

AUDIO STREAMS IN SNAPCHAT  
ACROSS DIFFERENT MOBILE OPERATING SYSTEMS

by

DAVID ARIAS

B.A., California State University of Dominguez Hills, 2018

A thesis submitted to the  
Faculty of the Graduate School of the  
University of Colorado in partial fulfillment  
of the requirements for the degree of  
Master of Science  
Media Forensics Program

2021

© 2021

DAVID ARIAS

ALL RIGHTS RESERVED

This thesis for the Master of Science degree by

David Arias

has been approved for the

Media Forensics Program

by

Catalin Grigoras, Chair

Gregory Wales

Cole Whitecotton

Date: December 18, 2021

Arias, David (M.S., Media Forensics Program)

Audio Streams in Snapchat Across Different Mobile Operating Systems

Thesis directed by Associate Professor Catalin Grigoras

### **ABSTRACT**

Snapchat has grown in popularity among teens and young adults since its inception in 2011. Originally released under the name Picaboo on iOS, the popularity and intrigue with Snapchat came from the ability to send snaps, which can be either pictures or short videos and audio messages, that are typically accessible only once under certain settings set by the sender. Once the snap has been viewed by the receiver, they no longer have access to the file and the snap seemingly disappears from the user's device. This disappearing snap later went on to become the popular 24-hour stories on seemingly every social media app nowadays.

This research thesis analyzes the similarities and differences between the audio files when snaps are sent and received on device running different mobile operating systems. Snapchat was downloaded onto four mobile devices running different operating systems with the test administered sending Snaps from devices running iOS to iOS, iOS to Android, Android to iOS, and Android to Android. Snaps were first recorded, downloaded on the created device, and sent to the receiving device. On the receiving device they were opened, saved, and downloaded on the receiving device. This resulted in the one snap saved onto the device that created it and received it. All the files were then transferred to a MacBook Pro and saved onto an external hard drive to be thoroughly examined. Examinations of the audio files included comparisons between the files metadata, hash values, audio file structure anomalies.

The form and content of this abstract are approved. I recommend its publication.

Approved: Catalin Grigoras

## **DEDICATION**

This thesis is dedicated to my parents for their patience and support throughout the years.

## **ACKNOWLEDGEMENTS**

I thank my committee board for all their assistance, guidance, and constructive criticism. To Catalin Grigoras for his enthusiasm and ability to steer me in the right direction. To Gregory Wales for his knowledge on Digital Forensics and advice for working procedures. To Cole Whitecotton for his presence in the lab and being available with any issues that arose.

A special thanks to both Jeff Smith and Leah Haloin for their work throughout the entire program. To Jeff for the time he put into the lessons, lectures, and clear explanations. To Leah for her emails with critical information and deadline days throughout the entire program that helped me stay connected even from afar.

## TABLE OF CONTENTS

### CHAPTER

I.	INTRODUCTION.....	1
	Previous Research.....	1
II.	MATERIALS .....	4
III.	METHODOLOGY .....	6
	Methods .....	6
IV.	RESULTS.....	13
	Test 1: iOS to iOS.....	13
	Test 2: iOS to Android.....	22
	Test 3: Android to iOS.....	30
	Test 4: Android to Android.....	38
V.	CONCLUSIONS .....	47
	Future Research .....	47
	REFERENCES .....	48

## LIST OF TABLES

### TABLE

1. Android and iOS Devices .....	4
2. Computer Platforms.....	4
3. Software Programs and Versions .....	5
4. Audio Snap Tests 1 – 4.....	6

## LIST OF FIGURES

### FIGURE

1. Snapchat Sending Process .....	7
2. Saving Snaps .....	8
3. MATLAB Frequency Analysis Spectrogram Test .....	9
4. MATLAB Frequency Analysis PSD Test .....	10
5. MATLAB FFT Test .....	11
6. MATLAB Consecutive Zero Level Samples Test .....	11
7. iOS Created to iOS Received Audio Metadata .....	13
8. iOS Created Audio Spectrogram .....	14
9. iOS Received Audio Spectrogram.....	15
10. iOS to iOS Audio PSD .....	16
11. iOS Created Audio FFT.....	17
12. iOS Received Audio FFT .....	17
13. iOS Created Audio MATLAB Zero Level Samples .....	18
14. iOS Created Audio Adobe Audition Zero Level Samples .....	19
15. iOS Created Audio Hex Fiend Zero Level Samples .....	19
16. iOS Received Audio MATLAB Zero Level Samples .....	20
17. iOS Received Audio Adobe Audition Zero Level Samples .....	21
18. iOS Received Audio Hex Fiend Zero Level Samples .....	21
19. iOS Created to Android Received Audio Metadata .....	22
20. iOS Created Audio Spectrogram .....	23
21. Android Received Audio Spectrogram .....	24

22. iOS to Android Audio PSD .....	25
23. iOS Created Audio FFT .....	26
24. Android Received Audio FFT .....	26
25. iOS Created Audio Adobe Audition Zero Level Samples .....	27
26. iOS Created Audio Hex Fiend Zero Level Samples .....	28
27. Android Received Audio MATLAB Zero Level Samples .....	28
28. Android Received Audio Adobe Audition Zero Level Samples .....	29
29. Android Received Audio Hex Fiend Zero Level Samples .....	29
30. Android Created to iOS Received Audio Metadata .....	30
31. Android Created Audio Spectrogram .....	31
32. iOS Received Audio Spectrogram .....	32
33. Android to iOS Audio PSD .....	33
34. Android Created Audio FFT .....	34
35. iOS Received Audio FFT .....	34
36. Android Created Audio MATLAB Zero Level Samples .....	35
37. Android Created Audio Adobe Audition Zero Level Samples .....	36
38. Android Created Audio Hex Fiend Zero Level Samples .....	36
39. iOS Received Audio MATLAB Zero Level Samples .....	37
40. iOS Received Audio Adobe Audition Zero Level Samples .....	37
41. iOS Received Audio Hex Fiend Zero Level Samples .....	38
42. Android Created to Android Received Audio Metadata .....	39
43. Android Created Audio Spectrogram .....	40
44. Android Received Audio Spectrogram .....	41

45. Android to Android Audio PSD .....	41
46. Android Created Audio FFT .....	42
47. Android Received Audio FFT .....	42
48. Android Created Audio MATLAB Zero Level Samples .....	43
49. Android Created Audio Adobe Audition Zero Level Samples .....	44
50. Android Created Audio Hex Fiend Zero Level Samples .....	44
51. Android Received Audio MATLAB Zero Level Samples .....	45
52. Android Received Audio Adobe Audition Zero Level Samples .....	45
53. Android Received Audio Hex Fiend Zero Level Samples .....	46

## **LIST OF ABBREVIATIONS**

OS – Operating System

App / Apps – Application

PSD – Power Spectral Density

CC – Correlation

MQD – Medium Quadratic Difference

FFT – Fast Fourier Transform

NFFT – FFT Order

# CHAPTER I

## INTRODUCTION

The world we know today is connected now more than it has ever been in our short history. Almost every person on this planet holds in their possession a mobile cellular device that allows them to connect to other people across the world. Mobile device applications known as social media apps facilitate connecting with distant friends and relatives on the fly. Though many social media apps have come and gone, there are those who stand the test of time such as Facebook, Twitter, and Instagram. There is one app that arose from Santa Monica, California that took social media by storm with its ingenuity, and that was Snapchat. Initially released under the name Picaboo in 2011, Snapchat has come to challenge the stereotype of “once it’s been on the internet, it’s there forever.” The allure Snapchat granted came from creating pictures or short videos called “snaps” that can be sent to friends, but once the Snap has been viewed it disappears and is, basically, no longer accessible. This feature when first released garnered interest from users across the age spectrum. Snapchat is nowadays most prominently used by teenagers and young adults who enjoy having the fail-safe of disappearing snaps and messages.

This thesis will focus on the audio aspect of sending and receiving snaps sent between different mobile devices. Forensically examining the audio metadata, stream hash values, and consecutive zero level errors when different mobile OS are in play. This analysis will help distinguish aspects of different audio files among the most used mobile devices on the market.

### **Previous Research**

There have been many papers published about social media apps and what occurs to messages, videos, and audio when uploaded and downloaded. Snapchat’s intrigue for its disappearing videos, audio, and messages has been a leading topic to research for years as the

question remains as to what occurs after the files are accessed by the user. Before even attempting to recover anything forensically, you can recover data directly from Snapchat by simply requesting it. A few key pieces of information are required though, such as the Snapchat username/email address and password. Also access to the same email account is needed to download the data after it has been sent. Snapchat does store all the data sent by the user for different periods of time. Key information that is available and retrievable includes login history, account information, snap history, memories, and saved chat history.

A paper published in the Journal of Theoretical and Applied Information Technology in 2017 titled *The Digital Forensic Analysis of Snapchat Application using MXL Records* attempted to search for any files that had previously been believed to have been deleted by the application. Digitally forensically searching for accessed Snaps is extremely different from searching for deleted messages or emails. Prior to analyzing audio files, this paper's examination also looked at Snapchat using another piece of forensic software titled Cellebrite UFED and Cellebrite Physical Analyzer. The results were similar as both examinations recovered information about snaps sent, received, and if they were accessed. Artifacts remained stored on the device that a user may believe don't exist anymore. These included start times, last activity, participants, timestamps, and source app information. The only drawback is that most of the information is either metadata, hex code, or files now reduced to cookies. This significantly reduces the capability of understanding what Snapchat does to Snaps after being sent. Thus, this approach was not pursued in greater detail.

In *A Comparison Analysis of Saved Snapchat Video Files on Android VS iPhone*, a paper published in 2021, Angela Rae Malley conducted a similar approach to what this paper attempts. In Malley's paper however, saved snaps were transferred through different methods such as

Dropbox, Gmail, and MMS. Notable results in Malley's study showed that snaps saved and transferred from Android files had no changes to the original files while iOS video files had hash value mismatches. As will be noticed later in this paper, the results in this study are not too different from those achieved by Ms. Malley. For instance, the results of her Android tests match results achieved in this study because the Android files matched the original file. The iOS results, however, had hash value mismatches in both our studies. One interesting aspect to note is that the Snapchat version used in Ms. Malley's paper and this paper are different. Her paper states Android Snapchat v11.15.1.34 and iOS Snapchat v11.15.0 were used, while the version used here was v.11.44.0.37 for Android and v11.45.1 for iOS. It would be of interest to go through Snapchat's updated patch notes to analyze what potential effects are done to keep consistent results between versions.

## CHAPTER II

### MATERIALS

To accurately conduct the tests for this research thesis, several hardware and software equipment were required. In terms of hardware, four mobile devices were needed with two devices running iOS software and the other two devices running Android OS. The device's OS was checked for any pending updates meanwhile the Snapchat app was also downloaded and checked for any further updates. Two computers were used to gather and analyze the data acquired from Snapchat, one system running Windows 10 for analyzing the cell phones and the other computer being a Macintosh for analyzing the audio data. The software used for this research is readily accessible online and most are open-source software. The testing of sending Snaps took place the day of September 15th, 16th, and the 29th. The audio file examinations took place on those days leading up to the first week of October.

*Table 1: Android and iOS Devices*

Make	Model	Model Number	Operating System	Snapchat Version
Apple	iPhone 7 A1778	MN9U2LL/A	iOS 14.8	v.11.45.1 Boo
Apple	iPhone 8 A1905	MQ6Y2LL/A	iOS 14.7.1	v.11.45.1 Boo
Samsung	Galaxy S20+ 5G	SM-G986U	Android 11	v.11.44.0.37
Samsung	Galaxy A12	SM-A125U	Android 11	v.11.44.0.37

*Table 2: Computer Platforms*

Make	Model	Operating System
Dell	XPS 7760 AIO	Windows 10 Enterprise v.21H1 19043.1237
Apple	MacBook Pro	macOS Catalina v.10.15.7

Table 3: Software Programs and Versions

Software	Version	Platform
Cellebrite UFED	v.7.45.0.96	Windows 10
Cellebrite Physical Analyzer	v.7.45.0.96	Windows 10
Adobe Audition 2020	v.13.0.13.46	macOS
MatLab R2021a	v.9.10.0.1684407	macOS
Terminal	v.2.10	macOS
ffmpeg	v.4.4	macOS
MediaInfo	v.21.09	macOS
Hex Fiend	v.2.14	macOS
iHash	v.2.0.6	macOS

Cellebrite UFED is a software program that is used to lawfully access and collect digital data. The UFED database for mobile devices spans many current and previous devices that may still be in use. UFED works in conjunction with Cellebrite's Physical Analyzer for data examination extracted from the mobile device. This allows for deep dive searches into the device's previously inaccessible data. Adobe Audition was used as a non-destructive editing environment to analyze audio files for consecutive zero level samples. MathWorks MATLAB is a mathematical computing software that uses a command line to support data analysis and simulation. Various scripts were used to analyze the audio files singularly and collectively to interpret the results. FFMPEG is an open-sourced command-line software that is used to handle videos, audio, and other types of multimedia files. This software was runs in Mac's terminal on a MacBook to extract audio files and examine them as well. MediaInfo is an open-source tool used to display technical information and metadata about media files. Hex Fiend, another open-source software, is a simple hex editor that allows for inserting, deleting, and rearranging hex data. iHash was used to obtain the Hash Values of the audio files to verify them with the values obtained from FFMPEG.

