

Simple Audio Cryptography

Yusuf Adriansyah, 13507120¹
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
¹ if17120@students.itb.ac.id

Abstract— *Audio cryptography is not about audio encryption, or encryption in audio data. Instead, audio cryptography shares similar conception to visual cryptography. A plain data is split into two or more shares. Each single share does not convey any meaning, but when shares are combined together they will reveal the original plain data.*

In visual cryptography, an image – the "plaintext" here – is broken down to pixel level and its pixels are distributed among shares, with proper calculation. These shares now only consist of scattered pixels or black and white blocks, meaningless. Decryption process does not require computer aid, the receiver party simply piles up those meaningless pictures, then the original picture appears.

Similarly in audio cryptography, the "plaintext" is an audio data. This audio data is broken down to its samples and they are distributed among shares. If an eavesdropper plays one share in a media player, he or she will only hear meaningless hiss sound, or possibly annoying noise sound. But when shares are mixed together i.e. using an audio editor, the original audio comes back.

Index Terms—*audio cryptography, audio mixing, audio shares, digital audio, kriptografi audio.*

I. INTRODUCTION

Classical cipher algorithms deal with text data. Terms such "plaintext" and "ciphertext" emerge from here. As world moving to more modern techniques, modern cipher algorithms deal with binary data. These binaries can represent everything — texts, spreadsheets, images, multimedia, programs, etc. At this point, an audio data can be encrypted using any modern cipher, from simple XOR method to ElGamal.

The weak point is, these ciphers are breakable. Even that strong DES can be broken using brute force method and appropriate amount of patience. Then cryptographers made a new invention called *steganography*.

Steganography does not encrypt data, but *hides* it. The main reason is, cryptograms are suspicious. By removing this "suspicious" property of securing data, nobody would suspect the steganogram. Unfortunately, steganography is no longer secure. Steganalysis methods have been researched, and anyone can find the hidden message. Even the steganogram contains encrypted text, various cryptanalysis techniques can be employed to find the secret message. A tool such StegSecret (<http://stegsecret.sourceforge.net/>) has ability to detect LSB and EOF steganography methods inside a steganart.

Later, Moni Naor and Adi Shamir proposed a new

scheme in EuroCrypt 94 meeting^[1]. This was called "visual cryptography". Visual cryptography is interesting because it does not require computer aid in decryption process. The idea was to split the original image into two or more images called *share(s)*. As smallest units which build up an image are *pixels*, visual cryptography distributes these pixels among shares. The additive property of light is used. Also the shares are random and therefore suspect to a censor^[2]. Encryption process *does* require computer, but the decryption does not. According to the prior agreement between the sender (one who break the image to shares) and the recipients (called "participants"), there is a minimum number of participants having to gather around then stack the shares up, in that way only the original image will appear. Yes, the shares were printed in transparencies.

Similar conception can be applied in audio data. The big obstacle is, audio data is large (and so is video). Basically, any media data which has *temporal aspect* is large. In audio or video, whatever you hear or see is valid only for a relatively short moment. For example, consider you are listening to someone speaking "hello". At this second you hear he says the syllable "hé" [/hɛ/], but 0.2 second later he is already saying "llo" [/loʊ/] and does not saying "hé" anymore. This is contrasted to *still images*; an image will look exactly the same at any second you look at it.

That is why steganography in audio and video are hardly applied. An uncompressed audio for 1 second length in audioCD-quality consumes 44100×16 bits = 88200 bytes of data. If it is *stereo*, just double the number. Uncompressed still image having dimensions of 1 cm \times 1 cm with 72 dpi resolution consumes only 2523 bytes of data. Yes, it's only 2,5 KB. A compressed audio using MP3 compression at 128 kbps demands 16 KB per second, so 1-minute MP3 audio is about 960 KB. How do you find a steganogram container which capable to hold this 960 KB of data?

Audio cryptography also suffers the same problem, audio is large. Even if you managed to find a steganogram container large enough for your audio data, how do you send this steganogram? Transmission bandwidth in computer networks (or internet) is limited; therefore it is not efficient to send this extremely-large steganogram. Instead of *hiding* it inside a steganophon, let's *break* it into shares which could be *mixed* to reveal the original audio. Since each share will have the same size of data

compared to the original audio, producing 2 shares will produce $2 \times$ size of the original audio. This is much smaller than hiding it in a steganophon. But yes, we still need a computer for the decryption process.

