南開大學
Nankai University

网络空间安全学院
信息隐藏技术报告

姓名：姜涵 李晓彤 丁奕可 胡亚飞 孙蕗

学号：2113630 2112075 2113345 2111690 2112060

专业：信息安全

指导教师：李朝晖

2024 年 4 月 23 日

目录

1 实验要求

1. 任选一种媒体文件，进行格式剖析（建议用 UltraEdit）。
2. 针对该类型的文件，讨论可能的隐藏位置和隐藏方法。
3. 实现秘密信息的隐藏和提取。

2 wav 文件格式剖析

2.1 WAV 文件简介

WAV（Waveform Audio File Format）是一种流行的、以无损的方式存储音频数据的音频文件格式，通常用于在计算机系统中存储音频。WAV 文件使用 PCM（脉冲编码调制）编码，这意味着它们以原始音频样本的形式存储音频数据，保留了高质量的音频信息。由于 WAV 文件不经过任何压缩，它们通常具有较大的文件大小，但也因此保留了音频的完整性，使其成为专业音频编辑和处理的常见选择。

2.2 WAV 文件的结构

2.2.1 背景知识补充

WAV 文件属于 RIFF 结构形式的文件，这种结构是一种树状结构，其最基本的组成单元是 chunk（块），而每一个 chunk 则由辨识码，数据大小，以及数据本身组成，其中辨识码就是 RIFF 四个字母的 ASCII 码，共占用 4 个字节，用十六进制表示就是 52 49 46 46；辨识码之后的四个字节，表示除去文件辨识码和它自己之后剩余文件的大小，最后就是文件的数据本身了。chunk 的基本结构如下所示:

chunk 的基本结构

```
struct chunk
{
    uint32_t    ID; //块标识符 4Byte
    uint32_t    Size; //块数据大小 4Byte
    uint8_t     Data[Size]; //块数据
};
```

RIFF chunk 的数据域首先是 4 字节的 FormType，接着是若干个 subchunk，每一个 subchunk 又包含有自己的标识符、数据域的大小以及数据域。

其中:

1. WAV 格式的文件包含两个这样的 subchunk:FORMAT 子块、fact 子块.
2. fmt 子块主要用于描述该 wav 文件的信息，例如，采样率，数据量，编码格式，声道数等，其结构也如 RIFF，最开始 4 个字节表示标识符“fmt”，接着四个字符表示该块大小，后面的 18 个字节则表示文件的信息，同样，也是 fmt 子块的“数据”;
3. fact 子块并不是必须的，在标准的 WAV 文件中没有该子块，只有一些软件转化而来的才会增加该子块，其主要存储关于该文件内容的重要信息；data 块则主要就是 WAV 文件真正能播放的声音数据部分了;

4. 除了 RIFF chunk 可以嵌套其他的 chunk 外，另一个可以包含 subchunk 的就是 LIST chunk.

可以得到”RIFF Chunk” 的示意图 (由于在标准的 WAV 文件中没有 fact 子块, 这里不对其进行说明)、RIFF chunk 和 LIST chunk 的基本结构如下所示:

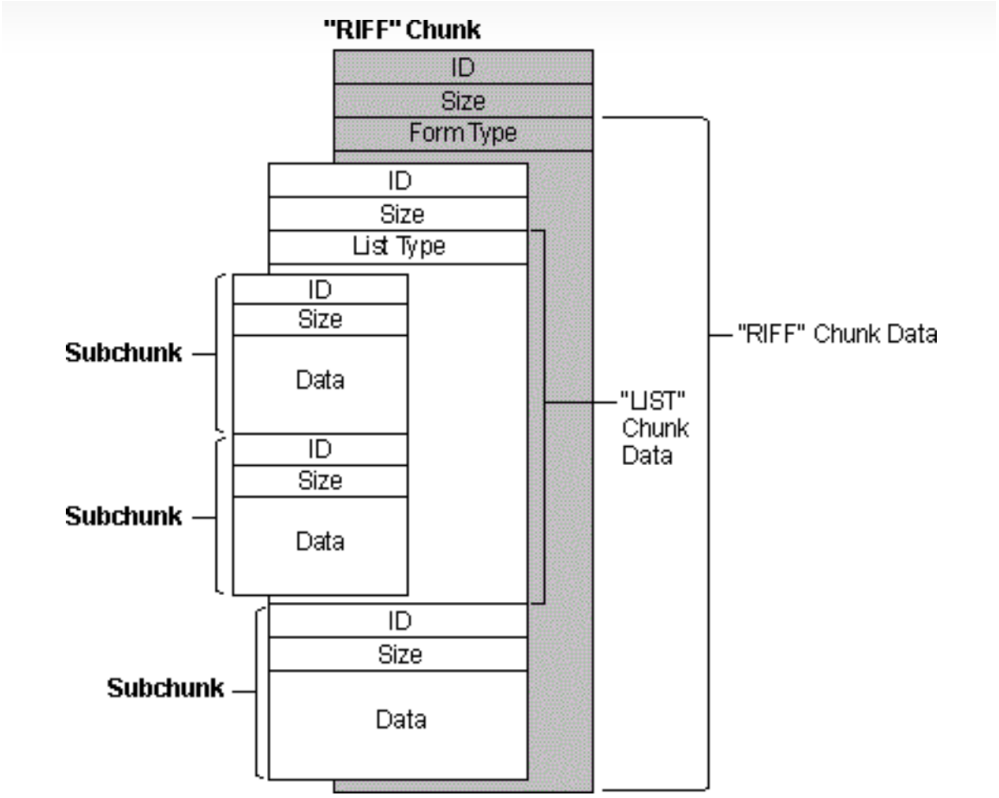

图 2.1: RIFF chunk 和 LIST chunk 的基本结构

RIFF chunk 和 LIST chunk 的基本结构

```
struct chunk
{
    uint32_t ID; // 块标识符: 'RIFF'或者'LIST'
    uint32_t Size; // 块数据大小
    struct ChunkData { // 块数据
        uint32_t Type; // 包含的subchunk的数据类型,
            与上面图中的FormType和ListType对应
        uint8_t Data[Size-4]; // 包含的subchunk
    };
};
```

其中一个 RIFF 文件的总大小为：RIFF chunk 的 Size+8，这里的 8 是 ID 和 Size 所占用的空间.

2.2.2 WAV 文件的 RIFF 区块

基于以上背景知识,WAV 文件的 RIFF 区块结构如下:

名称	偏移地址	字节数	端序	内容
ID	0x00	4Byte	大端	'RIFF' (0x52494646)
Size	0x04	4Byte	小端	fileSize - 8
Type	0x08	4Byte	大端	'WAVE'(0x57415645)

图 2.2: WAV 文件的 RIFF 区块结构

- 以’RIFF’为标识;
- Size 是整个文件的长度减去 ID 和 Size 的长度;
- Type 是 WAVE 表示后面需要两个子块：Format 区块和 Data 区块.

在 C++ 中的表示为:

WAV 文件的 RIFF 区块

```
struct chunk
{
    uint32_t ID; // 块标识符: 'RIFF'或者'LIST'
    uint32_t Size; // 块数据大小
    struct ChunkData { // 块数据
        uint32_t Type; // 包含的subchunk的数据类型,
            与上面图中的FormType和ListType对应
        uint8_t Data[Size-4]; // 包含的subchunk
    };
};
```

2.2.3 WAV 文件的 FORMAT 区块

WAV 文件的 FORMAT 区块结构如下:

名称	偏移地址	字节数	端序	内容
ID	0x00	4Byte	大端	'fmt ' (0x666D7420)
Size	0x04	4Byte	小端	16
AudioFormat	0x08	2Byte	小端	音频格式
NumChannels	0x0A	2Byte	小端	声道数
SampleRate	0x0C	4Byte	小端	采样率
ByteRate	0x10	4Byte	小端	每秒数据字节数
BlockAlign	0x14	2Byte	小端	数据块对齐
BitsPerSample	0x16	2Byte	小端	采样位数

图 2.3: WAV 文件的 FORMAT 区块结构

- 以'fmt ' 为标识
- Size 表示该区块数据的长度（不包含 ID 和 Size 的长度）
- AudioFormat 表示 Data 区块存储的音频数据的格式，PCM 音频数据的值为 1