**CHAPTER III**  
**METHODOLOGY**

Four mobile devices were used, two of which ran iOS software and the other two ran Android software. The test ran as follows: iOS to iOS, iOS to Android, Android to iOS, and Android to Android. Audio messages were sent from the first device, referred to as created, and opened on the second device, referred to as received. The purpose of these initial messages was to run the phones through Cellebrite’s UFED and Physical Analyzer to search for audio data forensically and understand what Snapchat does with the files that are seemingly generated and disappear after being accessed. The results in this first experiment were unfavorable which lead to the core of this thesis in the form of recording Snaps on the creation device and allowing for the receiver to save the snap on the receiving end. This process allowed for the same Snap to manifest on two separate devices which would allow for comparison as to what occurs when Snaps are sent to devices running similar and different OS.

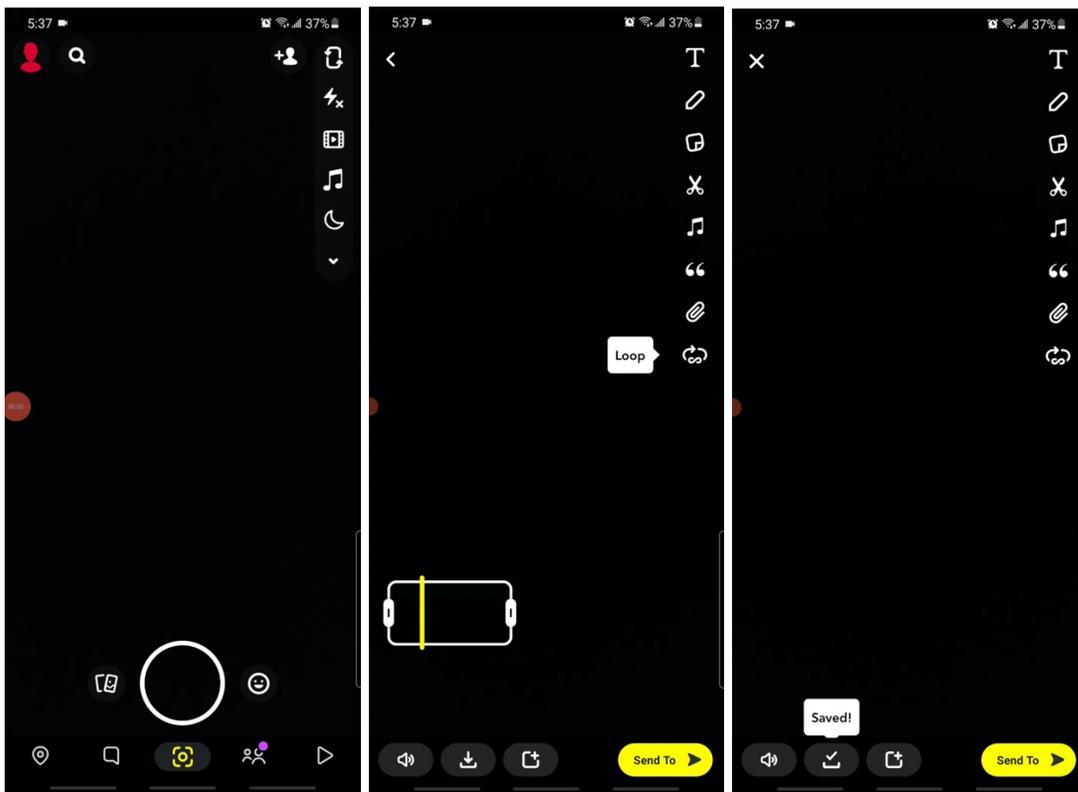
*Table 4: Audio Snap Tests 1 – 4*

Creation on	Received by
iPhone 7 running iOS	iPhone 8 running iOS
iPhone 8 running iOS	Samsung Galaxy S20+ running Android
Samsung Galaxy S20+ running Android	iPhone 8 running iOS
Samsung Galaxy A12 running Android	Samsung Galaxy S20+ running Android

**Methods**

A Snapchat account was created on every mobile device and was connected to the other accounts to be able to chat with one another. The next step was to send a snap to the other mobile devices following the order of testing that was previously established. Even on different OS

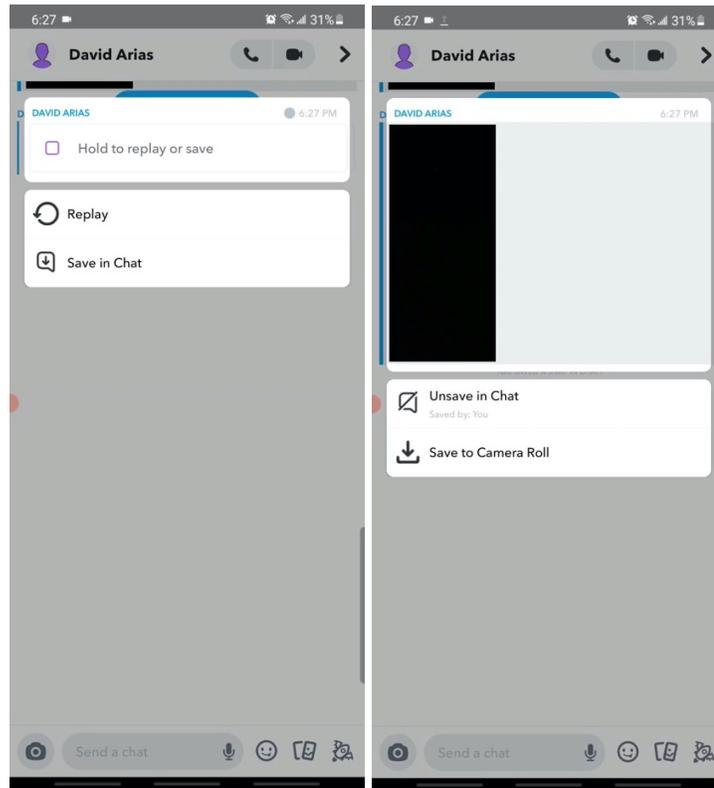
devices the method of sending a snap is the same. Upon opening the app, it is a simple matter of holding the circular record button at the base of the screen and recording for however long you desire. After the recording process is finished, the app will present you with the snap, in a looped manner, so the user can edit the image, video, and audio. For the purpose of these tests, blacked out videos were recorded and left on the loop setting. This setting allows for the receiver to save the video in their chat and save it onto their device. Prior to sending the snap, Snapchat allows for the users to save the video as well, this will produce two copies of the same snap from the device that creates it and the device that receives the snap.



*Fig 1: Snapchat Sending Process*

On the receiving device, the user would open Snapchat and navigate to the chat section of the app and watch the looped snap they were sent. After closing the video, the user can save the video into the chat and subsequently save the video onto their devices storage. This process

describes how one created snap now exists on two separate devices for examination. This process was repeated for the four tests using different devices. The saved videos were then transferred to a MacBook using stock USB cables for storage on an external hard drive.



*Fig 2: Saving Snaps*

Using different methods and techniques, the files were then analyzed to understand the effects occurring when sending the snap from one device to another. The audio files were extracted from the video files using the FFmpeg script `ffmpeg -i in.mov -vn -acodec pcm_s16le -ar 44100 -ac 1 aud.wav` (in.mov and aud.wav are stand in names for the actual files used). Testing began by examining the hash values of each pair of WAV files along with examining the files metadata. The eight audio files were then ran using various scripts in MATLAB including frequency spectrogram, frequency spectrum, FFT, and Data Energy Zero tests to examine the

audio spectrograms and energy levels against one another. The MATLAB Frequency Analysis Spectrogram Test produces a visual representation of the spectrum of frequencies in a sound or signal as it varies with time that is computed using short-time FFT. The graph will display frequencies on the vertical axis and time on the horizontal axis. Raising the FFT order to a higher value is recommended for frequency measurements, meanwhile a lower FFT order is recommended for time analysis events.

```
figure
[x,fs]=audioread('aud08.wav');
t=(0:length(x)-1)./fs;
subplot(211)
plot(t,x,'k'), grid on
xlabel('time [sec'],'FontSize',10),
ylabel('Amplitude [V'],'FontSize',10)
title('Waveform','FontSize',10,'FontWeight','normal')
axis([0 max(t) min(x).*1.1 max(x).*1.1])
subplot(212)
% here we change the settings
spectrogram(x,round(fs/100),round(fs/200),1024,fs,'yaxis'),
grid on,
xlabel('time [sec'],'FontSize',10),
ylabel('Frequency [KHz'],'FontSize',10),
title('Spectrogram','FontSize',10,'FontWeight','normal')
```

*Fig 3: MATLAB Frequency Analysis Spectrogram Test*

To compare the created and received audio signals against one another, a Power Spectral Density, or PSD, test was administered. This test will plot the signals on a graph presenting magnitude (dB) versus frequency (Hz), along with a visual graph. This test also establishes and displays the Correlation, CC, and the Mean Quadratic Difference (MQD) between the audio signals. The CC is a value between 0 and 1 with values ranging in the decimals between the two whole numbers. The higher the values between the two indicates a higher CC. The same can be said for the MQD as a higher numerical value signifies better constancy between the signals. It is worth noting that the equation for MQD contains a zero in the denominator, that depending on

the files used, may produce a negative infinite, -inf, as a result. This is no error but an indication of consistent results between files.

```

% PSD CC & MQD
clear all; fclose all;
% first case
% aud07.wav and aud08.wav were generated by
% the same WAV PCM device
[x,fs]=audioread('aud07.wav');
[y,fs]=audioread('aud08.wav');
% PSDs
[PSDx,Fxx]=periodogram(x,rectwin(length(x)),512,fs);
PSDx=10*log10(PSDx);
plot(Fxx,PSDx,'k');
grid on;
hold on;
[PSDy,Fyy] = periodogram(y,rectwin(length(y)),512,fs); PSDy=10*log10(PSDy);
plot(Fyy,PSDy,'b');
grid on;
hold on;
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectral Density Estimate');
axis('tight');
% mean value of x (x can be vector or matrix)
meanx=sum(PSDx:)/prod(size(PSDx));
% mean value of y (y can be vector or matrix)
meany=sum(PSDy:)/prod(size(PSDy));
% remove DC component
PSDx=PSDx-mean(PSDx);
PSDy=PSDy-mean(PSDy);
% compute the correlation coefficient cc
cc_1=sum(sum(PSDx.*PSDy))/sqrt(sum(sum(PSDx.*PSDx))*sum(sum(PSDy.*PSDy)))
% compute the MQD
mqd_log_1=log10(sqrt((sum((PSDx(:)-PSDy(:)).^2))./length(PSDx)))

clear all;
figure
% second case
% aud07.wav was generated by a WAV PCM device
% aud08.wav is a WAV PCM version of the above file
[x,fs]=audioread('aud07.wav');
[y,fs]=audioread('aud08.wav');
% PSDs
[PSDx,Fxx]=periodogram(x,rectwin(length(x)),512,fs); PSDx=10*log10(PSDx);
plot(Fxx,PSDx,'k');
grid on;
hold on;
[PSDy,Fyy] = periodogram(y,rectwin(length(y)),512,fs);
PSDy=10*log10(PSDy);
plot(Fyy,PSDy,'b');
grid on;
hold on;
xlabel('Frequency (Hz)');
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectral Density Estimate');
axis('tight');
% mean value of x (x can be vector or matrix)
meanx=sum(PSDx:)/prod(size(PSDx));
% mean value of y (y can be vector or matrix)
meany=sum(PSDy:)/prod(size(PSDy));
% remove DC component
PSDx=PSDx-mean(PSDx);
PSDy=PSDy-mean(PSDy);
% compute the correlation coefficient cc
cc_2=sum(sum(PSDx.*PSDy))/sqrt(sum(sum(PSDx.*PSDx))*sum(sum(PSDy.*PSDy)))
% compute the MQD
mqd_log_2=log10(sqrt((sum((PSDx(:)-PSDy(:)).^2))./length(PSDx)))

```

*Fig 4: MATLAB Frequency Analysis PSD Test*

The third MATLAB test administered to all the files was FFT script which converts signals from time to frequency domain and vice versa. The FFT order, or NFFT, can be increased to improve accuracy and measurement uncertainty reduced.

```
clear all;
% read the aud08.wav file
% select the working folder
dir1=uigetdir;
% change directory to working folder
cd(dir1);
% select the vowels.wav file
[name1,path1]=uigetfile('*.wav');
% load the audio file
[x,Fs]=audioread(name1);
subplot(211),
plot(x,'k'),
xlabel('Samples'),
ylabel('Amplitude'),
grid on,
title('Signal x'),
axis([0 length(x) min(x) max(x)])
subplot(212)
spectrogram(x, 256, 128, 256, Fs,'yaxis')
```

*Fig 5: MATLAB FFT Test*

```
% select the audio file
[name1,path1]=uigetfile({'*.wav'; '*.mp3'; '*.wma'}, 'Select a WAV, MP3, WMA', 'Multiselect', 'on');
% cd to the file's folder
cd(path1);
% read the info
info=audioinfo(name1);
NC=info.NumChannels;
N=info.TotalSamples;
% load the file data and sampling frequency
[x,fs]=audioread(name1);
% if the file is stereo stop/return
if NC==2
    disp('This is a stereo file, load a mono file.');
```

```
    return
% if the file is mono then continue to detect zeros and plot the data and power
elseif NC==1
    title1=[name1];
    % check if the first sample is zero
    if x(1)==0
        % create kz to count zeros
        kz=1;
        for k1=2:N-1
            if x(k1)==0 && x(k1+1)==0
                kz=kz+1;
            elseif x(k1)==0 && x(k1+1)~=0
                break
            end
        end
        title1=[name1];
        text1=[num2str(kz-1), ' consecutive zero level samples detected at the beginning of the signal.'];
        disp(text1);
        plot(x(1:kz*2), 'k'),
        xlabel('samples'),
        ylabel('Amplitude'),
        grid on,
        hold on
        plot(1:kz,x(1:kz), 'b', 'LineWidth', 2)
        axis([0 kz*2 min(x(1:kz*2)) max(x(1:kz*2))])
    end
end
if exist('kz')==0
    disp('No consecutive zero level samples detected.')
end
```

*Fig 6: MATLAB Consecutive Zero Level Samples Test*

The Consecutive Zero Levels Samples test was used to indicate traces of lossy compression or some kind of discontinuities. This script is meant to cut away with the time-consuming approach of searching manually for the number of consecutive zeros. A file with no consecutive zeros will display a message saying “No consecutive zeros level samples detected” which did occur during this testing. The remaining tests did encounter zero level samples and the number was reported along with a visual display of a blue line at the beginning of the signal. Finally, to confirm the number of consecutive zero level samples, the audio files were examined in Adobe Audition and Hex Fiend. Using the numbers provided by MATLAB.

