Is there an echo

Challenge Description: Maybe next time you should record your music in an acoustically treated space.

Competition: CSAWCTF 2024 / Forensic

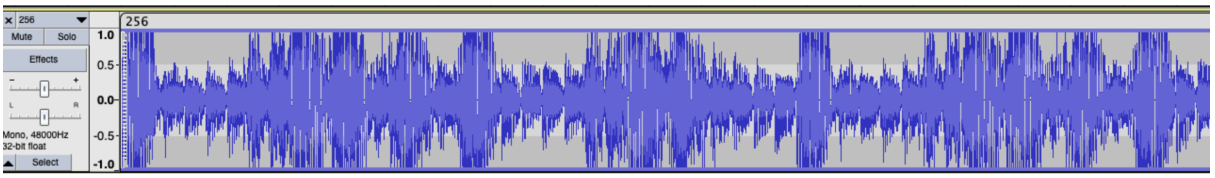
TL;DR:

In the end of the file hex stream it says “cepstral domain single echo”, which leaded to a github repo with a solve function `echo_dec(signal, L, d0, d1, len_msg)`. Each beat takes 0.35 second, sample rate 48000. Do a cepstral domain drawing with Hann window, window size 256, can see the echo hiding peak happen in 0.0010 and 0.0012 second. 0.35*48000=L, d0=0.001*48000, d1=0.0012*48000. Final function call: `echo_dec(audio.data, 16800, 48, 58);`

Generic

The challenge has a file called 256.wav.

```
ExifTool Version Number         : 12.70
File Name                       : 256.wav
Directory                       : /Users/chara/ctf/csaw
File Size                       : 8.6 MB
File Modification Date/Time     : 2024:09:07 17:56:11-07:00
File Access Date/Time           : 2024:09:07 17:56:16-07:00
File Inode Change Date/Time     : 2024:09:07 17:56:16-07:00
File Permissions                : -rw-r--r--
File Type                       : WAV
File Type Extension             : wav
MIME Type                       : audio/x-wav
Encoding                        : Microsoft PCM
Num Channels                    : 1
Sample Rate                     : 48000
Avg Bytes Per Sec               : 96000
Bits Per Sample                 : 16
Duration                        : 0:01:30
```



After listen to it can here it is repetitive music.

First approach

Consider the title called echo, was first thinking to isolate out the echos. Uses the first audio clip as the original no echo sound, split the audio and subtract every clip the first audio clip.

Since the audio repeats every 6.4 second, split the first 6.4 second.

```
from pydub import AudioSegment
import os

def split_wav(file_path, clip_duration=6.4):
    # Load the audio file
    audio = AudioSegment.from_wav(file_path)
    # Calculate duration of the audio file in milliseconds
    audio_duration_ms = len(audio)
    # Clip duration in milliseconds
    clip_duration_ms = clip_duration * 1000
    # Create output directory for the clips
    output_dir = "files"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    # Split the audio into clips
    for i in range(0, audio_duration_ms, int(clip_duration_ms
        start_time = i
        end_time = min(i + int(clip_duration_ms), audio_durat
        clip = audio[start_time:end_time]
        # Save the clip
        clip_file_name = f"clip_{i//1000}_to_{end_time//1000}
        clip.export(os.path.join(output_dir, clip_file_name),
```

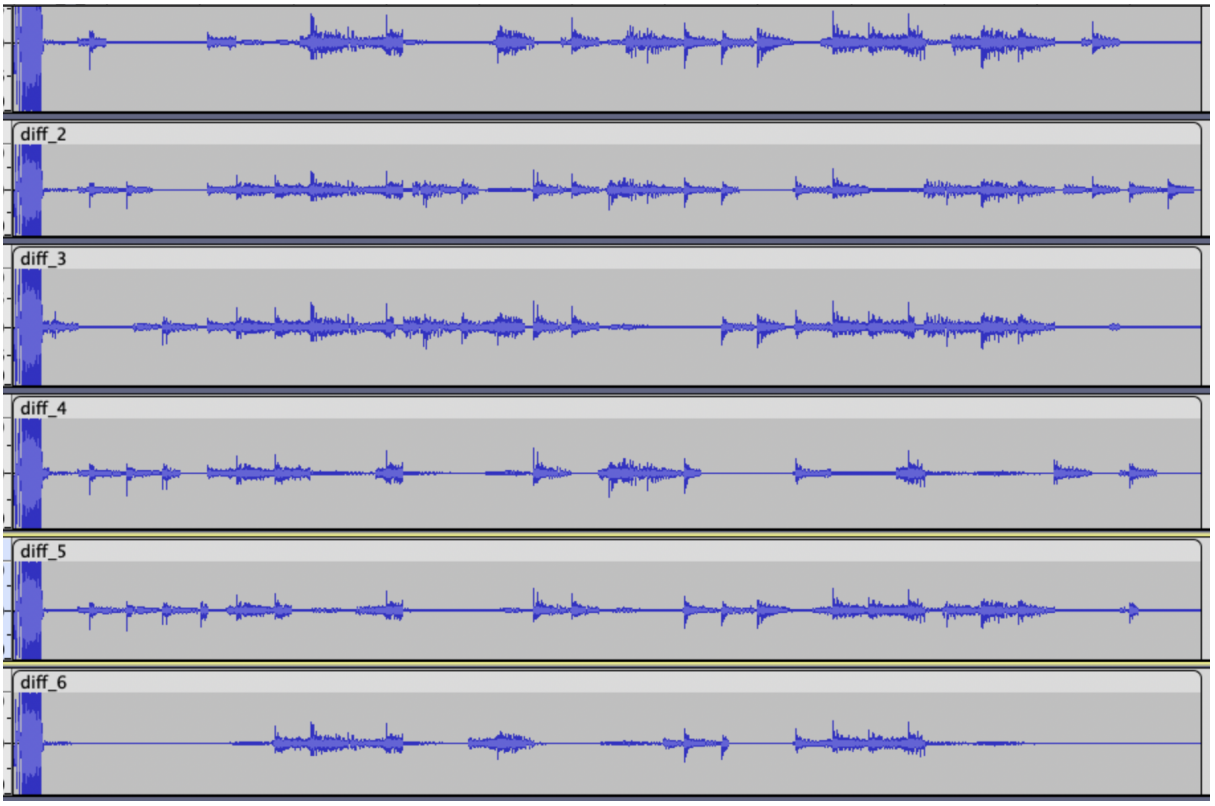
```
        print(f"Exported {clip_file_name}")
 file_path = "256.wav"
 split_wav(file_path)
```