- NumChannels 表示音频数据的声道数，1：单声道，2：双声道
- SampleRate 表示音频数据的采样率
- ByteRate 每秒数据字节数 $= SampleRate * NumChannels * BitsPerSample/8$
- BlockAlign 每个采样所需的字节数 $= NumChannels * BitsPerSample/8$
- BitsPerSample 每个采样存储的 bit 数，8：8bit，16：16bit，32：32bit

在 C++ 中的表示为:

WAV 文件的 FORMAT 区块

```
struct chunk
{
    uint32_t ID; // 块标识符: 'RIFF'或者'LIST'
    uint32_t Size; // 块数据大小
    struct ChunkData { // 块数据
        uint32_t Type; // 包含的subchunk的数据类型,
            与上面图中的FormType和ListType对应
        uint8_t Data[Size-4]; // 包含的subchunk
    };
};
```

2.2.4 WAV 文件的 DATA 区块

WAV 文件的 DATA 区块结构如下:

名称	偏移地址	字节数	端序	内容
ID	0x00	4Byte	大端	'data' (0x64617461)
Size	0x04	4Byte	小端	N
Data	0x08	NByte	小端	音频数据

图 2.4: WAV 文件的 DATA 区块结构

- 以’data’为标识
- Size 表示音频数据的长度，N = ByteRate * seconds
- Data 为音频数据

在 C++ 中表示为:

WAV 文件的 DATA 区块

```
typedef struct
{
    uint32_t ID;  /* "data" = 0x61746164 */
```

```
    uint32_t Size;
} WavDATA;
```

2.3 使用 UltraEdit 剖析 WAV 文件

右键单击从语音库下载的 WAV 文件 male_voice.wav, 查看其属性:

图 2.5: 文件属性

可以看到其大小为 146kB(150，044 字节)。接下来，使用软件 UltraEdit 将文件以十六进制数据的形式打开，文件打开之后如下图所示:

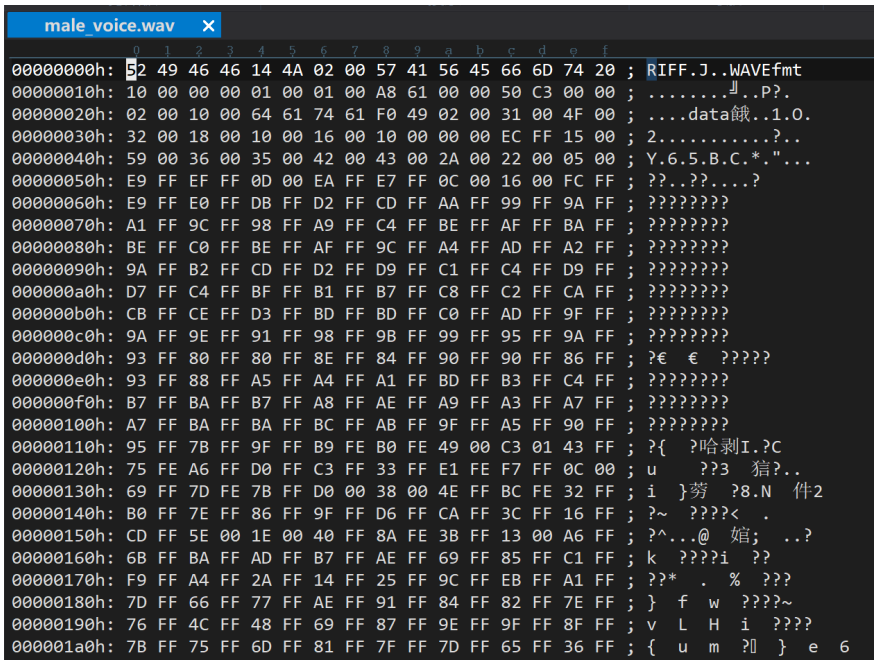

图 2.6: 使用 UltraEdit 打开文件

2.3.1 RIFF 区块

文件开头是以 52 49 46 46，正好是 RIFF：

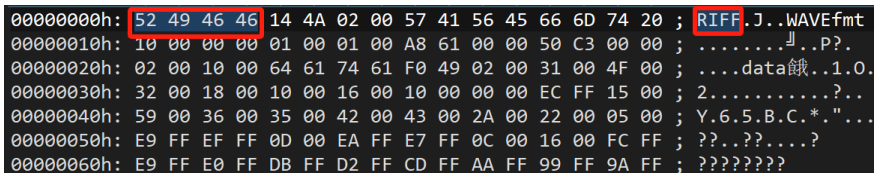

图 2.7: RIFF 标识符

而其后面的四个字节为 14 44 02 00，按照前面所说的文件结构，该数据应该是表示此 WAV 文件的大小才对，在前面的属性描述里，我们已经得知该文件大小为 150044 字节；此处采用的是小端方式存储数据，即低位在前、高位在后，因此前面的 14 44 02 00，实际应该是 00 02 44 14.