## CHAPTER IV

### RESULTS

#### Test 1: iOS to iOS

After acquiring both snaps from their respected mobile devices and separating the audio files, the first analytical step was the examination of the files hash values and metadata. Using iHash, both files were tested under the SHA256 hash function with the resulting hashes being:

Created SHA256:

c180cd6891d4b5f2042411f77b866c57d07bebfba9b0eb4894024970503e3e07

Received SHA256:

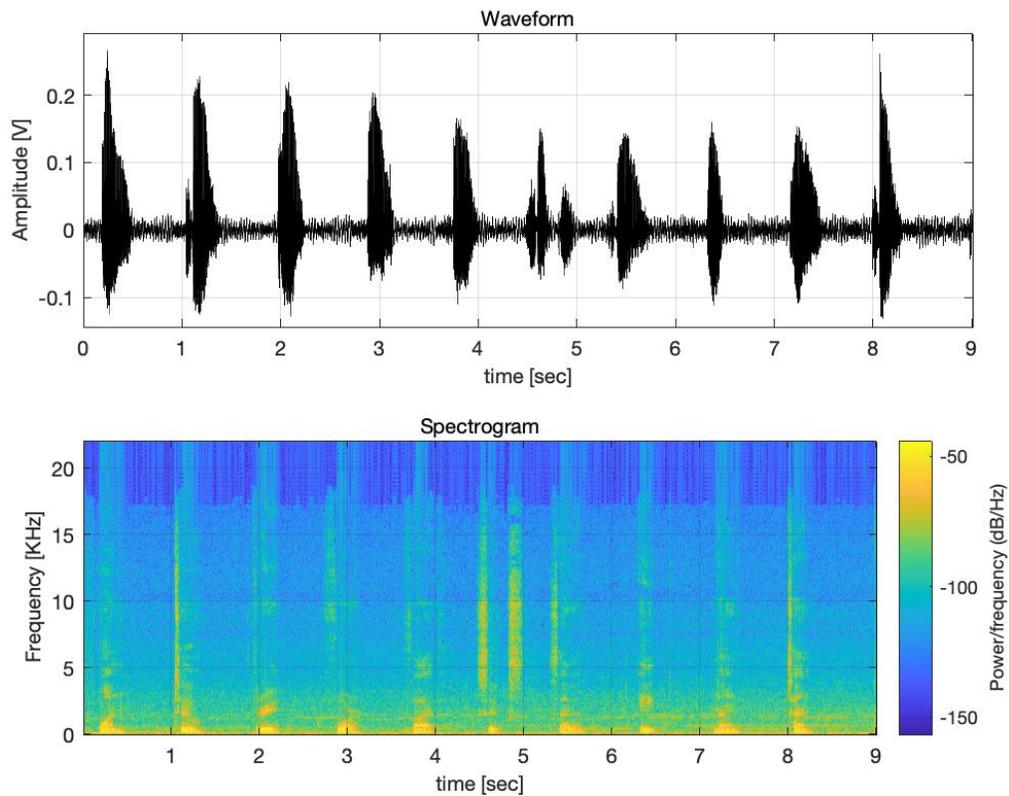
3ff222599143938c16fa35a5d77ec6089e64463145a0869532d13341291c1cd7

The first observations indicated the changes that occur when sending a snap from one iOS device to another iOS device. The results of the metadata produce similar results but with minor changes.

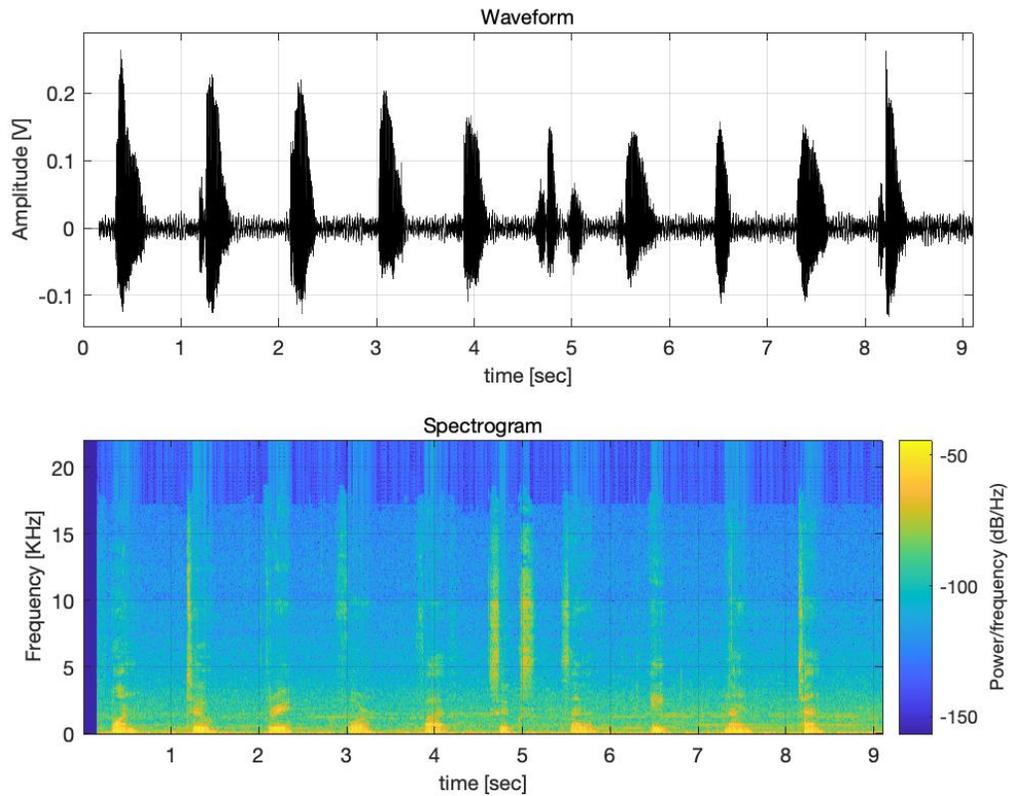
<b>Audio</b>		<b>Audio</b>	
<i>ID :</i>	2	<i>ID :</i>	1
<i>Format :</i>	AAC LC	<i>Format :</i>	AAC LC
<i>Format/Info :</i>	Advanced Audio Codec	<i>Format/Info :</i>	Advanced Audio Codec
<i>Codec ID :</i>	mp4a-40-2	<i>Codec ID :</i>	mp4a-40-2
<i>Duration :</i>	9 s 133 ms	<i>Duration :</i>	9 s 100 ms
<i>Source duration :</i>	9 s 79 ms	<i>Source duration :</i>	9 s 195 ms
<i>Bit rate mode :</i>	Constant	<i>Bit rate mode :</i>	Constant
<i>Nominal bit rate :</i>	98.0 kb/s	<i>Bit rate :</i>	98.0 kb/s
<i>Channel(s) :</i>	1 channel	<i>Channel(s) :</i>	1 channel
<i>Channel layout :</i>	C	<i>Channel layout :</i>	C
<i>Sampling rate :</i>	44.1 kHz	<i>Sampling rate :</i>	44.1 kHz
<i>Frame rate :</i>	43.066 FPS (1024 SPF)	<i>Frame rate :</i>	43.066 FPS (1024 SPF)
<i>Compression mode :</i>	Lossy	<i>Compression mode :</i>	Lossy
<i>Source stream size :</i>	106 KiB (2%)	<i>Stream size :</i>	107 KiB (10%)
<i>Title :</i>	Core Media Audio	<i>Source stream size :</i>	108 KiB (10%)
<i>Encoded date :</i>	UTC 2021-09-29 22:06:54	<i>Title :</i>	Core Media Audio
<i>Tagged date :</i>	UTC 2021-09-29 22:07:03	<i>Encoded date :</i>	UTC 2021-09-29 22:07:09
		<i>Tagged date :</i>	UTC 2021-09-29 22:07:10

*Fig 7: iOS Created to iOS Received Audio Metadata*

The duration and source duration of the snap seemingly changed as well as stream size and the source stream size. The percentage of the stream and source stream size signifies the percentage of the original size. Upon listening to the files and inspecting the metadata, a new theory unfolded as to why the received audio file had changed. The new hypothesis was that somewhere in the process of uploading the snap to Snapchat's server and downloading it back down onto the receiving device, there was a padding of zeros that occurred when transcoding. To visually view the differences between the two files a few MATLAB scripts were used. The first script ran against the audio files was the Frequency Analysis Spectrum test.

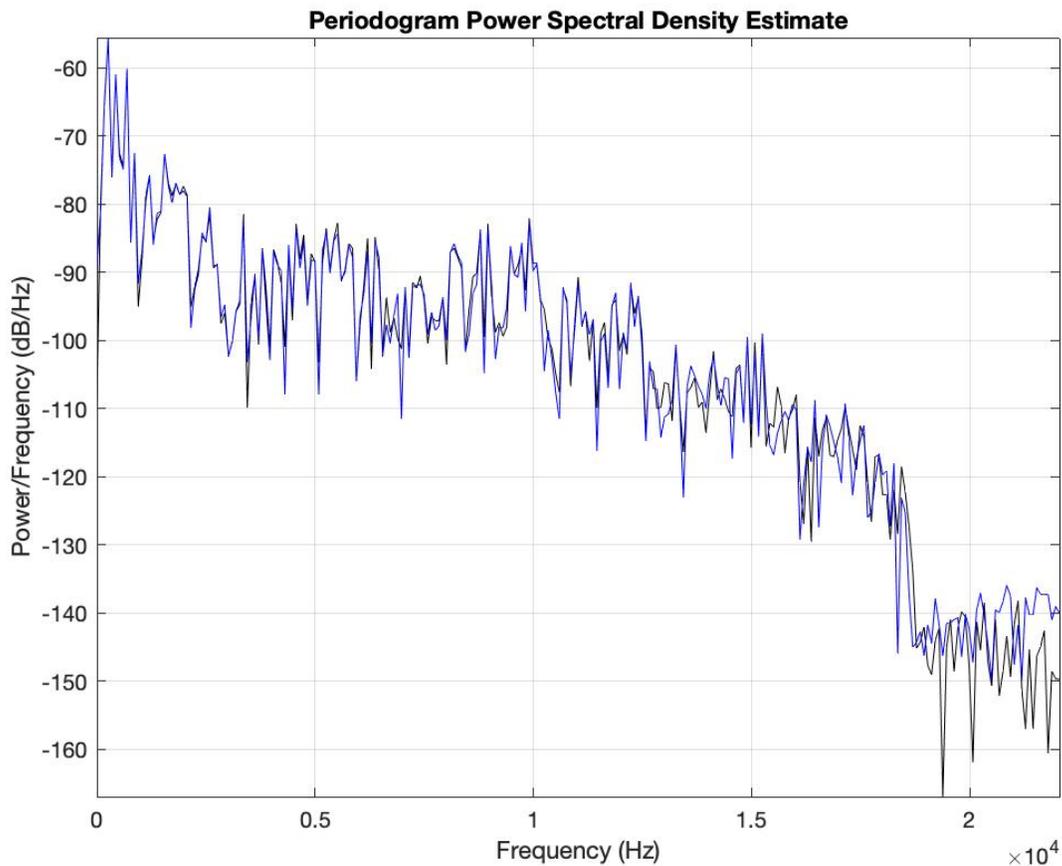


*Fig 8: iOS Created Audio Spectrogram*



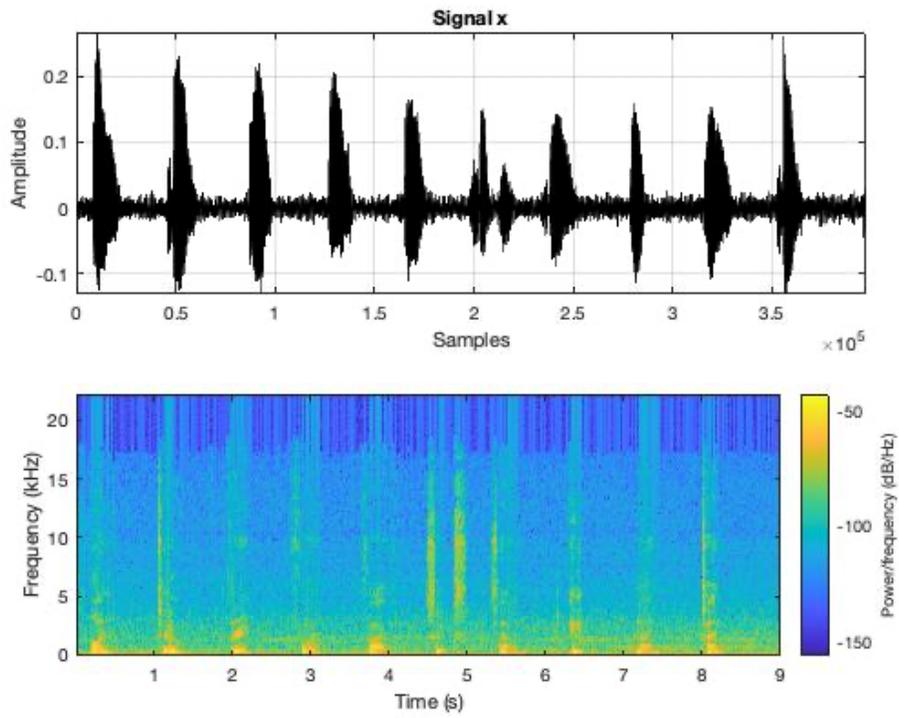
*Fig 9: iOS Received Audio Spectrogram*

The clear difference between these two spectrograms is the shifting of the audio file at the beginning in Fig 9. This is the potential zero level samples that arose and caused the file to be different from the original. The second test ran in MATLAB consisted of the Frequency Analysis Spectrum Test. This test provided unbiased results of the two audio files PSD by computing their CC and their MQD. The PSD is the representation of a signal with the resulting values presented as magnitude (dB) versus frequency (Hz). The results of the equations of CC and MQD will be used to interpret the consistency or inconsistency of the two audio files being examined.