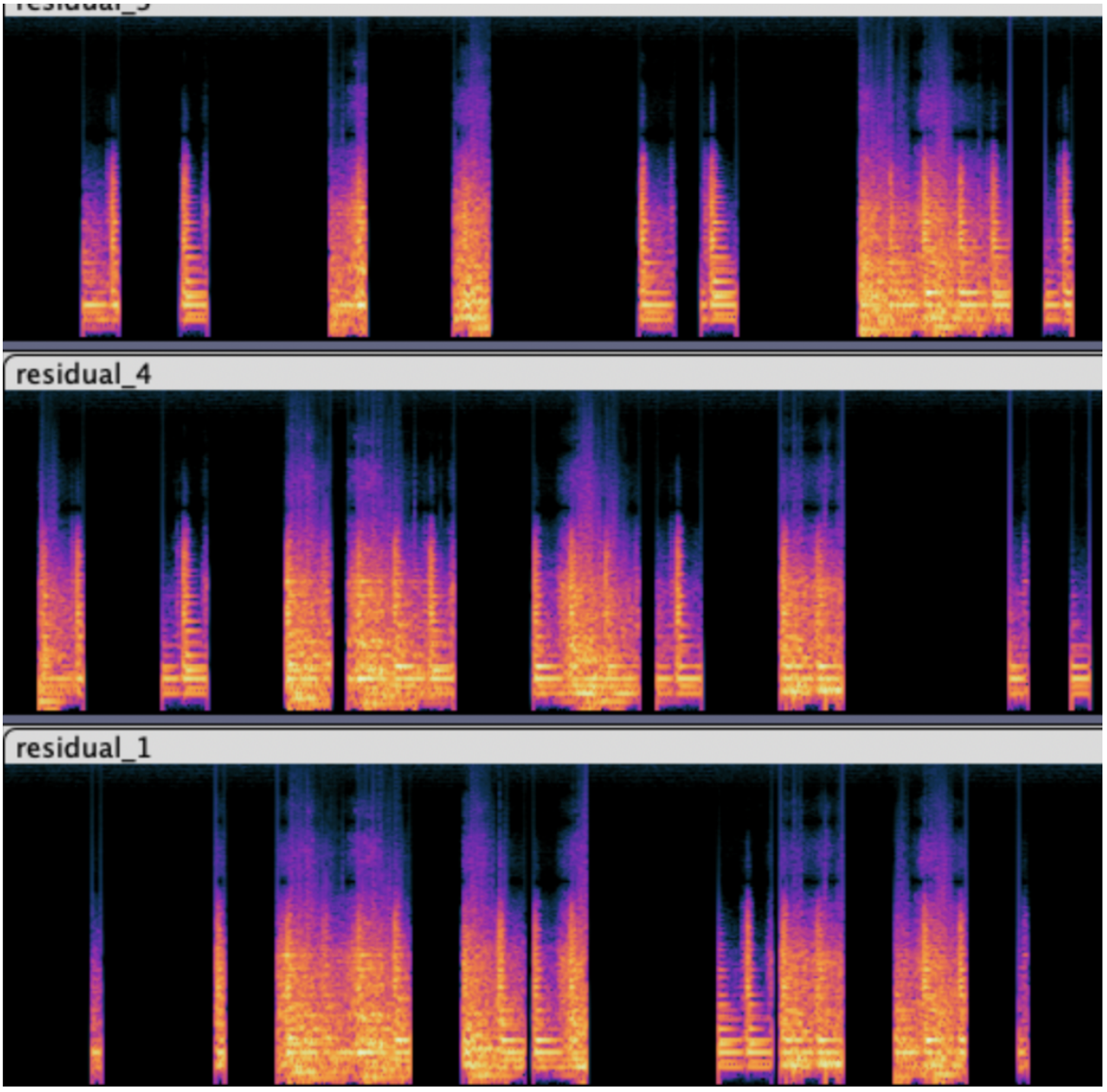
Then did some comparison

```
# convert wavs in the output clip to number list
from scipy.io.wavfile import read, write
from pydub import AudioSegment
import os
import numpy as np
def encode_wav(samples):
    # encode the wav range from -32768-32767 to 0-65535
    return samples + 32768
def read_wav(file_path):
    sample_rate, samples = read(file_path)
    samples = encode_wav(samples)
    return samples

wav_file_list = os.listdir("output_clips")
wav_file_list.sort()
wav_0 = os.path.join("output_clips", "clip_0_to_6.wav")
wav_0 = read_wav(wav_0)

sample_rate = 48000
save_path = "output_diff/"
differences = []
for i in range(1, len(wav_file_list)):
    file_path = os.path.join("output_clips", wav_file_list[i]
    samples = read_wav(file_path)
    # the difference compared to the wav 0 file
    diff = samples - wav_0
    differences.append(samples - wav_0)
#     store them back to a new wav file
    if not os.path.exists(save_path):
        os.makedirs(save_path)
    diff = diff.astype(np.int16)
    diff = AudioSegment(diff.tobytes(), frame_rate=sample_rat
```

```
    diff.export(save_path + f"diff_{i}.wav", format="wav")

together = []
for i in differences:
    together.extend(i)

together = np.array(together)
together = together.astype(np.int16)
together = AudioSegment(together.tobytes(), frame_rate=sample
together.export(save_path + "together.wav", format="wav")
```