II. SOUND

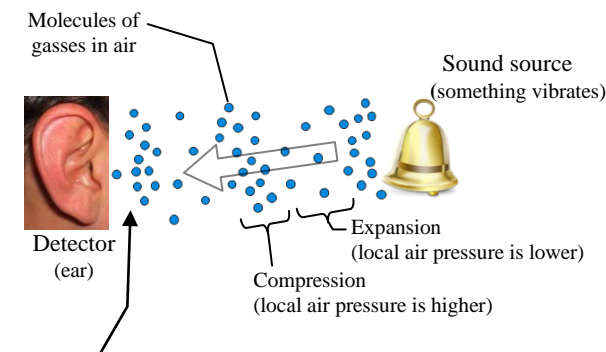
What is sound? Sound is a physical phenomenon caused by something that vibrates. To be honest, sound is *variations of pressure, travelling as wave*. Sound wave needs a *medium* to travel. This medium can be solid, liquid, or gas, such as the air. Sound wave cannot travel in vacuum. Thus, sound wave in air is variations of *air pressure*. If these variations of air pressure lie inside human hearing range (which has frequency limit from 20 Hz to 20 kHz), that sound is *audible* – which can be detected by human eardrums.

Sound wave is *longitudinal wave*. Longitudinal wave is a wave that its oscillation (vibration) direction is parallel to its travel direction. Another type of wave is transverse wave, which have its vibration direction is perpendicular to its travel direction. Therefore, longitudinal wave has *compressions* and *expansions*, while traverse wave has *peaks* and *valleys*.

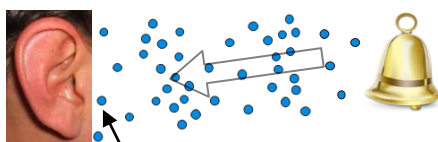
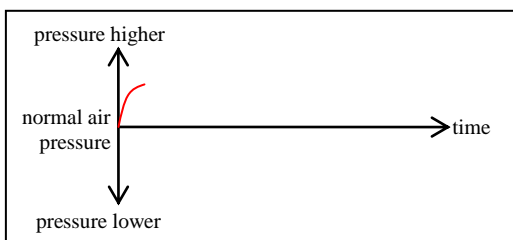
A. How sound represented

This section talks about analog sound.

Recall the definition that sound (in air) is variations of air pressure, and also sound is longitudinal wave which has compression and expansion. Hope this picture helps:



At $t = x$, ear detects local air pressure is higher.



At $t = x + \Delta x$, ear detects local air pressure is lower.

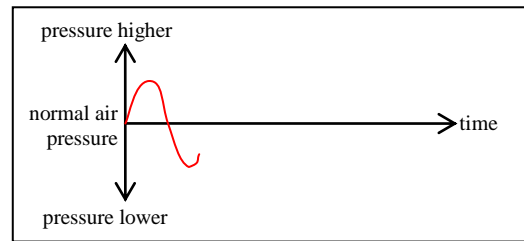


Figure 1. How sound represented

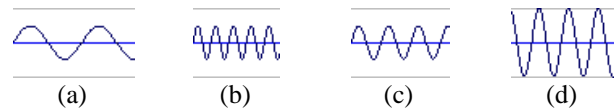


Figure 2. Properties of sound

- 2a. Low frequency, but same amplitude as 2b.
- 2b. High frequency, but same amplitude as 2a.
- 2c. Low amplitude, but same frequency as 2d.
- 2d. High amplitude, but same frequency as 2c.

High amplitude will produce a louder sound, and high frequency will produce a higher pitch.

B. How sound digitized

Analog sounds are recorded using a *microphone*. Microphone converts sound wave into electrical wave – but it is still analog. This analog electrical wave will be converted to digital audio inside an A/D (analog to digital converter) chip inside your *sound card*.

There are two methods to discretize analog signal into digital, they are called **PCM** (pulse code modulation) and **PDM** (pulse density modulation). The one which used to convert analog audio into digital audio is PCM. The PCM itself has three steps: sampling, quantizing, and coding.