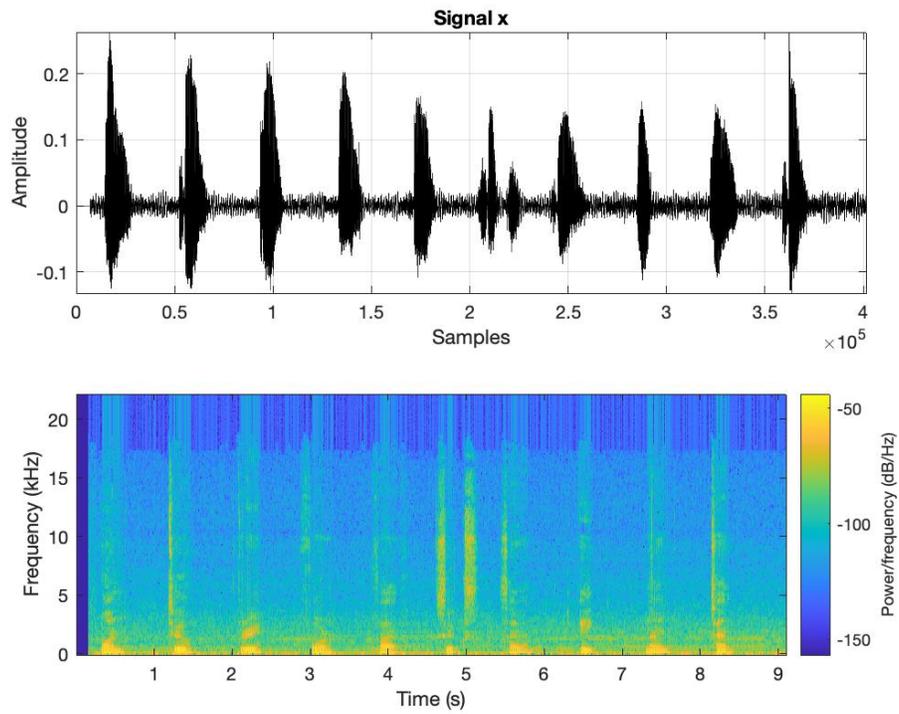


*Fig 10: iOS to iOS Audio PSD*

The resulting values for the CC was 0.9770 meanwhile the MQD results formulated to 0.6750. Although the files are different, their correlation demonstrates that there are consistencies between the files. The third script ran was the Fast Fourier Transform test. This algorithm converts signals from time to frequency domain and vice versa. The results introduced similar results as the initial spectrogram test, only using a different formula.

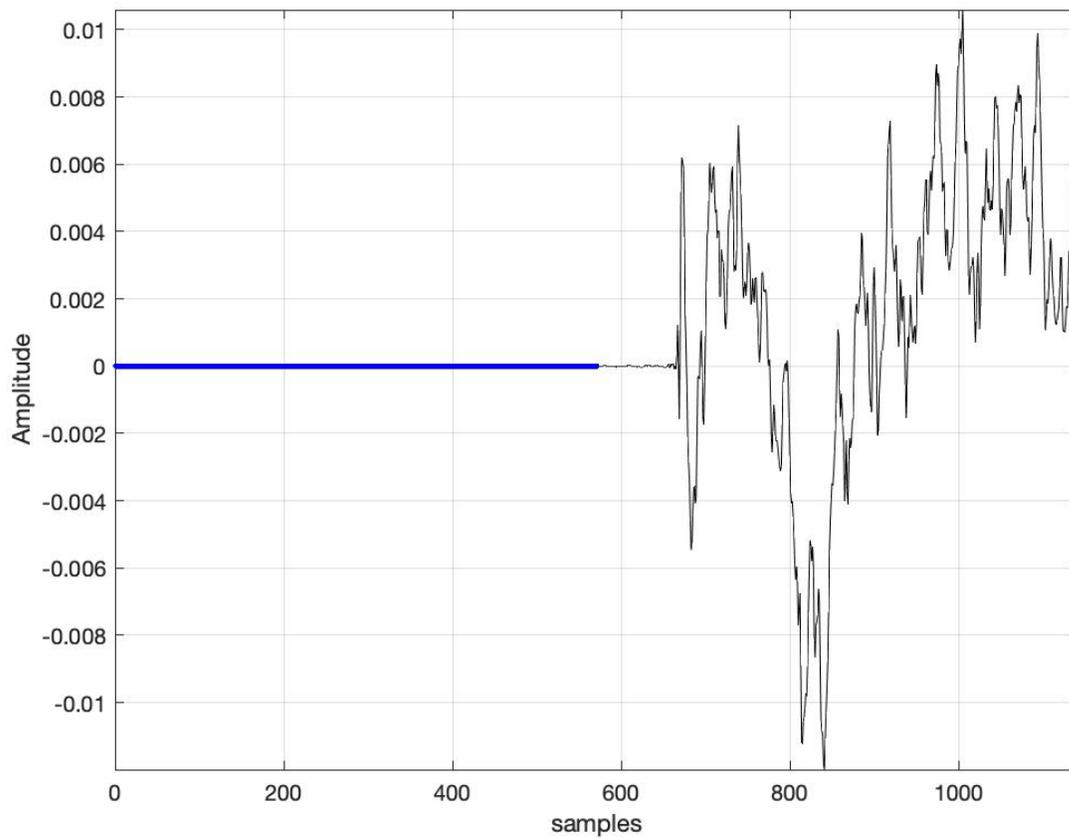


*Fig 11: iOS Created Audio FFT*



*Fig 12: iOS Received Audio FFT*

The final test administered in MATLAB consisted of a Consecutive Zero Level Samples test. This test usually indicates a corrupted audio signal, traces of lossy compression, or discontinuities. The script used for this test detects the zero level samples at the beginning of the signal and reports the said number. On the created file, MATLAB reported the number of zero level samples found in the created snap was 570. With that number in mind, a search was done in Adobe Audition and Hex Fiend, and it was determined that the true number of consecutive zero level samples was 572.



*Fig 13: iOS Created Audio MATLAB Zero Level Samples*



Fig 14: iOS Created Audio Adobe Audition Zero Level Samples

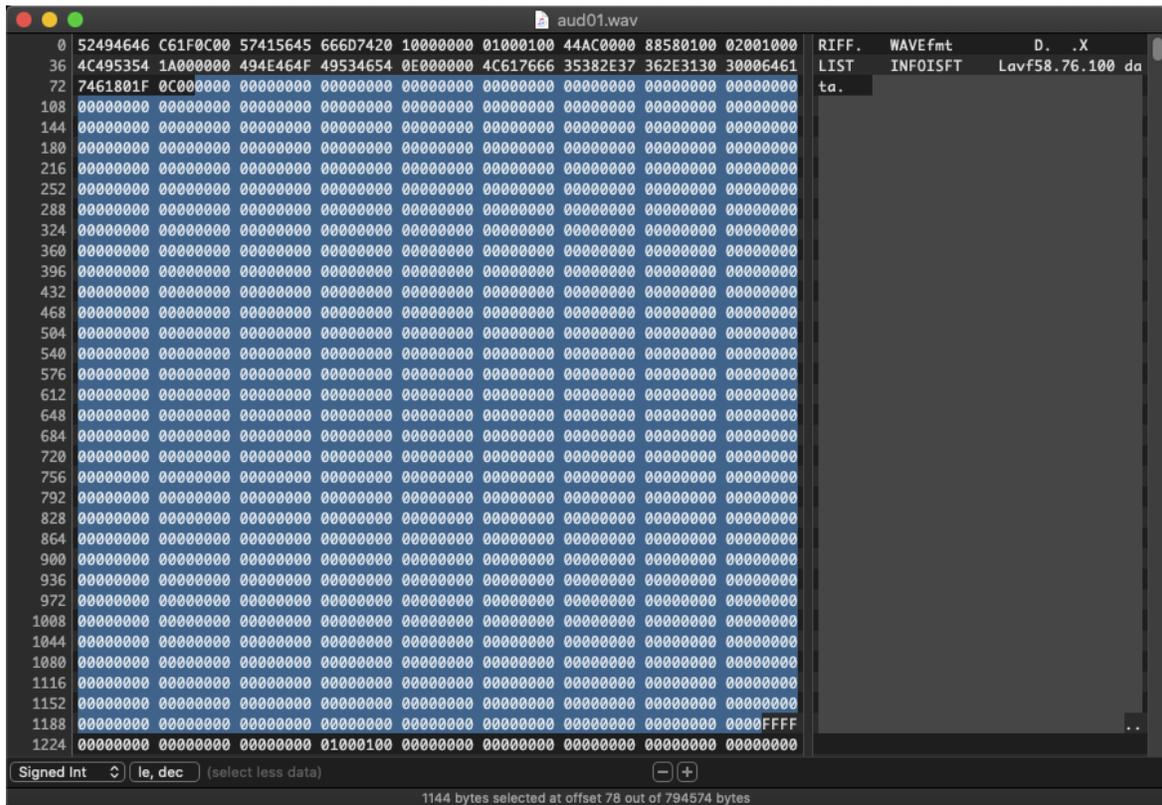
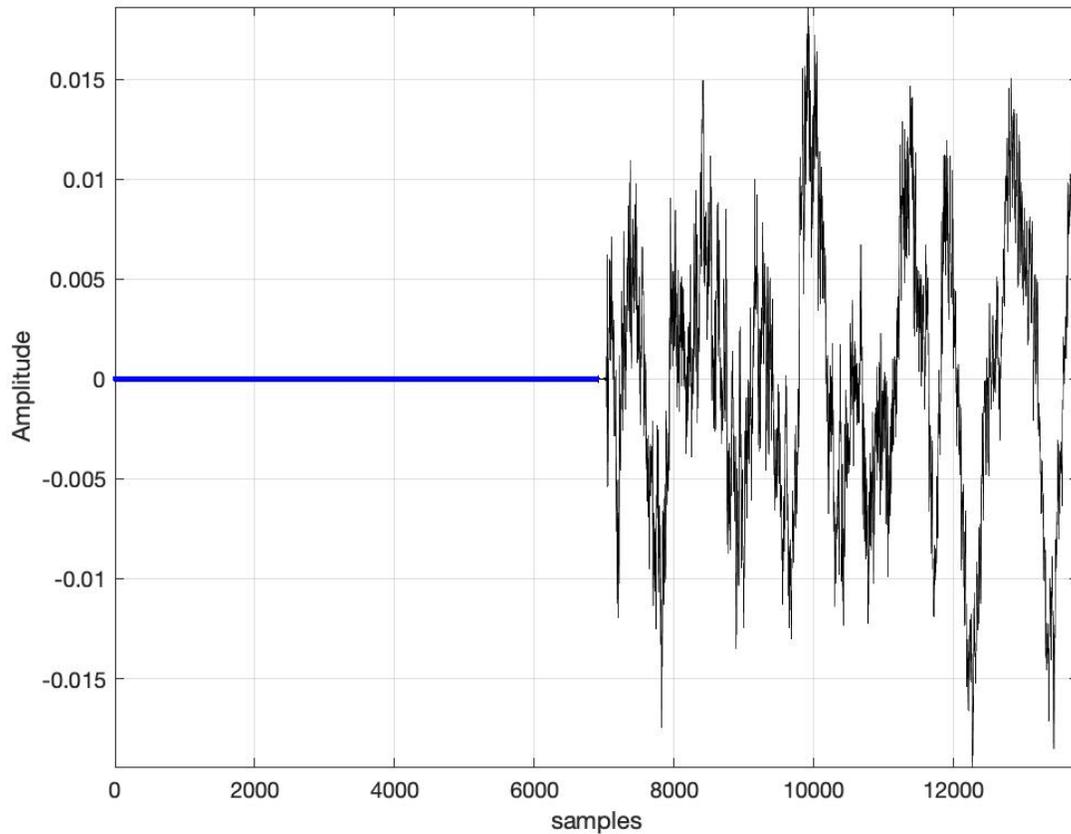


Fig 15: iOS Created Audio Hex Fiend Zero Level Samples

In Fig. 14, the samples are displayed in Adobe Audition with the highlighted portion accounting for 2 more samples that MATLAB did not count bring the total number up to 572 consecutive zero samples. In Hex Fiend, the number of bytes selected equals 1144 which is equal to the 572 samples encountered in Audition. The Received audio file however had significantly more samples present as shown in the spectrogram. MATLAB reported 6891 zero level samples found in the received audio file with the number being 6893 samples when viewed in Audition and Hex Fiend. Although the snap was created on an iOS device and received on another iOS device, the two snaps have different hash values, differentiating metadata and the audio files have varying number of zero level samples.