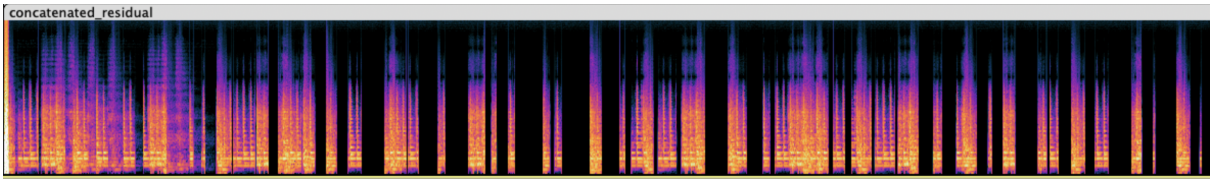
And saw some interesting behaviors:





I see each of them are cut of at different place

Then I tried to manually read of the different time and clip it off but the result was not understandable.




```
# the interval that it has music
diff0 = [(0, 0.35), (0.90, 1.05), (1.40, 1.95),(2.45, 2.65),
```

```
# get everything that is not in the interval
from pydub import AudioSegment
import os
import numpy as np

sample_rate = 48000
path = "output_diff/"
length = 6.25
output_wav = [0 for _ in range(int(length*sample_rate))]
clip = AudioSegment.from_wav(f"{path}diff_0.wav")
clip = np.array(clip.get_array_of_samples())
# inverse the intervals
inversed_intervals = []
for i in range(1, len(diff0)):
    inversed_intervals.append((diff0[i-1][1], diff0[i][0]))

inversed_intervals.append((diff0[-1][1], length))
for i, (start, end) in enumerate(inversed_intervals):
    output_wav[int(start*sample_rate):int(end*sample_rate)] =


output_wav = np.array(output_wav)
output_wav = output_wav.astype(np.int16)
output_wav = AudioSegment(output_wav.tobytes(), frame_rate=sa
output_wav.export(f"{path}output.wav", format="wav")
```

