Waveform of the “Pop” audio signal.

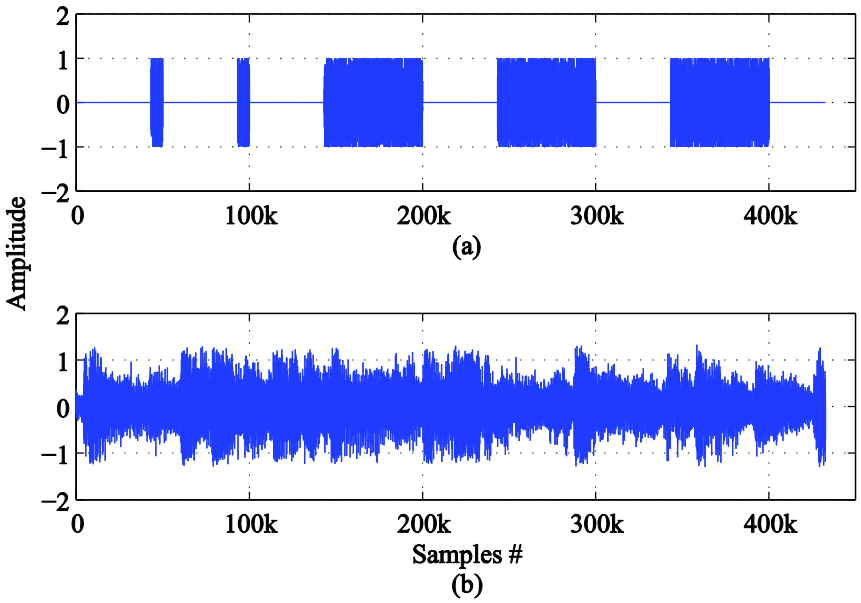


Figure 4.8: Waveforms of (a) the scrambled signal that is 60% randomly cropped and (b) the recovered signal.

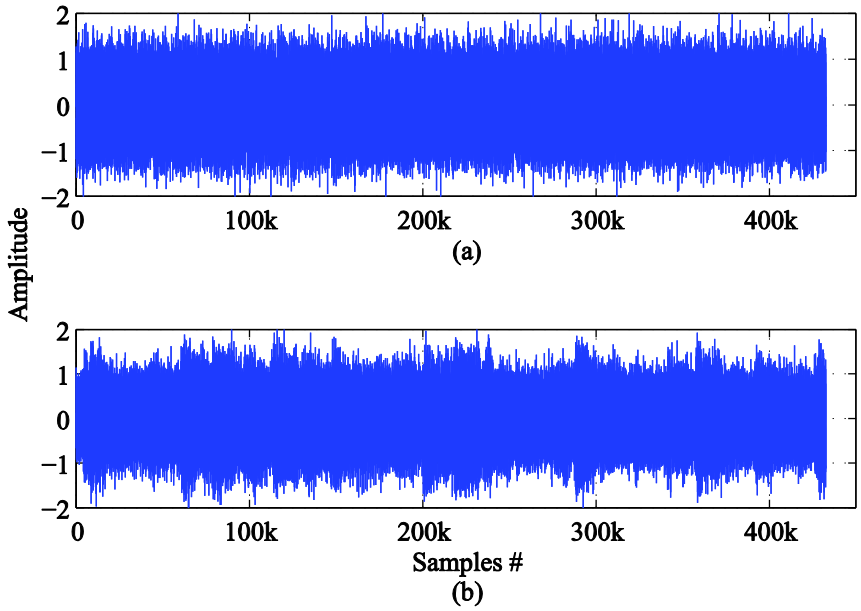


Figure 4.9: Waveforms of (a) the scrambled signal that is white noise added to and (b) the recovered signal.