Sampling — the time axis of an analog audio is sliced into many fixed intervals. For every beginning of each interval, we read the instantaneous value of the wave. How many samples do we need? That is *sample rate*, a number describing how many samples are taken every 1 second. According to the Nyquist theorem, if an analog signal contains frequency components up to f Hz, then the sampling rate should be at least $2f$ Hz. If the sampling rate is *exactly* $2f$ Hz, we call it critical sampling^[3]. Why $2f$? As you see, sound wave is oscillating up and down to represent one compression and one expansion of air pressure. One compression and one expansion is one wavelength, so a sound having frequency of f Hz has f positive samples (above the time axis) and f negative samples (under the time axis). Human voice rarely exceed 4 kHz, then telephone uses 8 kHz sample rate. Audio CD uses 44100 Hz sampling rate so that it can cover frequencies up to 22050 Hz, a little above the upper limit of human hearing.

Quantizing — there are various instantaneous values after sampling is done. For these values, we do *quantize* them to a fixed number of allowed values. How many quantization levels do we need? That is *bit depth*. Why it's called "bit" depth will be discussed in the next step, coding. There are 2 kinds of bit depth commonly used, 8-bit and 16-bit. Using 8-bit means you have 256 quantization levels available, numbered from 0 to 255 where value 128 is the central axis. Values above 128 are positive samples and below 128 are negative ones^[4].

Coding — is the last step. After quantization is done, computer stores every (quantized) value in all samples using binary representation. Since we use 8-bit or 16-bit quantization levels, computer stores each sample's value using one byte or two bytes, respectively. For wave format (those who end in .wav), 8-bit audio is stored as unsigned byte, 0 to 255. For 16-bit audio, they are stored as signed word ranging from -32768 to $+32767$.

C. How digital audio replayed

This section tells about the reverse process from previous section. Since human ear cannot understand digital signal (i.e. 10001011 11001000), we need to convert the digital audio back into analog audio. This has been done by D/AC (digital to analog converter) chip inside your sound card. D/AC will generate analog signal to drive the device called *speaker*.

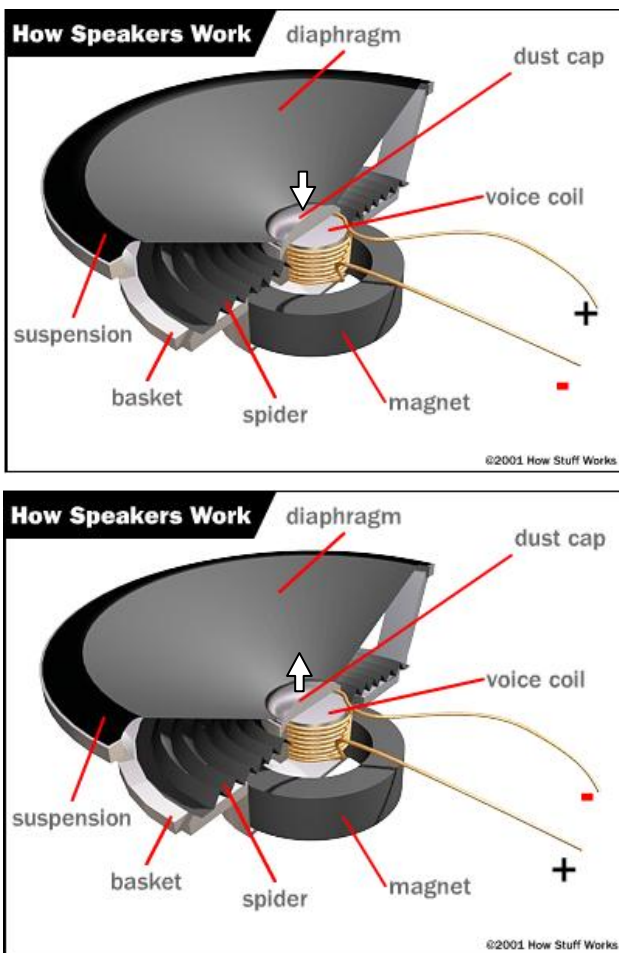


Figure 3. How speakers work. These pictures are actually an animation. Visit <http://electronics.howstuffworks.com/speaker5.htm> to see the full animation.