However, this looks like **Echo Hiding.**

Echo Hiding

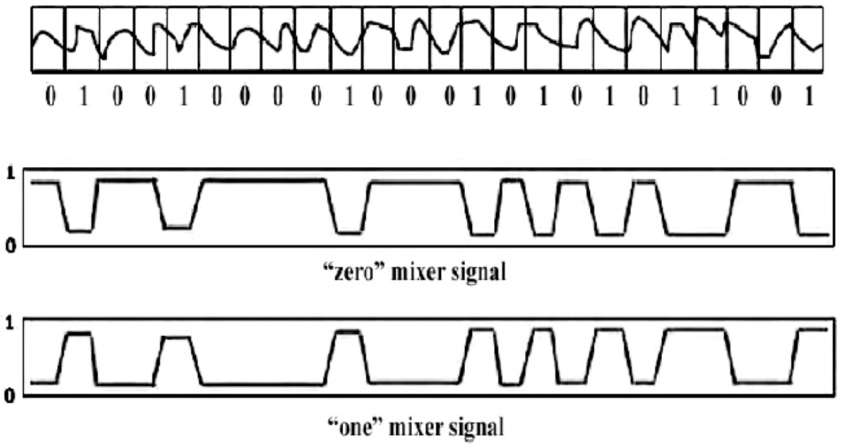
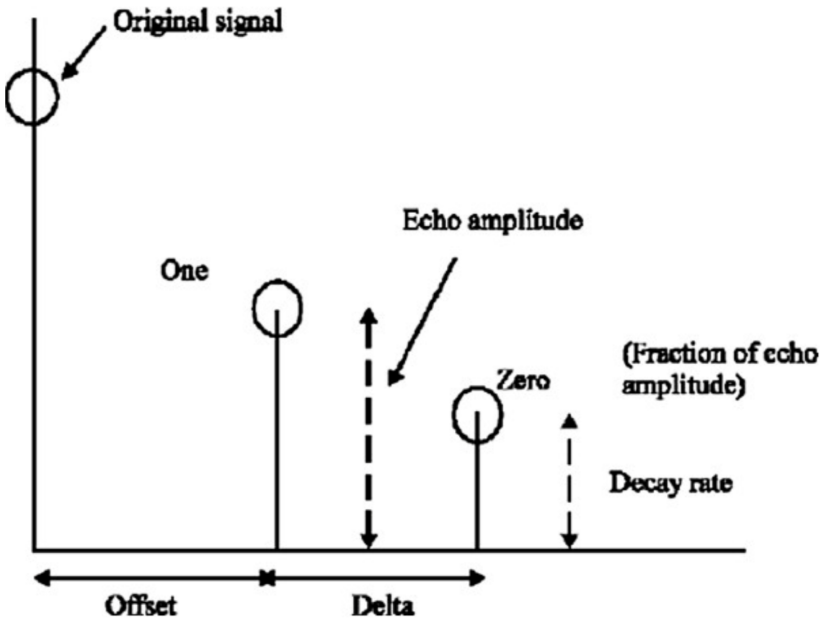


Figure 4 from AUDIO STEGANALYSIS OF LSB AUDIO USING MOMENTS AND MULTIPLE REGRESSION MODEL

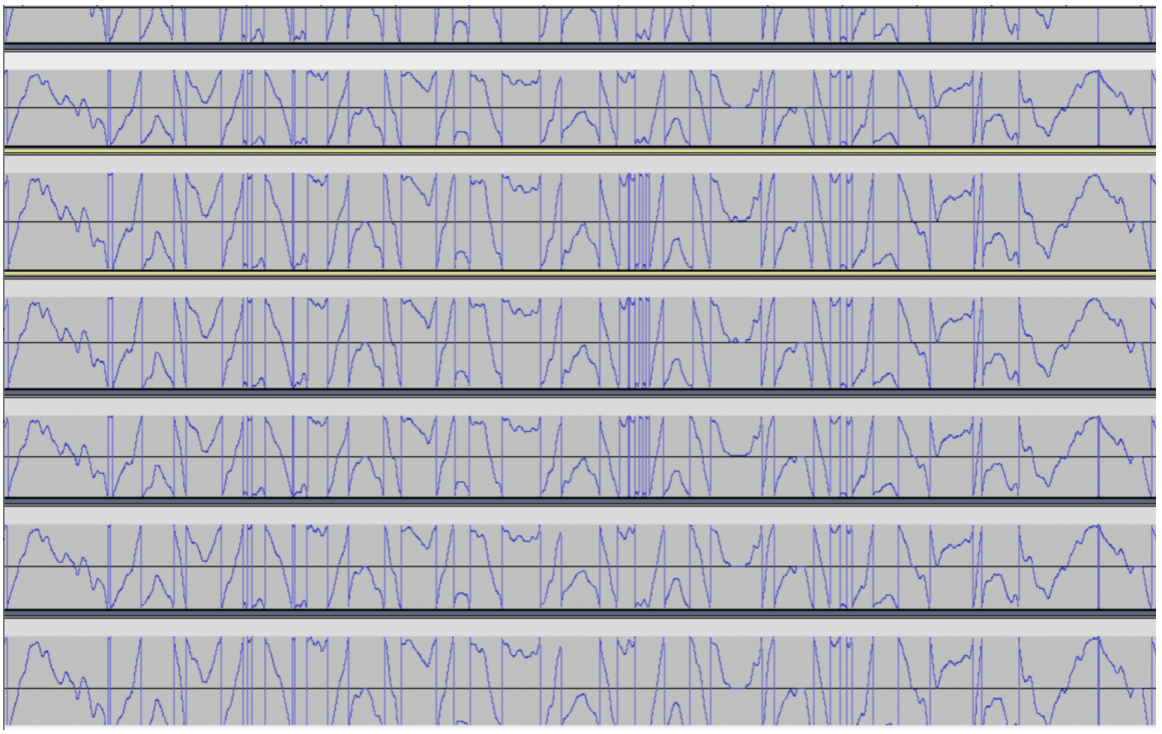
Echo hiding is a technique used in audio steganography, where the secret information embedded into audio signal by using small echos that human can't hear.