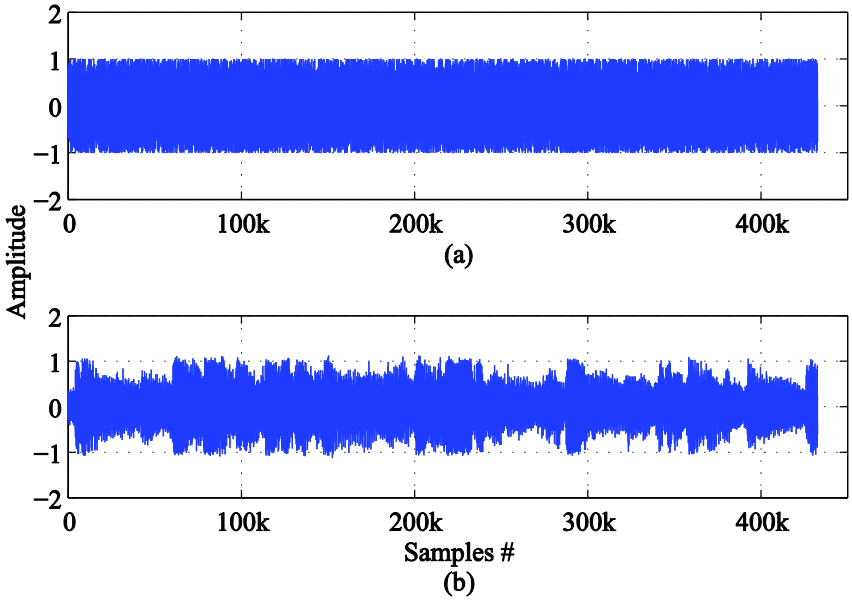


Figure 4.10: Waveforms of (a) the scrambled signal that is MP3 encoded and (b) the recovered signal.

Reference [17] proposed two audio scrambling methods in the time domain. One method reorders the audio samples in accordance with the indexing sequence generated based on a secret key and the other does the same thing in which the indexing sequence is generated based on the in-order traversal scrambling transformation. Reference [52] proposed a combined watermarking and scrambling scheme for MP3 in which the scalefactors in an MP3 frame are grouped in accordance with a secret key. Then, the scrambling and watermarking processes are carried out on the scalefactors of each group. The scrambling process here is swapping the scalefactors with the other ones in the same group. Reference [64] proposed an audio scrambling method in the compressed domain in which MP3 data are progressively scrambled by XOR-ing with the keys in a key table.

According to the results in Table 4.6, efficiencies (time and space complexity) of the methods are almost the same. Among the reference works, Reference [64] achieves the strongest security with the control of a key table for the price of increasing space complexity for storing that key table. However, the scrambling method in Reference [64] changes the values of the audio contents and thus causes operational conflict with the data hiding process.

As for the proposed methods, the *pre* and *scheme_1* have the comparable performances like Reference [17]. As for the *AS-MT* and *scheme_2*, they have better performances than Reference [17] in terms of security and scrambling effect (residual intelligibility). More specifically, according to Table 3.5 and Table 4.5, it can be seen that the *AS-MT* and *scheme_2* are very fast to execute, whereas the methods in Reference [17] are relatively slow (they take about $t/2$ seconds to scramble a t-second long signal). In comparison with Reference [64], the *AS-MT* and *scheme_2* achieve the security level as strong as [64]. In addition, they are more effective than Reference [64] in terms of space complexity and operational flexibility when combining with data hiding.

Table 4.6: Performance comparison with previous works.

Reference	Residual intelligibility	Time complexity	Space complexity	Security control	Conflict with data hiding
[52]	N/E	N/E	N/E	key	no
[17] (CDST)	very low	O(n)	O(n)	key	no
[17] (ITST)	very low	O(n)	O(n)	-	no
[64]	very low	O(n)	O(n)+key table	key table	yes
Our Contribution					
pre	very low	O(n)	O(n)	-	no
AS-MT	~ zero	O(n)	O(n)	key	no
scheme_1	very low	O(n)	O(n)	via wavelet	no
scheme_2	~ zero	O(n)	O(n)	keys + via wavelet	no

N/E means “no evaluation”; n is the number of samples containing in the signal;

4.5 Conclusion

With the aim of strengthening the cryptographic security of the methods proposed in Chapter 3, this chapter discussed two schemes that combine those methods and the DWT. In addition to providing strong security, the proposed schemes achieved progressive scrambling effect that enables the audio outputs with different quality levels to be generated by controlling the scrambling degree as required. This feature is very attractive for realizing the try-before-purchase model of DRM. In addition, the proposed schemes can also control the bandwidth of a signal to be unchanged based on the frequency sub-bands scrambled. Thus, they are possibly applied in real time speech scramblers, especially the *scheme_2* due to its fast execution time. Experimental results showed that the *scheme_2* is fairly robust against some common signal processing attacks.