The coil will move the membrane back and forth, according to the polarity of voltage given into it. Positive sample will push the diaphragm up, and vice versa. This up-and-down movement occurs very fast thus vibrates surrounding air. This vibration propagates to our eardrum and our brain interprets it as a sound.

This also explains why we cannot have all-positive or all-negative samples in the digital audio file. If all sam-

ples are positive, coil in speaker only push the membrane forward without ever pulling it backward — no vibration, no sound.

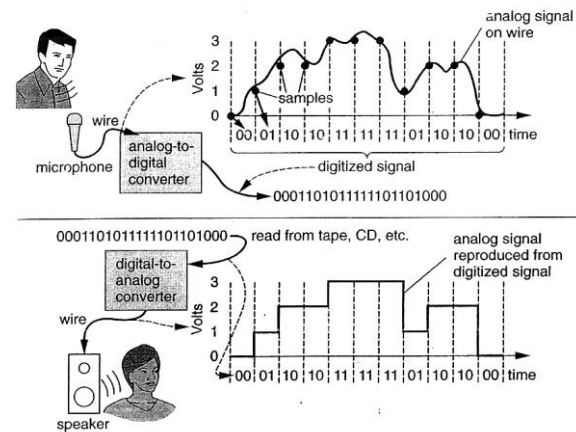


Figure 4. Analog audio being digitized using 4 quantization levels. This picture can be found on [VAH07] page 6.

D. Compressed audio

Uncompressed audio files are very large. Here is the formula:

$$\text{File size} = \text{SR} \times \text{BD} \times \text{L} \times \text{C} \quad (1)$$

where SR is the sample rate, BD is the bit depth, L is the length of audio (in seconds), and C is number of channels.

SR \times BD itself is called *bit rate*. For example, an audio CD always has 2 channels (that is stereo), 44100 Hz of sample rate, and 16-bit depth (2 bytes). One minute audio will be:

$$\begin{aligned} \text{File size} &= 44100 \times 2 \times 60 \times 2 \\ &= 10\,584\,000 \text{ bytes} \\ &\approx 10,1 \text{ MB.} \end{aligned}$$

That is why various audio compression algorithms have been researched. Most commonly used are MPEG audio layer III, a.k.a MP3. It can compress that 10 MB audio into about 1 MB. Other common alternatives are AAC (advanced audio coding) and WMA (windows media audio).

Equation (1) above only applies to uncompressed audio. For compressed ones, the formula is much simpler:

$$\text{File size} = \text{bit rate} \times \text{length of audio} \quad (2)$$

If the sound card only accepts uncompressed PCM, the CPU has to *decompress* the compressed audio first, before sending it to the sound card. When a compressed audio such as MP3 is being played on a media player, the media player will choose a suitable *codec* (compressor-decompressor) to decompress the audio. Every compression format has its own codec.

III. THE CRYPTOGRAPHY

A. Decryption

It's rather odd for discussing decryption first and en-

ryption later. However, an important theory used in decryption here is taken into consideration when encryption process is being done. Therefore decryption should be explained first.

Since ciphertexts take form as shares, they need to be combined in order to reveal the plaintext. As in visual cryptography we put one share on top of another, in audio cryptography we *mix* one share with the other share.

Audio mixing is just summing up the values of samples. For example, Alice is standing between two sound sources. One generates a $\sin x$ waveform, and the other generates a $\sin 2x$ waveform. When these two sources were made active at the same moment, Alice will hear a $\sin x + \sin 2x$ waveform. Depends on Alice's distance to source 1 and source 2, the mixed sound will be similar to $A \sin x + B \sin 2x$, where A and B are amplitudes factor being heard at Alice's place.

When a sample +7000 and +3000 are mixed, they becomes a +10000 sample. +5000 and -15000 result into value -10000. +7500 and -7500 will become zero — yes, they annihilate each other. If the left channel has certain samples and the right channel has the *exact negation value* from the left channel inside a stereo music, a strange sound effect (SFX) is heard. If you have a chance, try it. An example is a anime song titled "Naruto Ondo".

In audio mixing, an effect called *clipping* does occur. If one sample has a value of +10000 and another sample has a value of +15000, they sum up into one sample having value of +25000. This value is still allowed in 16-bit audio. But when one sample -25000 and -8000 are mixed, they resulted in a sample having value of -33000. This value is out of range in 16-bit audio. Audio mixer devices or the audio editor automatically cuts (*clips*) the sample at the maximum value allowed: -32768 for negative samples in 16-bit audio and +32767 for positive samples in 16-bit audio.