Typically there are two delays, one represent 0 and another one represent 1. It explains pretty well in the next picture:



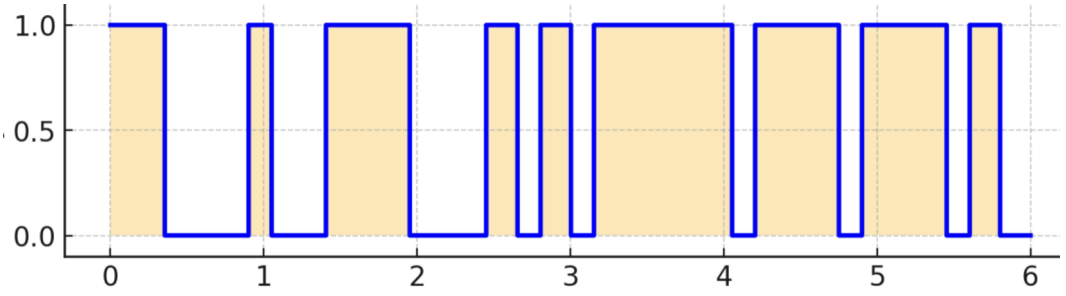
From Comparative Study of Digital Audio Steganography Techniques

At this point I had something I was confused on, because in every audio clip that I extracted, the beginning is some random sound I couldn't understand and look like this:

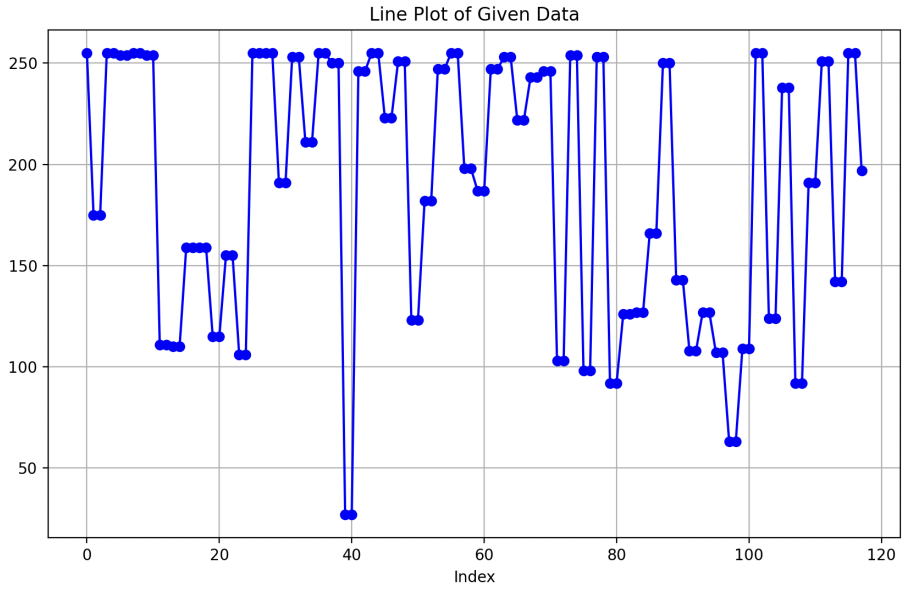


Isn't this similar to the echo hiding picture?

And I speed tons of time on this part and in end didn't figure out anything, and believe it's something about the music generation.



a graph that I was trying to extract which time it changes and result in nothing



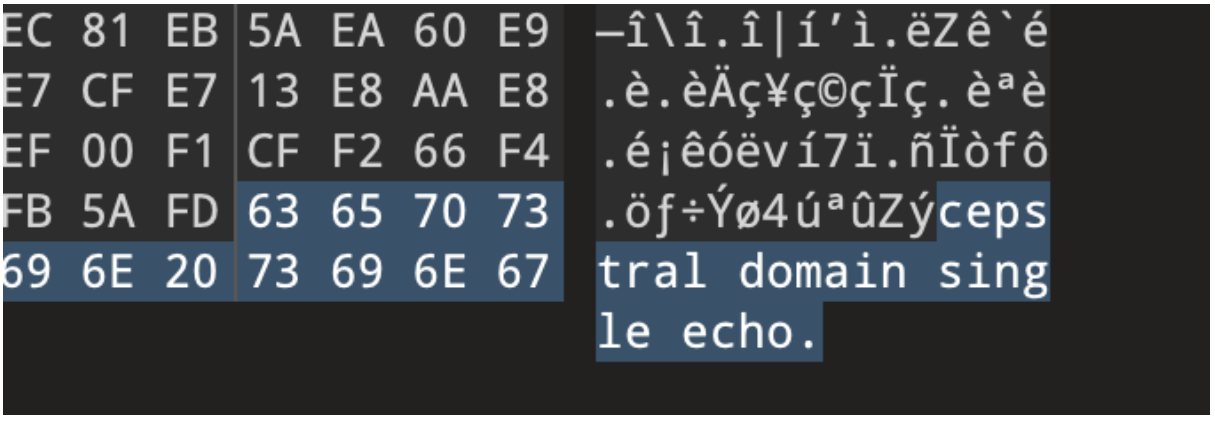
a random graph that I tried

Till this point I'm so lost in the chall, there are too many potential things to do.