Chapter 5

Conclusion

This thesis discussed the importance of audio encryption methods for providing confidential audio distribution in today's DRM environment. However, the naive encryption on audio data destroys compliance with the media format and thus the resulting encrypted file cannot be directly played back by existing standard music players. This thesis presented the problems of encryption on audio data and the solutions for those problems.

This thesis proposed a low-complexity partial encryption method for raw audio (WAV) together with a solution for the invalid amplitude problem. That problem makes the decryption process to be unsuccessful when the encrypted audio samples are beyond the valid audio amplitude range and thus they are clipped by the audio coder. This thesis also proposed an effective low-complexity partial encryption method for compressed audio (MP3). The idea of partial encryption is to protect the data by encrypting only perceptually important parts. During the MP3 encoding process, the input PCM signal transforms into several variants such as the frequency lines, quantized values, etc. Some of them are directly related to perceptibility and some are not. Thus, the effectiveness of partial encryption on MP3 depends on the choice of data to be encrypted. This thesis discussed how to effectively choose the perceptually important parts for encryption in accordance with the concept of the HAS. Experimental results showed that encrypting the whole MP3 audio file renders the audio signal meaningless while encrypting 2-10% of the file degrades the audio quality but not completely destroys the signal so it can be used as trial music for commercial purpose. That trial MP3 keeps compatibility with the standard so it can be rendered by any existing MP3 players without need to decrypt. Under the access of the correct decryption keys and specific MP3 players, full-quality MP3 can be successfully recovered. Experimental results also showed that the proposed method can be carried out without extensive computation, significant modification on the MP3 frame headers, and significant impact on compression efficiency of the MP3 encoders.

This thesis also presented two low-complexity transposition-based audio scrambling methods in the time domain. Audio scrambling methods are widely used for confidential distribution of audio data. They are more preferable to usual audio encryption to be used as pre- and post-processing of data hiding methods due to their nature of operational flexibility. They enable us to embed the watermark on the scrambled signal and to extract it from either the scrambled or descrambled version of the signal. Experimental results showed that the proposed

methods are very effective in terms of time and space complexity and scrambling effect although their cryptographic securities are limited. With the aim of enhancing their security, this thesis proposed two new schemes in the wavelet domain that combine those methods and the DWT. Instead of scrambling the whole signal, the new schemes firstly decompose an audio signal into different frequency sub-bands by using the DWT. Then the wavelet coefficients in each subband are separately scrambled. Experimental results showed that anyone without knowledge of the correct wavelet decomposition parameters and the correct method/key used for scrambling each subband will never be able to successfully descramble the signal. Those schemes also achieve the progressive scrambling effect that enables the audio outputs with different quality levels to be generated by controlling the scrambling degree on the basis of the system requirement: slightly distorted ones for the try-before-purchase model of the DRM systems and severely distorted ones for the systems with strong security needs. In addition, the new schemes are also fairly robust against common signal processing attacks. The resistances to those attacks are important for speech scramblers used in telecommunication and radio networks and hence the proposed schemes are possibly applied in those applications.

As a conclusion, this thesis presented various audio encryption methods to be used in today's DRM environment for both compressed and uncompressed audio data. This thesis gave a detailed discussion on 1) the problems of encryption on audio data and their solutions, 2) how to choose the data for partial encryption on MP3, 3) how to conduct encryption while keeping compliance with the media format, and 4) how to strengthen the security of audio scrambling. In addition to providing confidentiality in audio distribution, the proposed methods can also be used to realize the try-before-purchase model, which is one of the important business models of DRM. Thus, the proposals in this thesis strongly contribute to the development of efficient DRM systems.