*Fig 16: iOS Received Audio MATLAB Zero Level Samples*



Fig 17: iOS Received Audio Adobe Audition Zero Level Samples

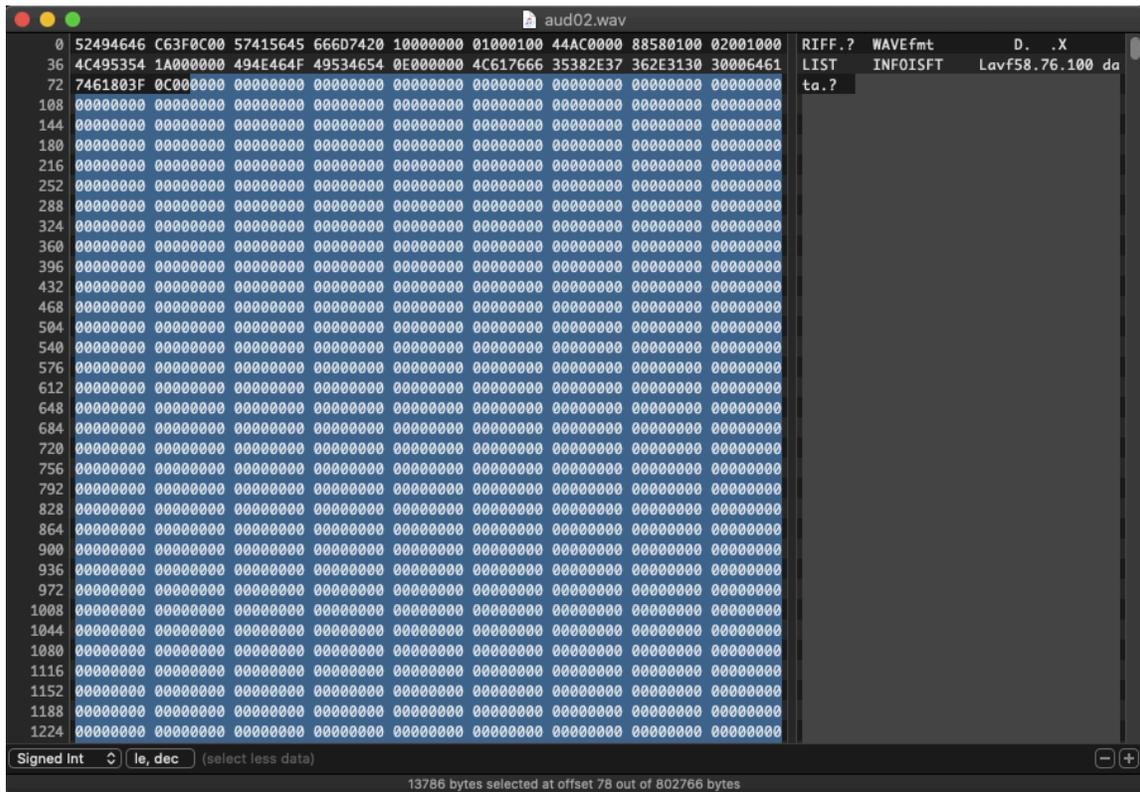


Fig 18: iOS Received Audio Hex Fiend Zero Level Samples

## Test 2: iOS to Android

Much like in the first test, there were minimal changes within the metadata of the files which did result in different sets of hash values. The changes in the metadata continued to be the duration, source duration, stream size, and source stream size. The hashes are as follows:

Created SHA256:

badad21f18eabfb766afc051eb123e45df8bff51c7c322ff669255a59b059771

Received SHA256:

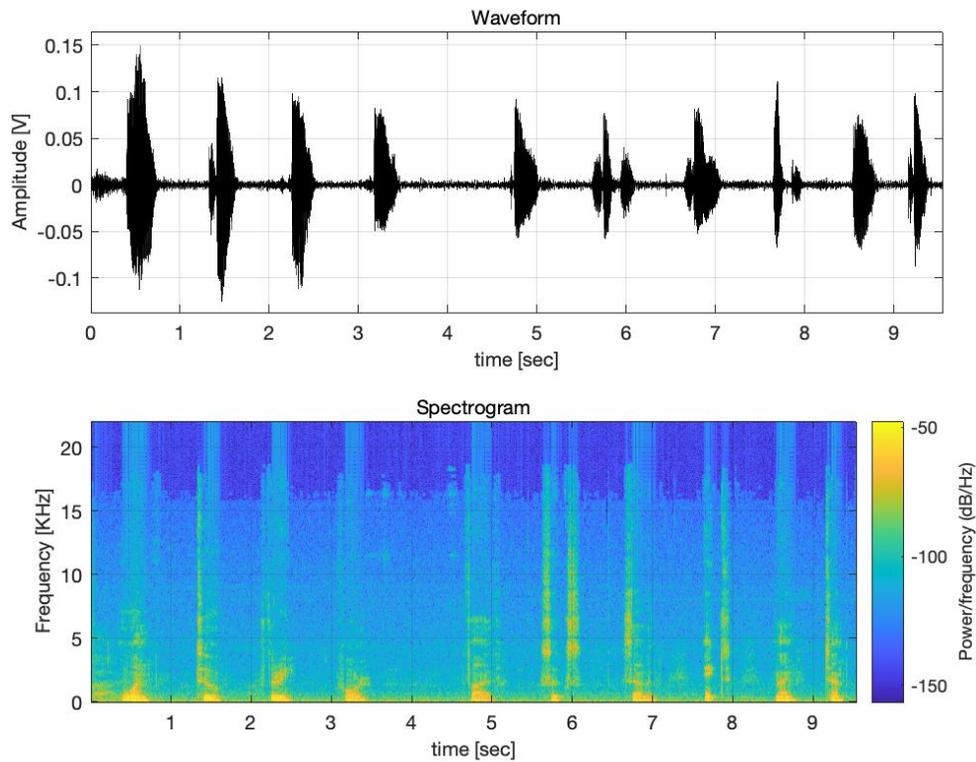
fd92c89b35f650fe1fe01fb46265b39a4342eb9c5b80b616a097f009c290800c

<b>Audio</b>		<b>Audio</b>	
<i>ID :</i>	2	<i>ID :</i>	1
<i>Format :</i>	AAC LC	<i>Format :</i>	AAC LC
<i>Format/Info :</i>	Advanced Audio Codec	<i>Format/Info :</i>	Advanced Audio Codec
<i>Codec ID :</i>	mp4a-40-2-2	<i>Codec ID :</i>	mp4a-40-2
<i>Duration :</i>	9 s 538 ms	<i>Duration :</i>	9 s 731 ms
<i>Source duration :</i>	9 s 660 ms	<i>Source duration :</i>	9 s 799 ms
<i>Bit rate mode :</i>	Constant	<i>Bit rate mode :</i>	Constant
<i>Bit rate :</i>	98.0 kb/s	<i>Bit rate :</i>	98.0 kb/s
<i>Channel(s) :</i>	1 channel	<i>Channel(s) :</i>	1 channel
<i>Channel layout :</i>	C	<i>Channel layout :</i>	C
<i>Sampling rate :</i>	44.1 kHz	<i>Sampling rate :</i>	44.1 kHz
<i>Frame rate :</i>	43.066 FPS (1024 SPF)	<i>Frame rate :</i>	43.066 FPS (1024 SPF)
<i>Compression mode :</i>	Lossy	<i>Compression mode :</i>	Lossy
<i>Stream size :</i>	112 KiB (11%)	<i>Stream size :</i>	114 KiB (24%)
<i>Source stream size :</i>	113 KiB (12%)	<i>Source stream size :</i>	114 KiB (24%)
<i>Title :</i>	Core Media Audio	<i>Title :</i>	Core Media Audio
<i>Encoded date :</i>	UTC 2021-09-29 22:33:17	<i>Encoded date :</i>	UTC 2021-09-29 22:33:35
<i>Tagged date :</i>	UTC 2021-09-29 22:33:27	<i>Tagged date :</i>	UTC 2021-09-29 22:33:35

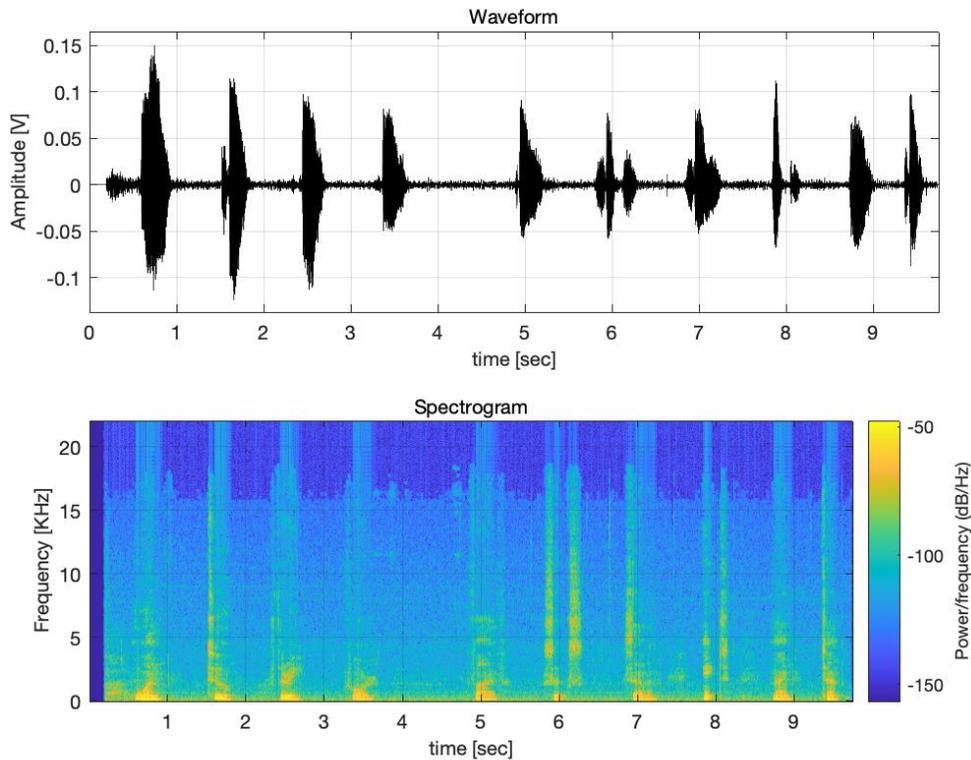
*Fig 19: iOS Created to Android Received Audio Metadata*

The duration of the snap seemingly increases in length on the received device in comparison to the created device. It will later be noted that the receiving device has had a padding of zeros added to the beginning of the file while the created device had no such padding. This is the only case where one file has no padding while the other file does. When examining

the audio files spectrogram in MATLAB, the shift in duration becomes apparent as well as the padding of the received audio file. Much like in the first test from iOS to iOS, the created audio file appeared to show little to no padding and was later confirmed that it did. Examining the spectrogram on this file, the created device appears to have none meanwhile the received device does. That's the second indication, behind the hash values, that the files are not exactly the same.