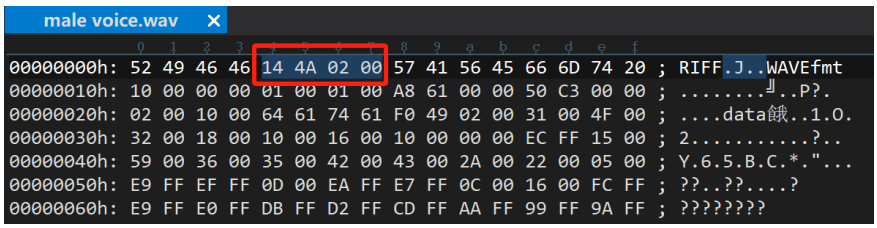

图 2.8: 文件大小

在将这个调换顺序之后的数字转换成十进制之后，会发现这个数字表示的大小是 150036，相比前面的数据，刚好差了一个 8，而这正好就是头文件的标识符和数据大小所占据的 8 个字节；

图 2.9: 文件大小进制转换

紧接着的四个字节为 57 41 56 45，表示 Type 是 WAVE。

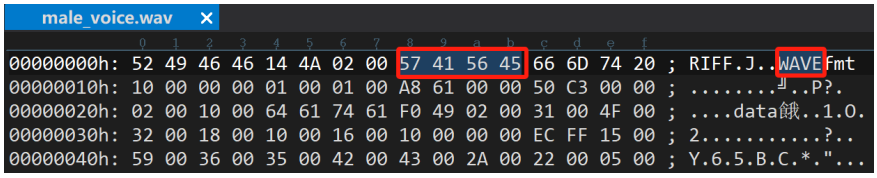

图 2.10: 文件类型

2.3.2 FORMAT 区块

由上一节的分析，我们可以对应到 EORMAT 区块的各个组成部分：

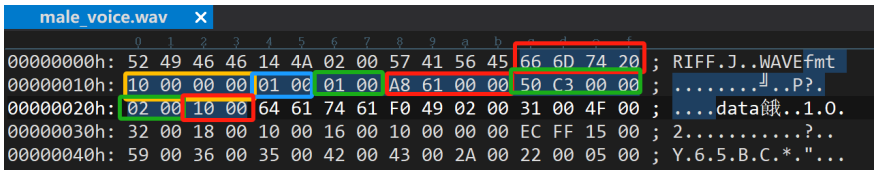

图 2.11: FORMAT 区块

NumChannels=1（单声道）、SampleRate = 25000Hz、ByteRate = 50000、BlockAlign = 2、BitsPerSample=16；

2.3.3 DATA 区块

由上一节的分析，我们可以对应到 DATA 区块的各个组成部分：

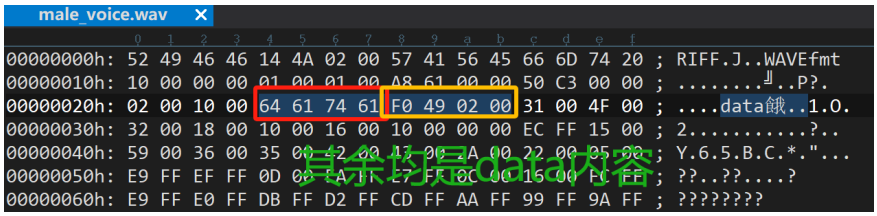

图 2.12: DATA 区块

size = 0x249f0=150000，从 data 头到尾，一共刚好 150000:

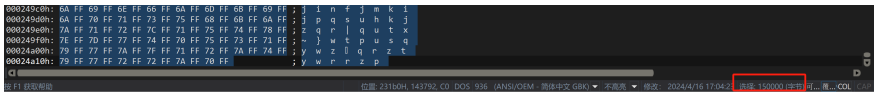

图 2.13: data 长度

3 隐藏位置及隐藏方法讨论

3.1 信息隐藏位置

在 WAV 文件中进行信息隐藏时，可以利用多个位置来隐藏信息。以下是可能出现的信息隐藏位置：

1. **音频样本数据：** WAV 文件的主要部分是音频样本数据，其中包含了实际的音频波形信息。最常见的信息隐藏方法是在音频样本数据中修改部分位，例如使用 LSB（Least Significant Bit）算法，在最低有效位中嵌入隐藏信息。

2. **文件头部信息：** WAV 文件的文件头部包含了关于音频格式、采样率、通道数等信息。这些信息通常不直接影响音频的听觉质量，因此可以在其中嵌入隐藏信息，例如通过修改某些标记或者元数据来隐藏信息。

3. **采样点之间的时间差异：**在音频数据的时间域中，可以通过微调每个采样点之间的时间差异，来嵌入隐藏信息。这种方法可以提供相对较高的容量和隐蔽性，但会对音频的质量产生一定的影响。

4. **附加数据块：**WAV 文件格式支持在音频数据块之外添加附加数据块，如 LIST、INFO 等。这些附加数据块可以用于存储额外的信息，如作者、专辑信息等。隐藏信息可以通过修改或添加这些附加数据块来实现。

5. **频谱信息：**WAV 文件的频谱信息也是隐藏信息的潜在位置。通过在音频频谱中嵌入隐藏信息，可以利用人类听觉系统对频谱变化的较低敏感度来实现隐秘传输。

6. **添加噪声信号：**可以在 WAV 文件中添加噪声信号来隐藏信息。这种方法通常称为扩频法。噪声信号可以与原始音频信号混合，以实现信息的隐藏和传输。

3.2 常见隐藏方法

常见的隐藏方法包括 LSB 算法、回声隐藏算法、相位隐藏算法、扩频法、傅氏变换域算法、离散余弦变换域算法、小波变换域算法等。

1. **LSB（Least Significant Bit）算法：**LSB 算法通过修改音频信号中的最低有效位来嵌入隐藏信息。这些最低有效位的变化通常对音频质量几乎没有影响，因此可以用来隐藏秘密信息。

   优点：简单易实现，计算成本低。对音频质量的影响较小，通常不容易被察觉。

   缺点：安全性较低，容易受到攻击。对某些音频处理操作可能会破坏隐藏的信息。

2. **回声隐藏算法：**回声隐藏算法通过在音频信号中添加微小的回声或混响来隐藏信息。这些微小的变化通常不会被人类听觉察觉到，因此可以用来嵌入秘密信息。

   优点：对人类听觉系统的影响较小，隐藏效果较好。隐藏信息的安全性较高。

   缺点：需要对回声参数进行精确控制，较为复杂。隐藏信息量有限，不能嵌入过多的信息。

3. **相位隐藏算法：**相位隐藏算法通过微调音频信号的相位来隐藏信息。由于人类听觉对音频相位变化的敏感度较低，因此可以通过调整相位来嵌入秘密信息。

   优点：对音频质量的影响较小，隐藏效果较好。隐藏信息的安全性较高。

   缺点：隐藏信息量有限。相位调整可能会引起音频信号的畸变。

4. **扩频法：**扩频法通过引入具有高度相关性的噪声信号，将秘密信息嵌入到音频信号中。这个噪声信号被称为扩频序列，通过与要隐藏的信息进行异或运算，将信息嵌入到扩频序列中，然后将扩频序列加入到音频信号中。

   优点：安全性高、抗攻击性强、隐藏容量大。

   缺点：难以适用于某些环境、复杂性高。

5. **傅氏变换域算法：**傅氏变换域算法将音频信号转换到频域（通过傅立叶变换），然后在频域中嵌入秘密信息，最后再通过逆傅立叶变换将其转换回时域。

   优点：可以在频域中嵌入大量信息。隐藏信息的安全性较高。

   缺点：需要进行频域转换，计算量较大。对音频信号的处理可能导致隐藏信息的丢失。

6. **离散余弦变换域算法：**离散余弦变换（DCT）是一种常用的频域变换方法，类似于傅立叶变换，但更适合音频和图像数据的处理。在 DCT 域中，可以嵌入秘密信息，然后通过逆 DCT 将其还原。