1. What has this to do with binary? the file title is 256, the audio repeat every 6.4 second.
2. What does echo hiding mean here?

Cepstral domain single echo

I never took a look at the file hex, because thought it's just a normal file. Thx to my teammate @cinabun reading the file, founded that in the end of the file there is a text saying "cepstral domain single echo"



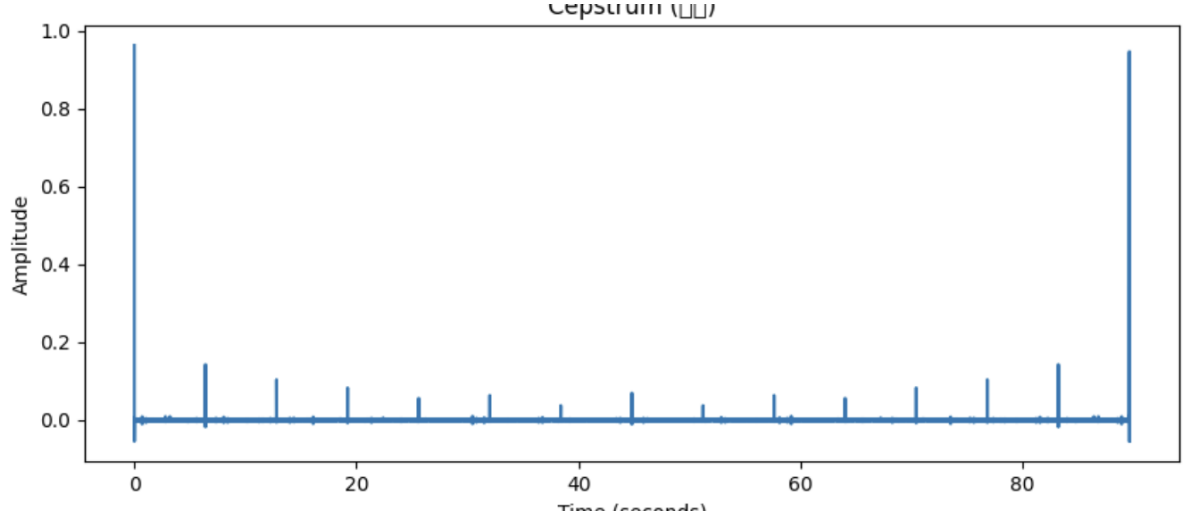
The cepstrum is a signal representation derived by taking the inverse Fourier transform(ift) of the logarithm of the magnitude of a signal's Fourier transform. It's useful for analyzing echoes because it transforms the convolution of two

signals (such as an original sound and its echo) into a sum, making the delay between the original and the echoed signal more distinguishable.

So I tried to plot the clips with 6.4 second:

```
import numpy as np
import librosa
import matplotlib.pyplot as plt
from scipy.fftpack import fft, ifft


def compute_cepstrum(audio_signal, sample_rate):
    spectrum = fft(audio_signal)
    log_spectrum = np.log(np.abs(spectrum) + 1e-10)
    cepstrum = np.real(ifft(log_spectrum))
    return cepstrum


def detect_echo_in_cepstrum(cepstrum, sample_rate):
    time_range = np.arange(int(0.001 * sample_rate), int(0.05
    echo_delay = np.argmax(np.abs(cepstrum[time_range])) + ti
    echo_delay_time = echo_delay / sample_rate
    return echo_delay_time


def plot_cepstrum(cepstrum, sample_rate):
    time_axis = np.arange(len(cepstrum)) / sample_rate
    plt.figure(figsize=(10, 4))
    plt.ylim(0, 0.1)
    plt.plot(time_axis, cepstrum)
    plt.title('Cepstrum')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.show()

file_path = 'output_diff/diff_1.wav'
audio_signal, sample_rate = librosa.load(file_path, sr=None)

# audio_signal = audio_signal[int(0.3 * sample_rate):int(6.2
```

```
cepstrum = compute_cepstrum(audio_signal, sample_rate)
echo_delay_time = detect_echo_in_cepstrum(cepstrum, sample_ra
plot_cepstrum(cepstrum, sample_rate)
```



Well looks no meaning to me. Did more research find the repo of https://github.com/ktekeli/audio-steganography-algorithms, so I tried 02-Echo-Hiding/01-Echo-Hiding-Single-Kernel.

The `echo_dec(signal, L, d0, d1, len_msg)` function takes in these four prams. Signal is the signal data, L is the Length of frames, d0 and d1 is the Delay rate.