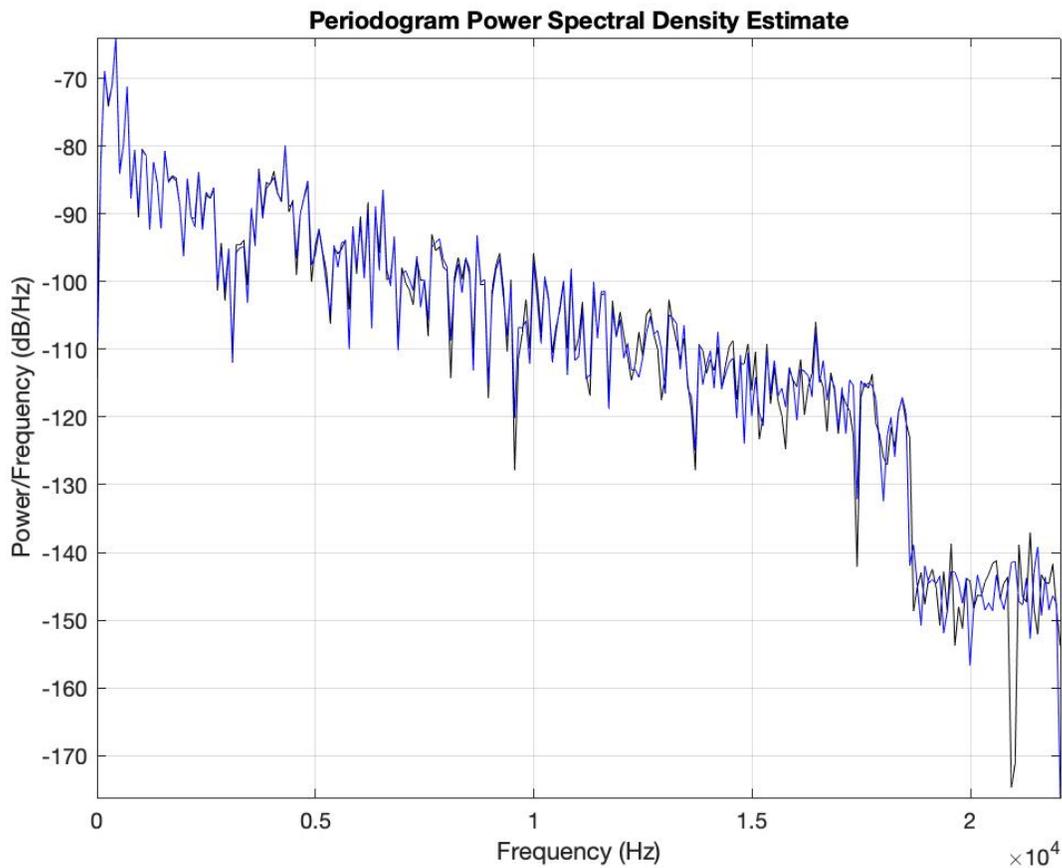


*Fig 20: iOS Created Audio Spectrogram*



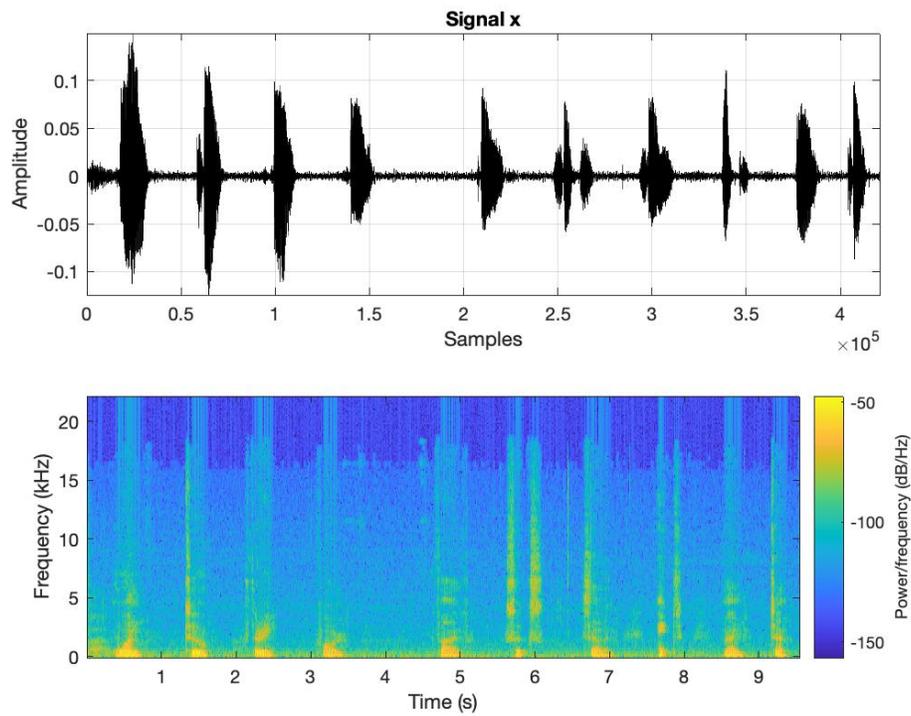
*Fig 21: Android Received Audio Spectrogram*

The Frequency Analysis Spectrum test displayed the PSD of the audio files and calculated the CC and MQD of the files, the resulting numbers being 0.9736 CC and 0.6646 MQD. Comparing these numbers to the results obtained in the first test, it can be determined that these audio files are slightly less consistent from iOS to Android. Conclusions can begin to be drawn to say that creating snaps on iOS devices will lead to inconsistent results when viewing them on another device regardless of OS the receiver has. The CC of the first test and this test is a drop of 0.0034 meanwhile the MQD drop off is of 0.0104. The only major difference between the two tests was the receiving device, which is cause for concern because the third and fourth tests don't display these types of changes as will be seen later.

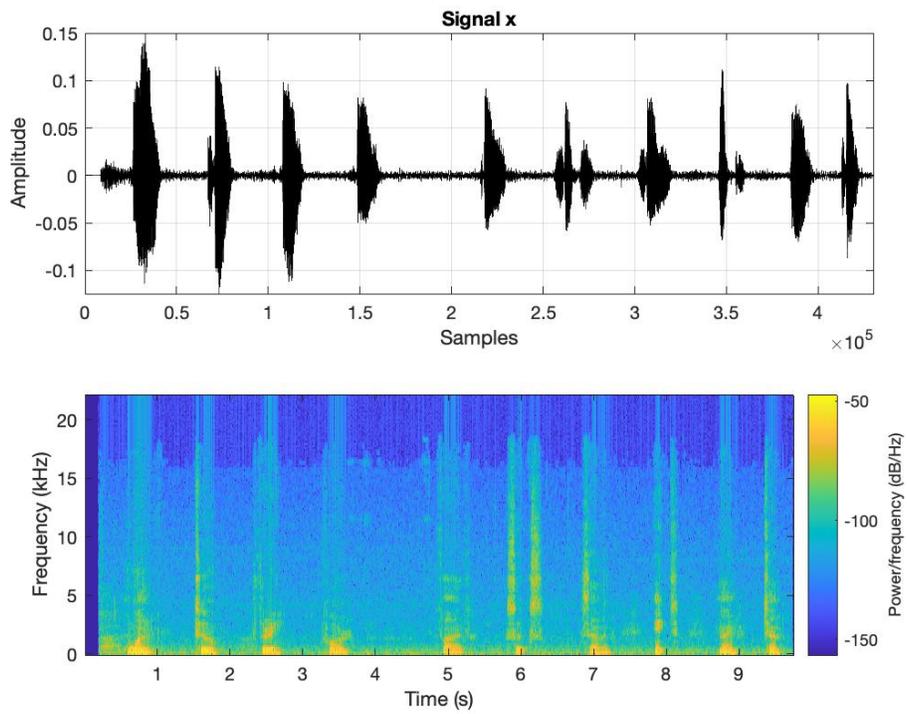


*Fig 22: iOS to Android Audio PSD*

The second to last test ran in MATLAB consisted of the FFT script test. Unlike the spectrogram test, the waveform displays the samples rather than the time. Like the spectrogram test however, the padding of zeros on the received android file is clear once again. This can be made clear with the script used in MATLAB contains a higher NFFT for improved accuracy and measurement of the file



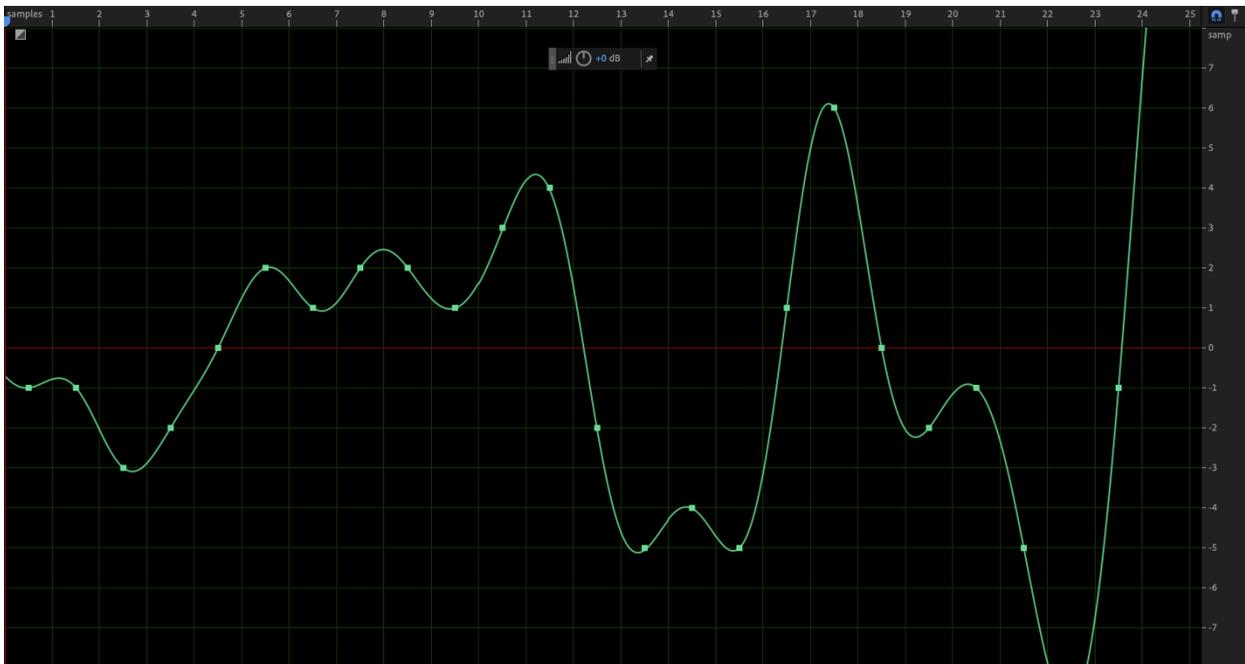
*Fig 23: iOS Created Audio FFT*



*Fig 24: Android Received Audio FFT*

As was stated earlier in this test, the number of consecutive zero level samples found in the created iOS snap was zero, but the number found in the received Android snap was 8156. Although this number was obtained within MATLAB, it was also checked manually in Adobe Audition and Hex Fiend. The true number of zero level samples in the received android file totaled 8158. One possible theory as to why there is padding is for the change in scripts when moving from one OS to another, it is worth remembering that Snapchat was created for initial use on iOS and a change occurs when crossing over onto a different OS.

The test between an iOS device and an Android device leads to the most uncertain of results among all the tests administered. The amount of padding between the two snaps is of concern because this was the only case when the created file had none meanwhile the receiving file had over 8000 consecutive zeros. This led to the receiving file being longer and larger in size than the original.



*Fig 25: iOS Created Audio Adobe Audition Zero Level Samples*

```

aud03.wav
0 | 52494646 C6D70C00 57415645 666D7420 10000000 01000100 44AC0000 88580100 02001000 RIFF.. WAVEfmt D. .X
36 4C495354 1A000000 494E464F 49534654 0E000000 4C617666 35382E37 362E3130 30006461 LIST INFOISFT Lavf58.76.100 da
72 74618007 0C00FFFF FFFFFDFF FFFF0000 02000100 02000200 01000300 0400FEFF FBFFCFDF ta. ....
108 FBFF0100 06000000 FEFFFFFF FBFF77FF FFFF0D00 0600EFFF F5FF0D00 0E00E9FF B3FF9FFF .. F Z Z ] W > " . 5
144 C7FF0C00 46005A80 5A005D00 57003E00 22001900 2E003500 1500FCFF 000077FF D0FFB9FF .. { . . . . . } $ E
180 C5FFC0FF 95FF78FF 83FF86FF 80FF86FF 9AFFA4FF A6FFC8FF 0A002200 00000800 24004500 \ h ^ T h . . l b T _ t n A
216 5C006800 60005400 68008C00 93007C00 62005400 5F007400 6E004100 0A00E0FF C8FFBEFF .....
252 C6FFD1FF F5FFE9FF D2FFC9FF C6FFB6FF 9EFFF9FF A2FFAFFF BFFFC4FF C7FFDEFF E9FFD7FF .....
288 CDFFD4FF D1FFCAFF D9FFEDFF DBFFC0FF D2FFF9FF 09000500 FBFFEEFF E0FFD1FF BDFFA4FF .....
324 9FFFAEFF BDFFD8FF FBFFCFDF FBFF0000 F5FFF2FF 11004400 67006800 74008200 84009C00 ..... D g k t . .
360 C400D600 C3008D00 66006200 62006E00 9400BD00 C900B000 99008F00 7A006700 5E004F00 . . . . f b b n . . . . z g ^ 0
396 3D002600 17001F00 23001C00 1D002700 31002C00 1D001C00 1300FCFF F8FF99FF F5FF0600 = & # ' ! ,
432 26003E00 4A00AD00 42002300 0700F8FF E3FFDFFF F3FFCFDF ECFFD9FF D1FFC7FF B3FFAEFF & > J M B #
468 C3FFE0FF E4FD22FF CAFFC9FF CDFD09FF E4FFEFFE EBFFD1FF C0FFC3FF CAFFC2FF AFFFACFF .....
504 B4FF88FF CBFFCCFF BEFF8E00 B9FFA8FF 9EFA22FF AEFFBAFF CAFFC8FF A0FF72FF 5DFF58FF ..... r ] [
540 5BFF55FF 4DFF58FF 62FF4E00 3E5FF51F 79FF96FF 93FF80FF 82FFA6FF CAFFBCFF A1FFB3FF [ . U . M . X . b . N . > Q . y . . . . .
612 A0FFA7FF AEFFC0FF C7FFCDFF EDF1700 29002E00 37002D00 14001200 24002C00 31004A00 ..... V . D . S . b . Z . H . > . 8 . I . r .
648 57004500 3E004E00 50003200 0900E5FF CEFFC6FF CBFFD8FF E7FFE1FF D0FFE7FF F6FF1200 ..... ) . 7 - $ , ! J
720 91008F00 84007900 78009200 8700D100 C3009900 71003F00 0800F0FF F2FF1400 44005F00 W E > N P 2
756 61004700 25003000 55005900 49005600 7A007F00 6D007600 85007D00 72006200 46004600 5 0
792 75009900 87007800 89009300 A000AD00 AB008800 43000800 F9FFFEFF 08000800 11002200 . . . . y { . . . . q ? . . . . D _
828 23004100 8900B100 AE00B600 C400B300 93008500 81007900 7D008100 69004D00 53006300 a G % 0 U Y I V z m v . } r b F F
864 5C004900 3C003600 38003B00 2A00C000 13003900 3C001800 F9FF55FF FAFFE6FF CAFFC3FF u . . { . . . . . C
900 C5FFC4FF C7FFE1FF 08001D00 1E002900 2E002E00 44006200 67004A00 1800F9FF FFFF8800 # A
936 1A004100 59006100 61004E00 48004400 0D00CDFF B5FFB8FF CAFFC6FF C0FFFFF5 25001100 \ I < 6 8 ; * 9 <
972 DAFFACFF 90FF74FF 6AFF78FF 8DFFA5FF BAFFB3FF A0FF89FF 76FF76FF 70FF72FF 8BFF99FF ..... ) . D b g J
1008 AEFFC9FF B8FFA0FF 90FF8AFF 9DFF9FFF 70FF49FF 47FF4AFF 48FF4FFF 46FF30FF 51FF68FF A Y a a N H D
1044 61FF44FF 31FF26FF 31FF61FF 82FF6CFF 47FF32FF 2DFF31FF 37FF42FF 5E9FF8FF A4FFA5FF ..... t . j { . . . . . v . v . p . r .
1080 A8FFA7FF AAFD44FF 1A005800 7B008400 77005E00 5D006D00 4F000800 C2FF92FF 80FF8FFF ..... p . I . G . J . K . O . F . = . Q . k .
1116 AAFF80FF B5FFC4FF C8FFCCFF CCFBAEFF 8BFF6AFF 4E5FF47F 4AFF59FF 74FF82FF 84FF77FF a . D . 1 . & . 1 . a . . l . G . 2 . - 1 . 7 . B . ^ .
1152 66FF7CFF ADFD08FF E9FFE3FF DBFFE2FF E6FFCDFF B2FFACFF A5FF9AFF 90FF75FF 4E5FF2AFF ..... X { w ^ } m 0
1188 1DFF2CFF 3DFF55FF 7AFF94FF A5FFB8FF C7FFB8FF AFFF81FF C1FFC9FF C6FFC3FF BAFFB2FF ..... j . N . G . J . Y . t . . . . w .
1224 CFFF88FF FDFFB8FF FEFFEAFF EAFF1400 36004300 3E002300 19002700 24000D00 F2FFCDFF f . | . . . . . u . N . *
..... U . z . . . . .
..... 6 C > # ' $