When mixing the audio shares, we must ensure that *no* clipping occurred. Clipping is *lossy*. Once a group of samples clipped, the original waveform of those samples will never possible to reconstruct, possibly makes loss of the original spoken word from the original audio. Thousands of samples having values like alternating maximum plus and minus values are really annoying loud noise. For this reason, we *lower* the volume (amplitude) of each shares first, before we mix them. A balanced mixed audio from audio 1 and audio 2 has the ratio 50% from audio 1 and 50% from audio 2.

B. Encryption

Encryption is done by breaking each sample's value into two values or more. These values must return to the original sample's value when they are summed up.

For instance, one sample having value of +17328 is broken into two samples, each of them has +8496 and +8832. The sample +8496 is stored in share number one, and +8832 is stored in share two. When these shares are mixed, the value +17328 comes back.

Note that there are many combinations that can yield +17328. Below are some of them:

- +17328 = (+8832) + (+8496)
- +17328 = (+8664) + (+8664)
- +17328 = (+10000) + (+7328)
- +17328 = (+19541) + (-2213)
- +17328 = (-10995) + (+28323)
- +17328 = (+32767) + (-15439)
- ... and so on.

So far we found three schemes we can use to break an audio into two shares:

1. *Break all samples into half of their original value.*
This is really a bad idea. Setting all samples' value half from the original means only setting their volume 50% from the original. Then each share is an exact copy from the original audio, only that they are quieter (has lower amplitude). Spoken words or music inside each share is still perceptible! Do not use this scheme.
2. *Break each sample into shares, with every share's sample has value lower than the original sample.*
This is what conveyed from this example above:

$$+17328 = (+8832) + (+8496)$$

Every sample is broken down into two values lower than its original values. Not necessarily half, but any combination that fullfills this formula

$$\begin{cases} x_1 + x_2 = x \\ |x_1| < |x| \\ |x_2| < |x| \end{cases} \quad (3)$$

can do. (x is the original sample's value.)

Now shares contain meaningless hiss sound. Since every shares has samples lower than the original audio, shares are quieter (have lower volume or amplitude). These could subject to *noise gate*. Noise gate, or noise reduction system, or hiss cut, or what ever the name is, removes low-volume hiss noise to make high-volume samples more clear to hear. Participants could encounter such situations if we transmit the shares using a radio station and their radio tuner has a noise reduction feature. We don't expect this happens, because original information can be lost. Then we can *double* the volume (amplitude) of each share, and then tell the participants to *cut down* the volume half before they mix the shares.

3. *Break each sample into a valid random possibility of combination.*

This idea was carried out by these examples above:

- +17328 = (+10000) + (+7328)
- +17328 = (+19541) + (-2213)
- +17328 = (-10995) + (+28323)
- +17328 = (+32767) + (-15439)

Every sample is broken down into any combination that satisfies this formula:

$$\begin{cases} x_1 + x_2 = x \\ \min \text{ value} \leq x_1, x_2 \leq \max \text{ value} \end{cases} \quad (4)$$

(Min value is -128 for 8-bit audio, and -32768 for 16-bit audio. Max values also apply.)

Using this third scheme, we need not to worry about noise gate and clipping any longer. The shares contain meaningless scrambled noise, and they are not always have low volume (amplitude). This scheme is the best.

IV. SAMPLE IMPLEMENTATION

I made an application in Visual Basic 2008 which does the encryption function. This application only accepts uncompressed audio (i.e. *.wav files), then breaks it into *two* shares, and saves the shares again as .wav files. These .wav share files are playable in any media player.