One of the future works of this thesis is to develop an effective combined encryption and data hiding scheme. As discussed in Chapter 1, using encryption or data hiding alone is not sufficient for today's DRM systems. Encryption protects the content just before decryption, and whereas data hiding only ensures copyright protection. In the literature, there have been many audio encryption and data hiding methods proposed individually; however, there are very few combined schemes. Thus, developing an effective combined scheme is very important.

In addition, the concept of the Human Auditory System (HAS) is very important for developing effective audio encryption or audio data hiding methods. The HAS is more sensitive than the Human Visual System (HVS). Even small changes to the audio samples can be detected by the HAS and it is very difficult to develop a data hiding method that perfectly hides the secret information in the host audio signal. Thus, studying the HAS is also one of the future works of this thesis.

Bibliography

[1] B. J. Sheu, M. Ismail, M. Y. Wang, and R. H. Tsai, Multimedia Technology for Applications, Wiley-IEEE Press, June 1998.

[2] V. W. S. Chow, Multimedia Technology and Applications, Springer, 1997.

[3] P. Biddle, P. England, M. Peinado, and B. Willman, “The darknet and the future of content protection,” Digital Rights Management- Lecture Notes in Computer Science, vol.2770, pp.344-365, ISBN 978-3-540-40465-1, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[4] “Digital rights management final report,” European Committee for Standardization/ Information Society Standardization System (CEN/ISSS), September 2003.

[5] X. Zhang and R. Jain, “A survey of digital rights management technologies,” http://www.cse.wustl.edu/~jain/cse571-11/ftp/drm/index.html

[6] EC-Council, Computer Forensics: Investigating Network Intrusions and Cybercrime, Cengage Learning Inc., pp.9-26, 2009.

[7] A. K. Talukder and M. Chaitanya, Architecting Secure Software Systems, CRC Press, 2009.

[8] Electronic Frontier Foundation, “Digital rights management and copy protection schemes,” https://w2.eff.org/IP/DRM

[9] Adobe, “Adobe digital editions 1.0,” https://www.adobe.com/aboutadobe/pressroom/pressreleases/pdfs/200706/061907DigitalEditions.pdf

[10] https://en.wikipedia.org/wiki/FairPlay#cite_note-2

[11] R. Venkataramu, “Analysis and enhancement of Apple's FairPlay digital rights management,” Project Report, May 2007.

[12] Microsoft Developer Network, “Digital rights management,” https://msdn.microsoft.com/en-us/library/cc838192(VS.95).aspx

[13] https://en.wikipedia.org/wiki/Napster_(pay_service)

[14] https://en.wikipedia.org/wiki/Sony_Connect

[15] Electronic Frontier Foundation, “DRM,” 2012, https://www.eff.org/issues/drm

[16] N. Jayant, B. McDermott, S. Christensen, and A. Quinn, “A comparison of four methods for analog speech privacy,” IEEE Trans. on Communications, vol.COM-29, no.1, pp.18-23, January

1981.

[17] G. Chen and Q. Hu, “An audio scrambling method based on combination strategy,” IEEE Intl. Conf. on Computer Science and Information Technology, vol.5, pp.62-66, July 2010.

[18] H. Kaur and G. S. Sekhon, “A four level speech signal encryption algorithm,” Intl. Journal of Computer Science & Communication, vol.3, no.2, September 2012.

[19] S. Sharma, L. Kumar, and H. Sharma, “Encryption of an audio file on lower frequency band for secure communication,” Intl. Journal of Advanced Research in Computer Science and Software Engineering, vol.3, no.7, July 2013.

[20] S. Sharma, H. Sharma, and L. Kumar, “Power spectrum encryption and decryption of an audio file,” Intl. Journal of Scientific Research in Computer Science, vol.1, no.1, August 2013.