```

Fig 26: iOS Created Audio Hex Fiend Zero Level Samples

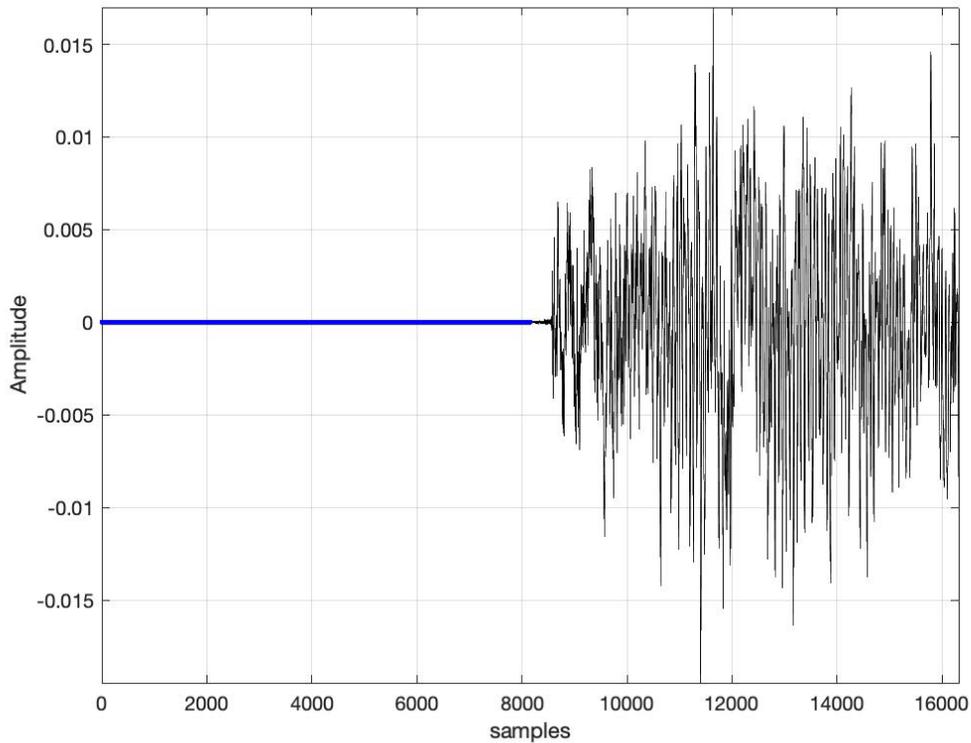


Fig 27: Android Received Audio MATLAB Zero Level Samples



Fig 28: Android Received Audio Adobe Audition Zero Level Samples

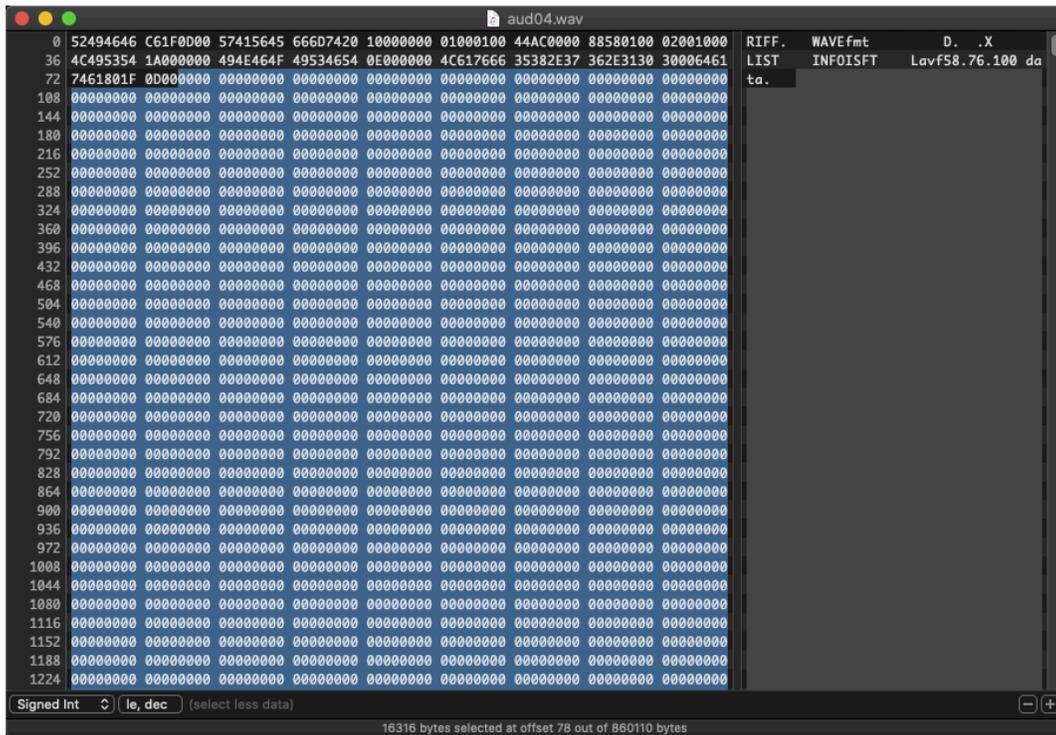


Fig 29: Android Received Audio Hex Fiend Zero Level Samples

### Test 3: Android to iOS

The third test, and subsequently the fourth test, administered yielded the most surprising results so far with the created and received audio files containing the most similarities. Again, this may be attributed to the scripting of Snapchat software when creating the Android version of the app. Beginning with the file analysis, the hash values were identical for both files for the first time. Sending a snap from an iOS device to an android or another iOS device yielded different hashes, but a snap created on an Android device has resulted in the receiving device to have an exact copy of the original snap. The SHA256 hash values for test three are as follows:

Created SHA256:

c94eb84b2d262669b8df5adb42749f689db30761da44d0156fd93b413e382c9a

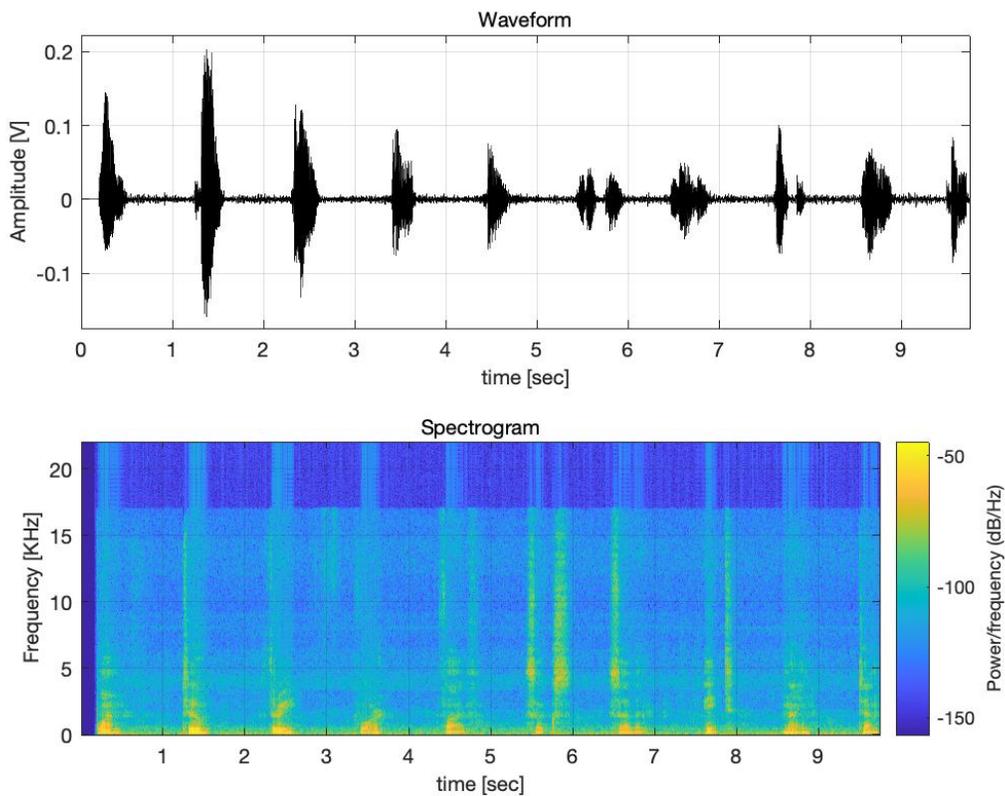
Received SHA256:

c94eb84b2d262669b8df5adb42749f689db30761da44d0156fd93b413e382c9a

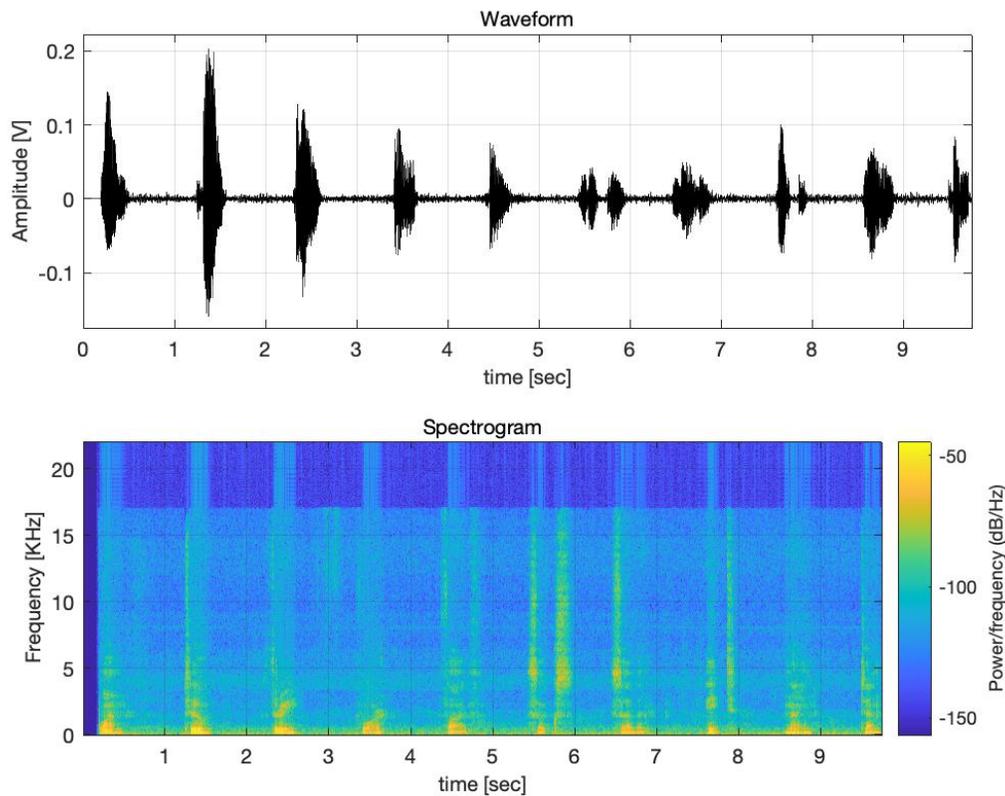
<b>Audio</b>		<b>Audio</b>	
<i>ID :</i>	256	<i>ID :</i>	256
<i>Format :</i>	AAC LC	<i>Format :</i>	AAC LC
<i>Format/Info :</i>	Advanced Audio Codec	<i>Format/Info :</i>	Advanced Audio Codec
<i>Codec ID :</i>	mp4a-40-2	<i>Codec ID :</i>	mp4a-40-2-2
<i>Duration :</i>	9 s 752 ms	<i>Duration :</i>	9 s 752 ms
<i>Bit rate mode :</i>	Constant	<i>Bit rate mode :</i>	Constant
<i>Bit rate :</i>	132 kb/s	<i>Bit rate :</i>	132 kb/s
<i>Channel(s) :</i>	1 channel	<i>Channel(s) :</i>	1 channel
<i>Channel layout :</i>	C	<i>Channel layout :</i>	C
<i>Sampling rate :</i>	44.1 kHz	<i>Sampling rate :</i>	44.1 kHz
<i>Frame rate :</i>	43.066 FPS (1024 SPF)	<i>Frame rate :</i>	43.066 FPS (1024 SPF)
<i>Compression mode :</i>	Lossy	<i>Compression mode :</i>	Lossy
<i>Stream size :</i>	156 KiB (25%)	<i>Stream size :</i>	156 KiB (52%)
<i>Title :</i>	Snap Audio	<i>Title :</i>	Snap Audio
<i>Language :</i>	English	<i>Language :</i>	English
<i>Encoded date :</i>	UTC 2021-09-29 22:45:47	<i>Encoded date :</i>	UTC 2021-09-29 22:45:50
<i>Tagged date :</i>	UTC 2021-09-29 22:45:47	<i>Tagged date :</i>	UTC 2021-09-29 22:45:50