   优点：与傅氏变换相比，DCT 更适合音频和图像数据的处理。隐藏信息的安全性较高。

   缺点：需要进行频域转换，计算量较大。对信号的处理可能导致隐藏信息的丢失。

7. **小波变换域算法：**小波变换是一种基于频率的变换方法，可以将信号分解成不同频率的成分。在小波变换域中，可以将秘密信息嵌入到不同频率成分中，从而隐藏信息。

   优点：具有更好的隐藏性能和鲁棒性，可以在不同频率成分中嵌入信息。

   缺点：计算量较大，实现相对复杂，需要对小波变换参数进行适当选择，以保持良好的隐藏效果。

4 秘密信息的隐藏和提取的实现

根据我们前面对 wav 文件格式的剖析，可以知道，在 wav 文件中，从文件头到数据前面一共有 44 字节的文件格式说明，后面便是数据段存储的位置，因此我们采用 LSB 算法在 wav 文件的数据段进行信息隐藏和提取。

4.1 LSB 算法简介

LSB 方法是一种最简单的数据嵌入方法。任何秘密数据都可以看做是一串二进制位流，而音频文件的每一个采样数据也是用二进制数来表示的。这样，可以将每一个采样值的最不重要位 (大多数情况下为最低位)，用代表秘密数据的二进制位替换，以达到在音频信号中编码进秘密数据的目的。

4.2 二值图片的隐藏

我们选用二值图片进行隐藏，将二值图像转为一维向量，并使用 *bitset* 函数将其嵌入到 wav 文件的数据段，即将隐藏信息嵌入音频数据的 44 字节之后，随后分别绘制原始音频和带有隐藏信息的音频波形图，并进行分析。

详细代码如下所示，具体过程已写在注释内，不再赘述。

Listing 1: 将二值图片隐藏至 wav 文件中

```
% 利用LSB方法实现在wav文件中的信息隐藏与提取
clc;
clear;
close all;

% 打开原始音频文件并读取数据
fid = fopen('male_voice.wav', 'rb');
origin_audio = fread(fid, inf, 'uint8');
fclose(fid);

% 计算可隐藏信息的长度（去除文件头44字节）
len = length(origin_audio) - 44;

```

```
% 秘密图像
ScrietImg = imread("Pig.jpg");
GrayScrietImg = im2gray(ScrietImg);
BinScrietImg = imbinarize(GrayScrietImg);

% 读取要隐藏的信息（二值图像）
[rows, cols] = size(BinScrietImg);
hidden_bits = BinScrietImg(:);
if rows * cols > len
    error('音频载体太小，请更换载体');
end

% 将隐藏信息嵌入到音频数据的44字节后面
modified_audio = origin_audio;
for k = 1 : rows * cols
    modified_audio(44 + k) = bitset(modified_audio(44 + k), 1,
       hidden_bits(k));
end

% 将带有隐藏信息的音频数据写入到新文件
fid = fopen('modified_audio.wav', 'wb');
fwrite(fid, modified_audio, 'uint8');
fclose(fid);

% 读取原始音频和带有隐藏信息的音频
[raw_audio, fs_raw] = audioread('male_voice.wav');
[modified_audio, fs_mod] = audioread('modified_audio.wav');

% 计算时间向量
t_raw = (0:length(raw_audio)-1) / fs_raw;
t_mod = (0:length(modified_audio)-1) / fs_mod;

% 绘制原始音频波形图
figure;
subplot(2,1,1);
plot(t_raw, raw_audio);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Raw Audio Waveform');

% 绘制带有隐藏信息的音频波形图
subplot(2,1,2);
```

```
plot(t_mod, modified_audio);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Modified Audio Waveform');

% 调整子图布局
sgtitle('Comparison of Raw and Modified Audio Waveforms');
```