[21] V. Makwana and N. Parmar, “Encrypt an audio file using combine approach of transformation and cryptography,” Intl. Journal of Computer Science and Information Technologies, vol.5, no.3, 2014.

[22] Twe Ta Oo, T. Onoye, and K. Shin, “An approach to amplitude scaling partial encryption for compressed audio,” IEICE Technical Report, vol.113, no.135, pp.239-244, July 2013.

[23] Twe Ta Oo, T. Onoye, and K. Shin, “A partial encryption scheme for compressed audio based on amplitude scaling,” Proc. of Intl. Workshop on Smart Info-Media Systems in Asia (SISA), pp. 73-77, September 2013.

[24] K. K. Kothamasu and V. K. Sehgal, “An audio watermarking algorithm using encryption, transformations, and quantization,” Intl. Journal of Advances in Engineering Science and Technology, vol.2, no.1, 2013.

[25] W. Bender, D. Gruhl, and N. Morimoto, “Techniques for data hiding,” IBM Systems Journal, vol.35, no.3.4, pp.313-336, 1996.

[26] B. Mondal and T. Mandal, “A multilevel security scheme using chaos based encryption and steganography for secure audio communication,” Intl. Journal of Research in Engineering and Technology, vol.2, no.10, October 2013.

[27] G. Chen and X. Dong, From Chaos to Order: Methodologies, Perspectives and Applications, World Scientific, Singapore, 1998.

[28] IBM Corporation and Microsoft Corporation, “Multimedia programming interface and data specifications 1.0,” August 1991.
http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf

[29] Microsoft Corporation, “New multimedia data types and data techniques,” April 1994.

[30] Apple Computer Inc., “Audio interchange file format: AIFF version 1.3,” January 1989.

[31] “WavPack: hybrid lossless audio compression,” http://www.wavpack.com.

[32] K. Brandenburg, “MP3 and AAC explained,” Proc. of 17th AES Intl. Conf. on High-Quality Audio Coding, August 1999.

[33] R. Raissi, “The theory behind MP3,” December 2002.

http://www.mp3-tech.org/programmer/docs/mp3_theory.pdf

[34] K. Brandenburg and G. Stoll, "ISO MPEG-1 audio: a generic standard for coding of high-quality digital audio," Journal of the AES, vol.42, no.10, pp.780-792, October 1994.

[35] P. Noll, "MPEG digital audio coding," IEEE Signal Processing Magazine, vol.14, no.5, pp.59-81, September 1997.

[36] M. Biskup, "Error resilience in compressed data," PhD Dissertation, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw.
https://www.mimuw.edu.pl/wiadomosci/aktualnosci/doktoraty/pliki/marek_biskup/mb-praca.pdf

[37] J. Daemen and V. Rijmen, "AES proposal: Rijndael,"
http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf#page=1

[38] National Institute of Science and Technology (NIST), "Announcing the advanced encryption standard (AES)," Federal Information Processing Standards Publication 197, November 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[39] "National policy on the use of the Advanced Encryption Standard (AES) to protect national security systems and national security information," CNSS Policy No.15, Fact Sheet No.1, June 2003.

[40] O. Dunkelman, N. Keller, and A. Shamir, "Improved single-key attacks on 8-round AES-192 and AES-256," Proc. of 16th Intl. Conf. on the Theory and Application of Cryptology and Information Security, December 2010.

[41] H. Mala, M. Dakhilalian, V. Rijmen, and M. Modarres-Hashemi, "Improved impossible differential cryptanalysis of 7-round AES-128," Proc. of 11th Intl. Conf. on Cryptology in India, December 2010.

[42] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full AES," http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf

[43] J. Daemen and V. Rijmen, "Efficient block ciphers for smart-cards," Workshop on Smartcard Technology (Smartcard' 99), pp.29–36, USENIX Eds., 1999.