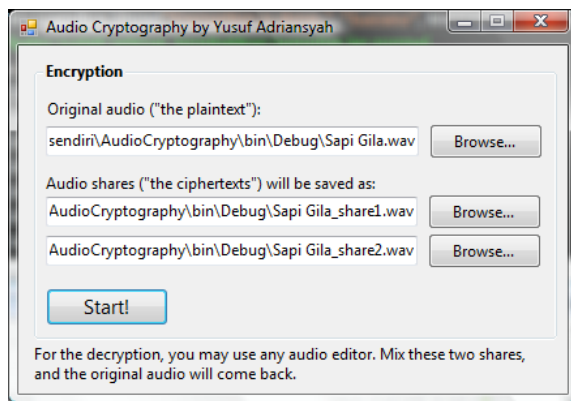


Figure 5. My implementation in VB

I use this algorithm to break samples into shares. Loop for each sample in the original audio file:

$$\begin{aligned} x &= \text{random}(0 \text{ to } z) \\ y &= \text{random}(0 \text{ to } 1) \\ s_1 &= sy + x \\ s_2 &= s(1 - y) - x \end{aligned}$$

where:

- s is the original sample,
- s_1 and s_2 are share one's and share two's sample,
- x is a random number ranging from 0 to z ,
- y is a random number ranging from 0 to 1,
- z is the maximum number permitted by the audio.

Maximum number permitted by audio depends on the situation. If the original audio is 8-bit, then $z = 127$ if s is positive, or -128 if s is negative. If 16-bit then $z = 32767$ if s is positive or -32768 otherwise. Also I have to guarantee that relationship $s = s_1 + s_2$ is still satisfied. And don't forget to take care of clipping, that is, s_1 and s_2 must be kept inside the range given by its bit depth.

The underlying core process is inside these codes. Yes this is Visual Basic, not C#. All comments (those who begin with a single quote ['']) are translated into English.

```
'variables declaration
Dim orijinal As WaveData
Dim s1 As WaveData
Dim s2 As WaveData
Dim result As Boolean
Dim asli As Short, rusak1 As Integer
Dim rusak2 As Integer
Dim temp1 As Single, temp2 As Short
Dim clipmax As Short, clipmin As Short
```

```
'load the audio
orijinal = New WaveData()
result = orijinal.OpenWAV(OrigAudio.Text)
If result = False Then
    MessageBox.Show( _
        "Unable to open the original audio.", "Error", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Exclamation)
    orijinal.Dispose()
    GoTo selesai
End If
'progress bar's upper limit
Progress.Maximum = orijinal.Length + 10
Progress.Value = 1
'determine clipping limits
If orijinal.BitDepth = WaveData.bitdepthtype._8bit _
Then
    clipmax = 127
    clipmin = -128
Else
    clipmax = 32767
    clipmin = -32768
End If
Progress.PerformStep()
'prepare two shares
s1 = New WaveData(orijinal.Channel, _
    orijinal.BitDepth, orijinal.SampleRate, _
    orijinal.Length)
Progress.PerformStep()
s2 = New WaveData(orijinal.Channel, _
    orijinal.BitDepth, orijinal.SampleRate, _
    orijinal.Length)
Progress.PerformStep()
'initialize the pseudo-random generator
Randomize()
Progress.PerformStep()
'start!
If orijinal.Channel = WaveData.channeltype.mono Then
    For i As Integer = 0 To orijinal.Length - 1
        asli = orijinal.GetSample(i, 0)
        If asli > 0 Then
            temp1 = Rnd() * clipmax
            temp2 = Int(temp1)
            temp1 = Rnd()
            rusak1 = Int(asli * temp1) + temp2
            'take care of the clipping
            If rusak1 > clipmax Then rusak1 = clipmax
            rusak2 = Int(asli * (1.0F - temp1)) - temp2
            If rusak2 > clipmax Then rusak2 = clipmax
        ElseIf asli < 0 Then
            temp1 = Rnd() * Math.Abs(clipmin + 1)
            temp1 *= -1
            temp2 = Fix(temp1)
            temp1 = Rnd()
            rusak1 = Int(asli * temp1) + temp2
            If rusak1 < clipmin Then rusak1 = clipmin
            rusak2 = Int(asli * (1.0F - temp1)) - temp2
            If rusak2 < clipmin Then rusak2 = clipmin
        End If
        s1.SetSample(i, 0) = rusak1
        s2.SetSample(i, 0) = rusak2
        Progress.PerformStep()
    Next i
ElseIf orijinal.Channel = _
WaveData.channeltype.stereo Then
    '-----8c snip-snip 8c-----
    'Codes for stereo files are exactly the same.
    'the only difference is that the encryption is done
    'on both channels (left and right).
    'To save space, codes are not shown.
    '-----8c snip-snip 8c-----
End If
'here encryption is done, save all shares.
s1.SaveWAV(Share1.Text)
s2.SaveWAV(Share2.Text)
```

Prior to develop this application, first I made a class called "WaveData", which is used in this application. WaveData provides basic operation for wave audio such as loading from and saving to .wav files, set volume, set mute, mixing, and getting or setting individual samples. This paragraph is written to clarify codes shown in previous page, which include "WaveData".