Since the sample rate is 48000, the slice is 6.4 second of repetitive music, first echo on the graph is 89

$$\mathrm{L} = 6.4 \bullet 48000 = 307200 \text{ samples}$$

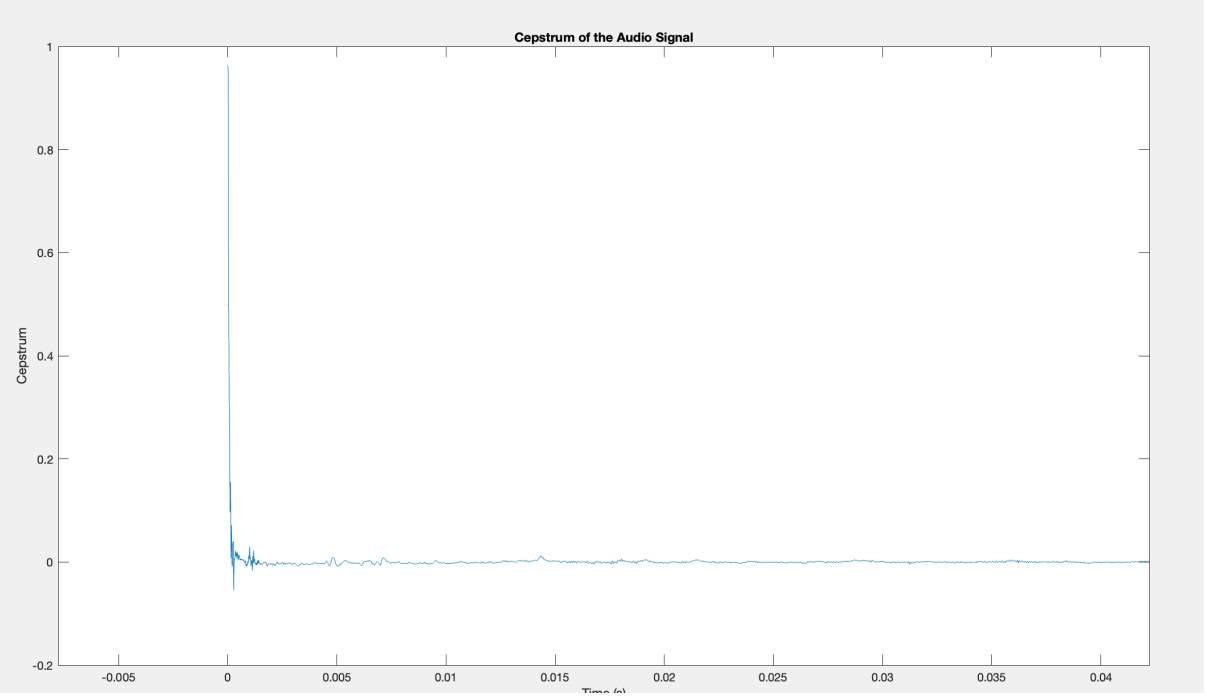
```
close all; clear all; clc;

audio = audioload();

msg = echo_dec(audio.data, 307200, 89, 57);
% brute
fprintf('Text: %s\n', msg);
```

The output was miserable:

```
Text: ÿ
```

Then I tried to plot it in matlab:

```
[signal, Fs] = audioread('256.wav');

signal = signal(:,1);
N = length(signal);
X = fft(signal);

log_magnitude_spectrum = log(abs(X));
rceps_signal = ifft(log_magnitude_spectrum);

t = (0:N-1)/Fs;
plot(t, rceps_signal);
xlabel('Time (s)');
ylabel('Cepstrum');
title('Cepstrum of the Audio Signal');
xlim([0 0.05]);
```