4.3 二值图片的提取

与隐藏信息的过程类似，从带有隐藏信息音频的 LSB 中提取出一维隐藏信息，并将其转换为二值图像，随后展示提取出的二值图像。

Listing 2: 提取隐藏在 wav 文件中的二值图片

```
audio_data = fread(fid, inf, 'uint8');
fclose(fid);

% 提取隐藏信息的二值图像
hidden_bits = zeros(rows * cols, 1);
for k = 1 : rows * cols
    hidden_bits(k) = bitget(audio_data(44 + k), 1);
end

% 将一维隐藏信息转换为二维图像
hidden_image = reshape(hidden_bits, rows, cols);

% 显示提取的隐藏信息图像
figure;
imshow(hidden_image,[]);
title('Extracted Hidden Image');
imwrite(hidden_image,"hidden_image.bmp");
```

4.4 实验结果及分析

在将二值图片隐藏至 wav 文件后，绘制了原始音频的波形图和带有隐藏信息的音频的波形图，如图4.14所示:

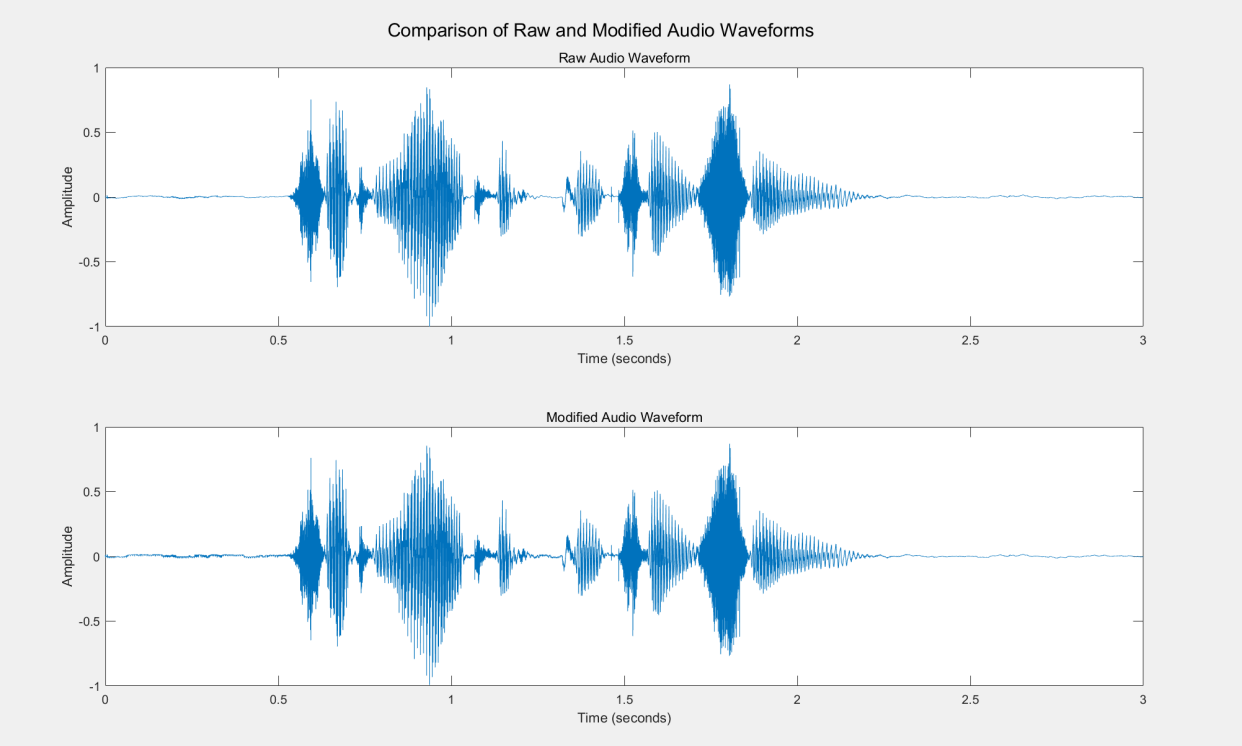

图 4.14: 原始音频和修改后音频的波形对比图

我们发现嵌入隐藏信息的波形图与原始音频的波形图几乎完全相同，因此也可以确定嵌入的隐藏信息并没有对原始音频有很大的影响，为了保证实验的严谨性，我们更进一步查看两个音频的区别，选取前 0.5s 的波形图4.15进行查看：