[44] D. Whiting, B. Schneier, and S. Bellovin, "AES key agility issues in high-speed IPsec implementations," Counterpane Internet Security, May 2000.
https://www.cs.columbia.edu/~smb/papers/AES-KeyAgile.pdf

[45] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, and S. Marchesin, "Efficient software implementation of AES on 32-bit platforms," Proc. of 4th Intl. Workshop on Cryptographic Hardware and Embedded Systems (CHES), August 2002.

[46] M. McLoone and J. V. McCanny, "High performance single-chip FPGA Rijndael algorithm implementations," Proc. of 3rd Intl. Workshop on Cryptographic Hardware and Embedded Systems (CHES), May 2001.

[47] V. Fischer and M. Drutarovsky, "Two methods of Rijndael implementation in reconfigurable hardware," Proc. of 3rd Intl. Workshop on Cryptographic Hardware and

Embedded Systems (CHES), May 2001.

[48] A. Dandalis, V. K. Prasanna, and J. D. P. Rolim, “An adaptive cryptographic engine for IPSec architectures,” IEEE Symposium on Field-Programmable Custom Computing Machines, pp.132–141, April 2000.

[49] N. Li, Y. H. Shang, and J. C. Zou, “An audio scrambling method based on Fibonacci transformation,” Journal of North China University of Technology, vol.16(3), pp.8-12, 2004.

[50] H. Y. Nie, C. Y. Zhu, and Q. Lu, “Audio encryption based on digital chaotic ciphers,” Signal Processing, vol.21(4A), 2005.

[51] H. L. Li and R. S. Ye, “Audio signal scrambling based on the Euler transformation,” Chinese Journal of Scientific Instrument, vol.28(4), pp.904-907, 2007.

[52] M. Steinebach, S. Zmudzinski, and T. Bolke, “Audio watermarking and partial encryption,” Proc. of SPIE, Security, Steganography, and Watermarking of Multimedia Contents VII, vol.5681, pp.779-788, March 2005.

[53] N. J. Thorwirth, P. Horvatic, R. Weis, and Jian Zhao, “Security methods for MP3 music delivery,” Proc. of the 34th Asilomar Conf. on IEEE Signals, Systems and Computers, vol.2, pp.1831-1835, November 2000.

[54] A. Servetti, C. Testa, and J. C. De Martin, “Frequency-selective partial encryption of compressed audio,” Proc. of IEEE Intl. Conf. on Acoustics, Speech, Signal Processing., vol.5, pp.668-671, April 2003.

[55] A. Torrubia and F. Mora, “Perceptual cryptography on MPEG-1 layer III bit-streams,” Proc. of IEEE Trans. on Consumer Electronics, vol.48, no.4, pp.1046-1050, November 2003.

[56] B. Gadanayak and C. Pradhan, “Selective encryption of MP3 compression,” Intl. Journal of Computer Applications, ICIST vol.1, pp.23-26, August 2011.

[57] Twe Ta Oo, T. Onoye, and K. Shin, “Partial encryption method that enhances MP3 security,” IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, vol. E98-A, no. 8, August 2015, to be published.

[58] “Method for objective measurements of perceived audio quality,” Rec. ITU-R BS.1387-1, 1998-2001.

[59] http://lame.sourceforge.net/links.php#OpenSource

[60] “General methods for the subjective assessment of sound quality,” Rec. ITU-R BS.1284-1, 1997-2003.

[61] A. V. Subramanyam and S. Emmanuel, “Audio watermarking in partially compressed-encrypted domain,” IEEE Intl. Conf. on Systems, Man, and Cybernetics, pp.2867-2872, October 2012.

[62] H. Wang, M. Hempel, D. Peng, W. Wang, H. Sharif, and H. H. Chen, “Index-based selective audio encryption for wireless multimedia sensor networks,” IEEE Trans. on Multimedia, vol. 12, no. 3, pp. 215-223, April 2010.

[63] K. Datta and I. S. Gupta, “Partial encryption and watermarking scheme for audio files with controlled degradation of quality,” Multimedia Tools and Applications, vol.64, no.3, pp.649-669, June 2013.

[64] W.-Q. Yan, W.-G. Fu, and M. S. Kankanhalli, “Progressive audio scrambling in compressed domain,” IEEE Trans. on Multimedia, vol.10, no.6, pp.960-968, October 2008.