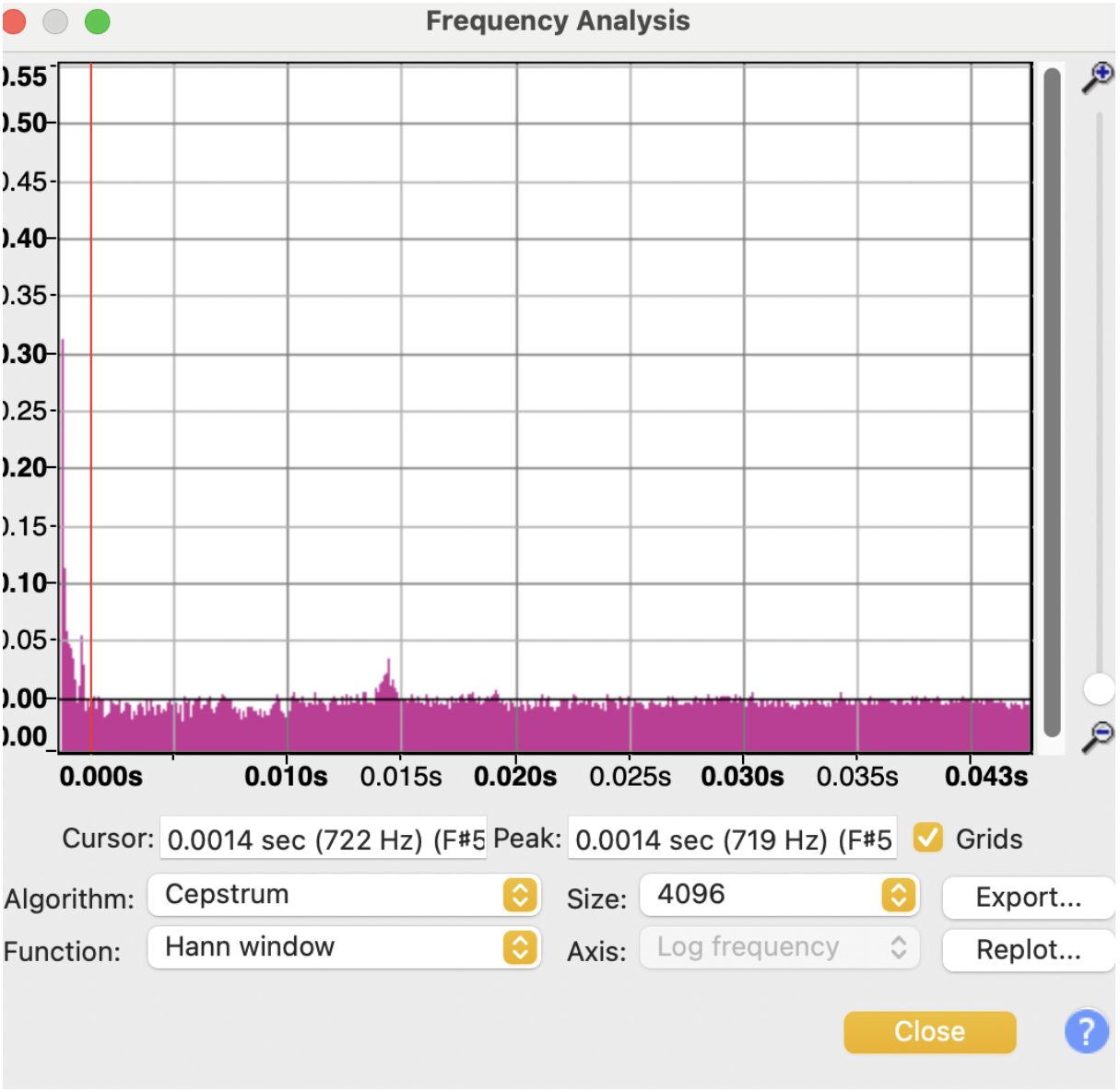


which is still lead to no where. I was even thinking about brute force the prams (but ended up too lazy to wrote for it)

The real prams

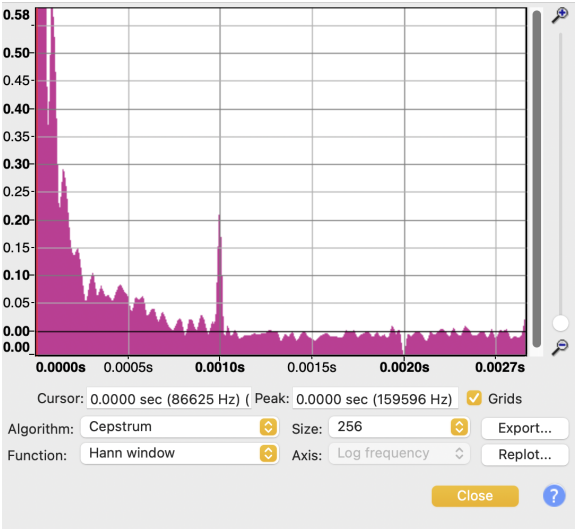
welp I spend tons of time trying to writing scripts or finding ways to find where is the wav. But never realize that the echo might not be repeated with each time music repeat.

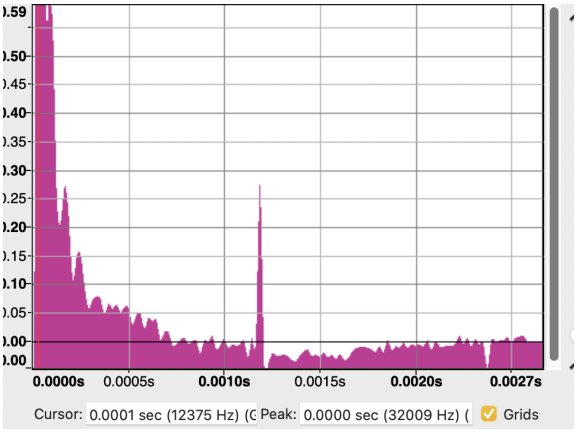
@SuperBettleGammer imported it into Audacity > Analyze > Plot Spectrum



Still looks like nothing, @Eth007 split it up into 0.1 sec intervals and saw where the spike changed sliced, and found out that frame window is actually 0.35, and the size is 256 according to file name. (I took little bit time look after challenge

solved, 0.35 actually is just one beat just finish). Every information until now is used.





And now magically the spikes comes up. Where it centers around 0.001s and 0.012s.

Now with these information, plug back into the script:

$$\mathrm{L} = 48000 \bullet 0.35 = 16800$$
$$d0 = 0.001 * 48000 = 48$$
$$d1 = 0.0012 * 48000 = 57.6 = 58$$

Final Script:

```
close all; clear all; clc;

audio = audioload();

msg = echo_dec(audio.data, 16800, 48, 58);

fprintf('Text: %s\n', msg);
```

Output from the script: `Text: ãsawctf{1nv3st_1n_s0undpr00F1ng}`

Yay!!!

final flag: `csawctf{1nv3st_1n_s0undpr00F1ng}`