The variables' names are still in Indonesian. The integer *asli* corresponds to *s*, the original sample. *Rusak1* and *Rusak2* are s_1 and s_2 respectively. But variables named *s1* and *s2* represent WaveData objects, *not* the samples, for which I will use their "saveWAV" function later. Next, *Temp1* and *Temp2* are temporary variables used for many purposes, such as defining *x* and *y* in my algorithm. *Clipmin* and *clipmax* defines the clipping limit permitted (*z*), and their values depend on *orijinal's* bit depth. At last, names such as "OrigAudio", "Share1", "Share2", are the names for text box controls inside my main window form.

Following this way, each share now features annoying noise. Unfortunately, after some trial using some .wav test files, the original audio is still *audible* inside each share. This algorithm does not completely obfuscate the original audio.

Now I will describe how to restore the audio from these two shares. Open all shares in your favourite audio editor. For the time being, I use Country.wav as a test file. After encryption is done, two files named Country_share1.wav and Country_share2.wav were created. Screenshots here are for Adobe® Soundbooth CS4:

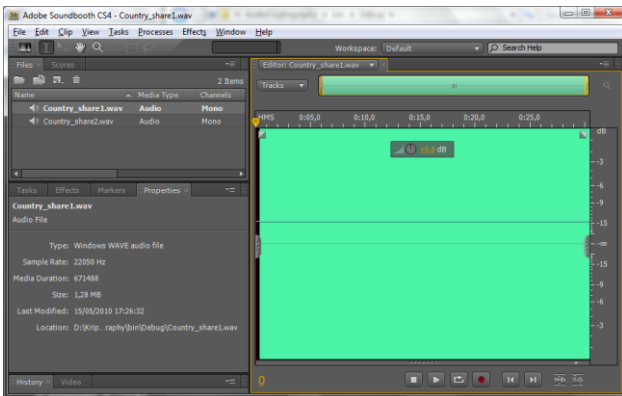


Figure 6. Open Country_share1.wav and Country_share2.wav in Adobe Soundbooth.

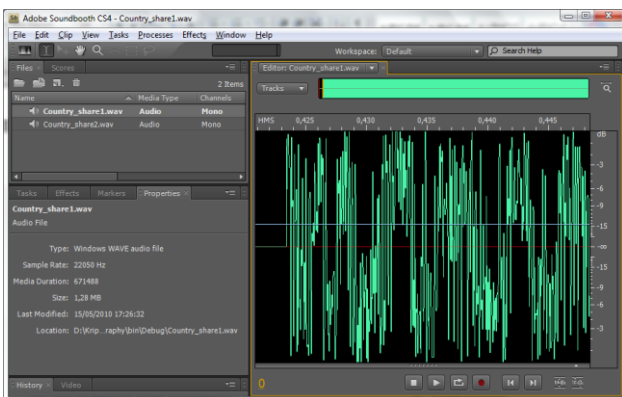


Figure 7. Probably you see a block of green color like in figure 6. But if you deep-zoom it, you will see how scrambled the actual waveform is.

At this point, do a "select all" operation (Ctrl+A) for Country_share1.wav, then copy it to clipboard (Ctrl+C). Afterwards, navigate to Country_share2.wav. Also select all (Ctrl+A) for Country_share2.wav, and then do a mix paste operation (Ctrl+Shift+V or by clicking menu Edit > Mix Paste).