[65] Twe Ta Oo and T. Onoye, “Progressive audio scrambling via complete binary tree’s traversal and wavelet transform,” Proc. of Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), December 2014.

[66] Twe Ta Oo and T. Onoye, “Progressive audio scrambling via wavelet transform,” Proc. of IEEE Asia Pacific Conf. on Circuits and Systems (APCCAS), pp.97-100, November 2014.

[67] M. Matsumoto and T. Nishimura, “Mersenne Twister: a 623-dimensionally equidistributed uniform pseudorandom number generator,” ACM Trans. on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation, 1998.

[68] “9.6 random - generate pseudo-random numbers,” Pythonv2.6.8 documentation, https://docs.python.org/release/2.6.8/library/random.html

[69] “Randomclass documentation,” Ruby 1.9.3 documentation, http://ruby-doc.org/core-1.9.3/Random.html

[70] “mt_srand,” PHP documentation, http://php.net/manual/en/function.mt-srand.php

[71] “Random number generator algorithms,” Documentation Center, MathWorks.

[72] http://en.wikipedia.org/wiki/Diehard_tests

[73] J. Soto, “Statistical testing of random number generators,” National Institute of Standards and Technology (NIST), http://csrc.nist.gov/groups/ST/toolkit/rng/documents/nissc-paper.pdf

[74] W. Rotz, E. Falk, D. Wood, and J. Mulrow, “A comparison of random number generators used in business,” Proc. of the Annual Meeting of the American Statistical Association, August 2001.

[75] http://en.wikipedia.org/wiki/TestU01

[76] B. D. McCULLOUGH, “Software reviews: a review of TestU01,” Journal of Applied Econometrics, pp.677-682, 2006.

[77] P. Leopardi, “Testing the tests: using random number generators to improve empirical tests,” http://maths-people.anu.edu.au/~leopardi/Leopardi-TestU01-paper-final.pdf

[78] P. L’Ecuyer and R. Simard, “TestU01: a C library for empirical testing of random number generators,” ACM Trans. on Mathematical Software, vol.33, no.4, article.22, August 2007.

[79] A. Mertins, Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms and Applications, John Wiley & Sons Ltd., 1999.

[80] R. X. Gao and R. Yan, Wavelets: Theory and Applications for Manufacturing, Springer, 2011.

[81] M. Vetterli and J. Kovacevic, Wavelets and Subband Coding, Prentice Hall, Englewood

Cliffs, NJ, 1995.

[82] http://en.wikipedia.org/wiki/Wavelet#Multiresolution_based_discrete_wavelet_transforms

[83] K. K. Shukla and A. K. Tiwari, Efficient Algorithms for Discrete Wavelet Transform, SpringerBriefs in Computer Science, 2013.

[84] M. Weeks and M. Bayoumi, "Discrete wavelet transform: architectures, design and performance issues," Journal of VLSI Signal Processing, 2003.

[85] C. E. Heil and D. F. Walnut, "Continuous and discrete wavelet transforms," Society for Industrial and Applied Mathematics, vol.31, no.4, pp.628-666, December 1989.

[86] J. Olkkonen, Discrete Wavelet Transforms – Theory and Applications, InTech, March 2011.

[87] http://www.mathworks.co.jp/jp/help/wavelet/ref/waveletfamilies.html

[88] S. Sridharan, E. Dawson, and B. M. Goldburg, "Speech encryption using discrete orthogonal transforms," Intl. Conf. on Acoustics, Speech, and Signal Processing, April 1990.

[89] S. Sridharan, E. Dawson, and B. Goldburg, "Fast Fourier transform based speech encryption system," IEEE Proc. of Communications, Speech and Vision, vol.138, no.3, pp.215-223, June 1991.

[90] D. S. Anjana and M. Kuriakose, "Frequency speech scrambler based on Hartley transform and OFDM algorithm," Intl. Journal of Computer Applications, vol.61, no.8, January 2013.