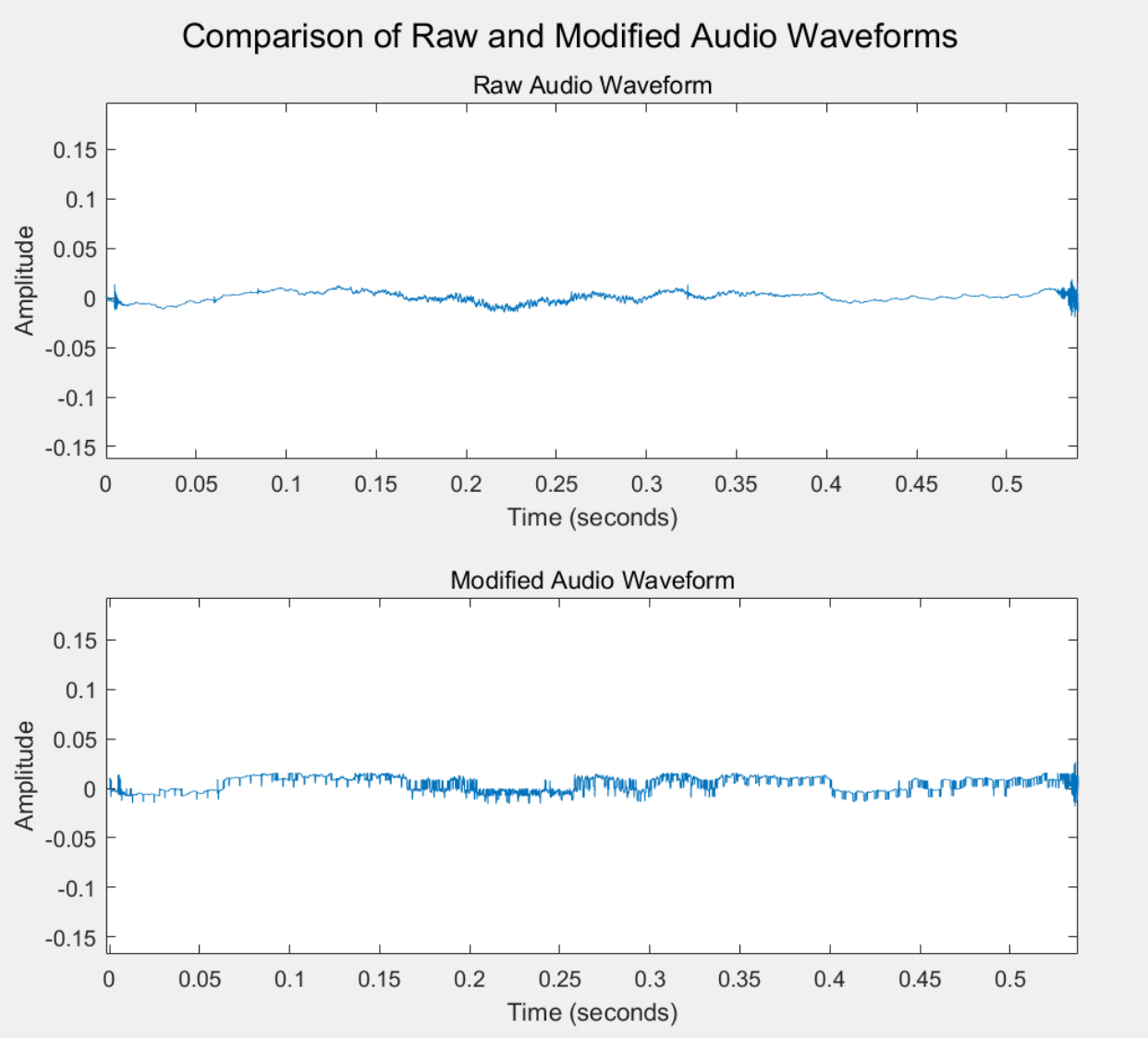

图 4.15: 原始音频和修改后音频的波形对比图 (前 0.5s)

可以看到，嵌入隐藏信息的音频与原始音频在细节上还是有一定区别的，会多出一些噪音，而这些噪音正是我们所隐藏的二值图片。

从嵌入隐藏信息的音频中进行提取，得到我们的二值图片：

图 4.16: 提取出的隐藏二值图片