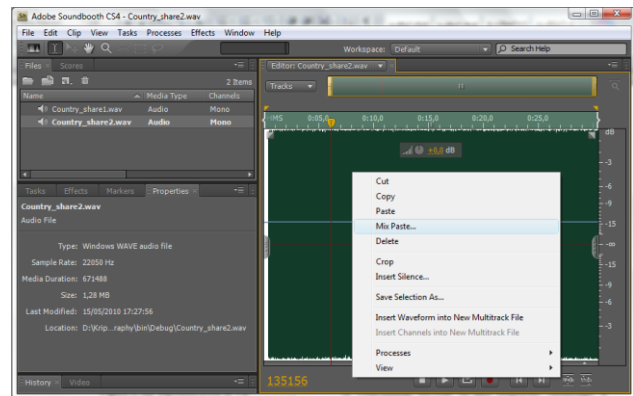


Figure 8. Mix-paste Country_share1.wav to Country_share2.wav.

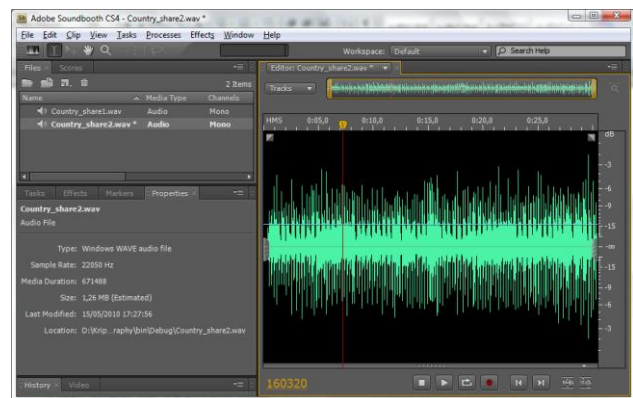


Figure 9. After mixing is done, you will see a waveform resembles the original audio file. Play it, the swinging country music comes back. Some little noises are introduced due to clipping effects occurred at the time of encryption.

V. CONCLUSION

Audio cryptography uses interferences of sound waves, similar to visual cryptography that uses addition of light. Why people don't use audio cryptography? Possibly the reason is that audio files are large, and the secret content inside the audio file (such as a person's speech) can be expressed in a text file and encrypted using any enciphering algorithm which does not waste spaces like audio do.

The strength of audio cryptography relies on the algorithm responsible for breaking samples into shares. Unlike mine, good algorithm should obfuscate all audio samples as random as possible, resulting shares containing only noise or hiss sound, and obligated to be reversible.

At the time this paper is written, only few references available. Audio cryptography still has wide prospect to grow. We are still dreaming for a *reversible* lossless audio compression, which can play major role in future audio cryptography.

REFERENCES

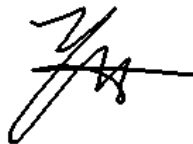
- [1] http://www.wisdom.weizmann.ac.il/~naor/PUZZLES/visual_sol.html
- [2] <http://www.springerlink.com/content/p4hqaw072d7e67pc/>
- [3] Guojun Lu, *Multimedia Database Management System*. Norwood, MA: Artech House, 1999. ISBN 0-890-06342-7.
(used in IF4055 Multimedia System)
- [4] <http://sharkysoft.com/archive/wave/docs/javadocs/lava/riff/wave/doc-files/riffwave-frameset.htm>
- [5] <http://electronics.howstuffworks.com/speaker.htm> through <http://electronics.howstuffworks.com/speaker8.htm>
- [6] <http://us.generation-nt.com/answer/mixing-audio-samples-help-31056382.html>
- [7] [VAH07] Frank Vahid, *Digital Design*. Hoboken, NJ: John Wiley & Sons, 2007. ISBN 0-470-04437-3.
- [8] Muhammad Fajrin Rasyid, "Kriptografi Audio dengan Teknik Interferensi Non-Biner". Makalah TA Teknik Informatika angkatan 2004, Institut Teknologi Bandung.
(http://www.informatika.org/~rinaldi/TA/Makalah_TA%20Fajrin.pdf)
- [9] <https://ccrma.stanford.edu/courses/422/projects/WaveFormat/>
- [10] <http://stackoverflow.com/questions/376036/algorithm-to-mix-sound>
- [11] <http://www.vttoth.com/digimix.htm>
- [12] <http://upload.wikimedia.org/wikipedia/commons/b/b8/Ear.jpg>

PERNYATAAN STATEMENT

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

I hereby declare that this paper is my own work, and is not a rechauffe, not a translation from other's paper, and is not a plagiarism.

Bandung, 29 April 2010



Yusuf Adriansyah
13507120