Fig 30: Android Created to iOS Received Audio Metadata

There are a few interesting points to notice when analyzing the audio metadata for both files. For starters, the ID name is the same for the first time as well as the Codec ID name for the receiving file is the same but with an extra character for naming purposes. The duration remains the same once again, but the only difference now has to do with the stream size percentage. The difference in percentage from the created audio and received audio is 27%. One final interesting finding is that Snapchat embeds a signature, here called Title, onto the audio file letting the analyst know it came from Snapchat. This only occurs when the creation device is an Android regardless of what OS is receiving the snap. In test 2, neither audio file had the title “Snap Audio” but rather “Core Media Audio.” With the examination of hash values and metadata yielding similar results, it was expected to continue that trend in MATLAB.

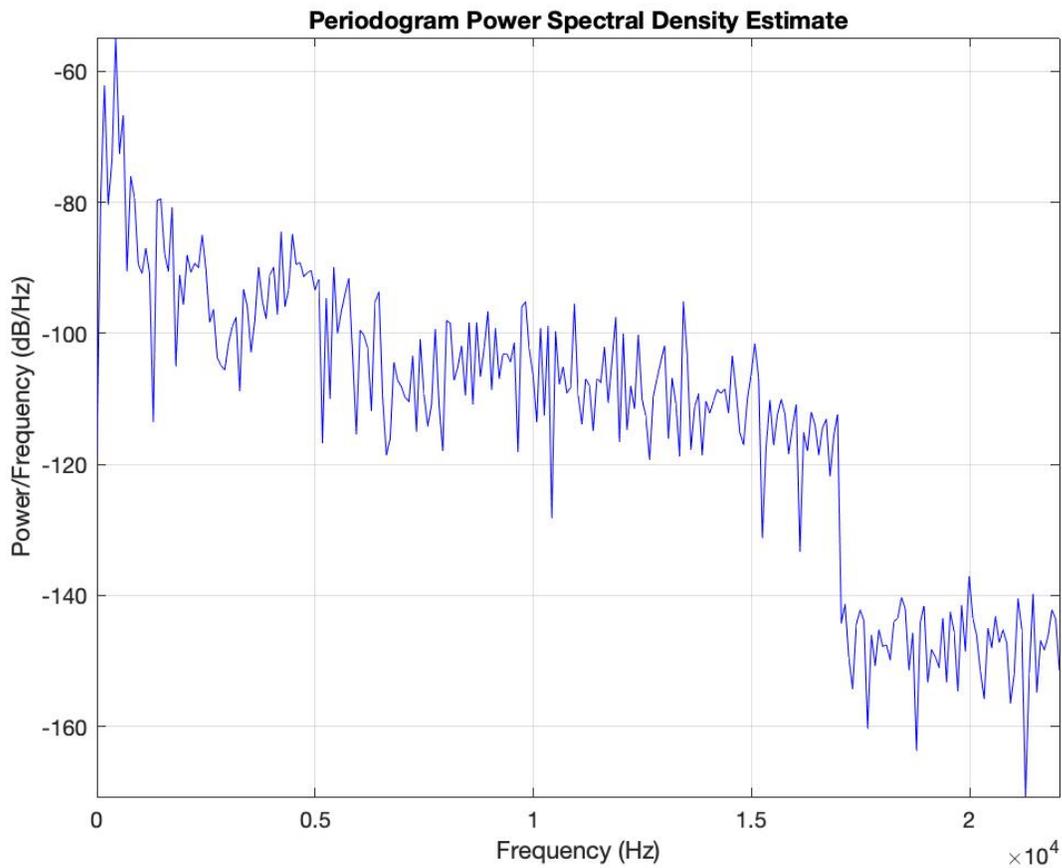


*Fig 31: Android Created Audio Spectrogram*



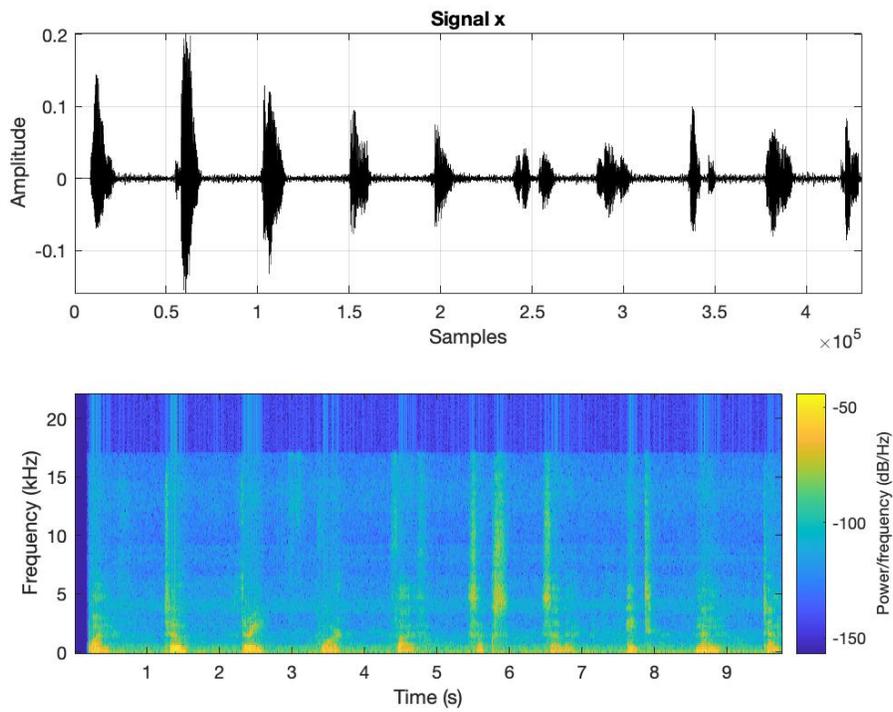
*Fig 32: iOS Received Audio Spectrogram*

Where the results become both intriguing and concrete is during the second MATLAB test with the frequency analysis spectrum test. This test once again outputs a PSD graph, the file CC and MQD. The PSD graph produced from these two audio files exhibits a singular frequency line because the two files are the same. The highest number obtainable from the CC test is 1, and that result was achieved by these audio files. The result from the MQD test was negative infinite, represented as  $-\text{Inf}$  in MATLAB. Achieving a negative infinite result from the MQD log is by no means an error with the script that was used or MATLAB's calculation. The reason why the MQD result is negative infinite is because the equation for this formula contains a zero in the denominator. The assessment that both files have a high correlation with each other can be made certain with those results.

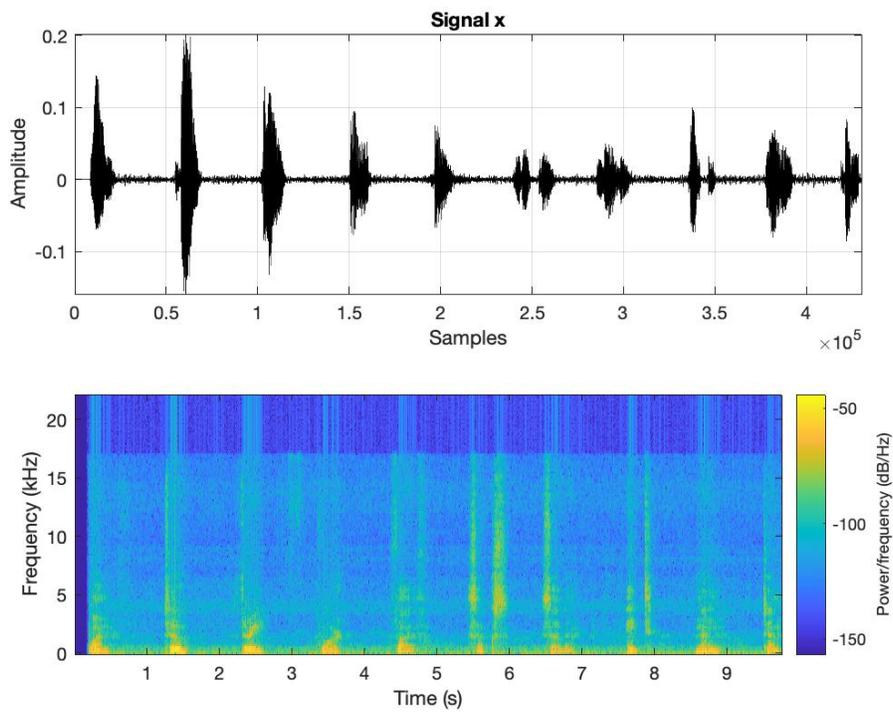


*Fig 33: Android to iOS Audio PSD*

The penultimate MATLAB test consisted of the FFT test with a high FFT order for frequency measurements. With a higher order, the accuracy of analysis between the two audio files can be examined. The following two images appear to be the same, but they are the results of the two separate acquired audio recordings.

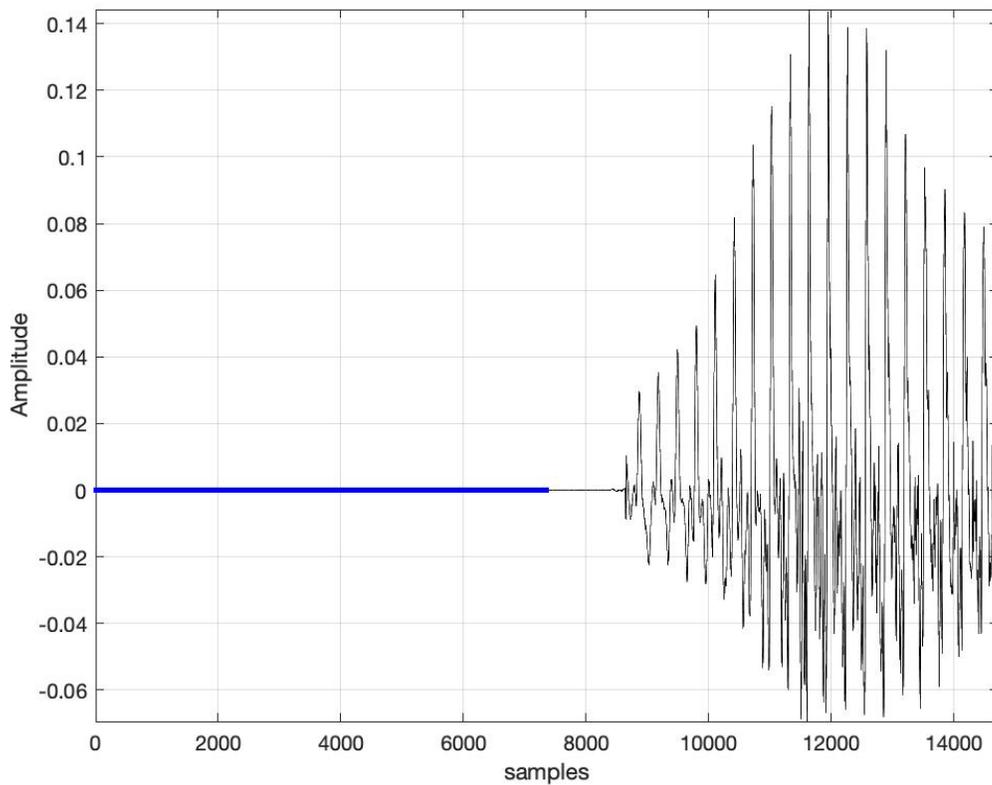


*Fig 34: Android Created Audio FFT*



*Fig 35: iOS Received Audio FFT*

The results from the spectrograms and spectrographs indicate an area of empty space in the beginning of the file. This is thought to be believed as the padding of zeros at the beginning of the file. With the matching of hashes, metadata, and seemingly similar graphs, this must mean that both audio files have the same number of consecutive zero level samples. The number of zeros reported by MATLAB is 7359. This was further checked with Audition and Hex Fiend with the number, yet again like the previous tests increasing by two, becoming 7361. This number is true in the case with both the created and received audio files.



*Fig 36: Android Created Audio MATLAB Zero Level Samples*

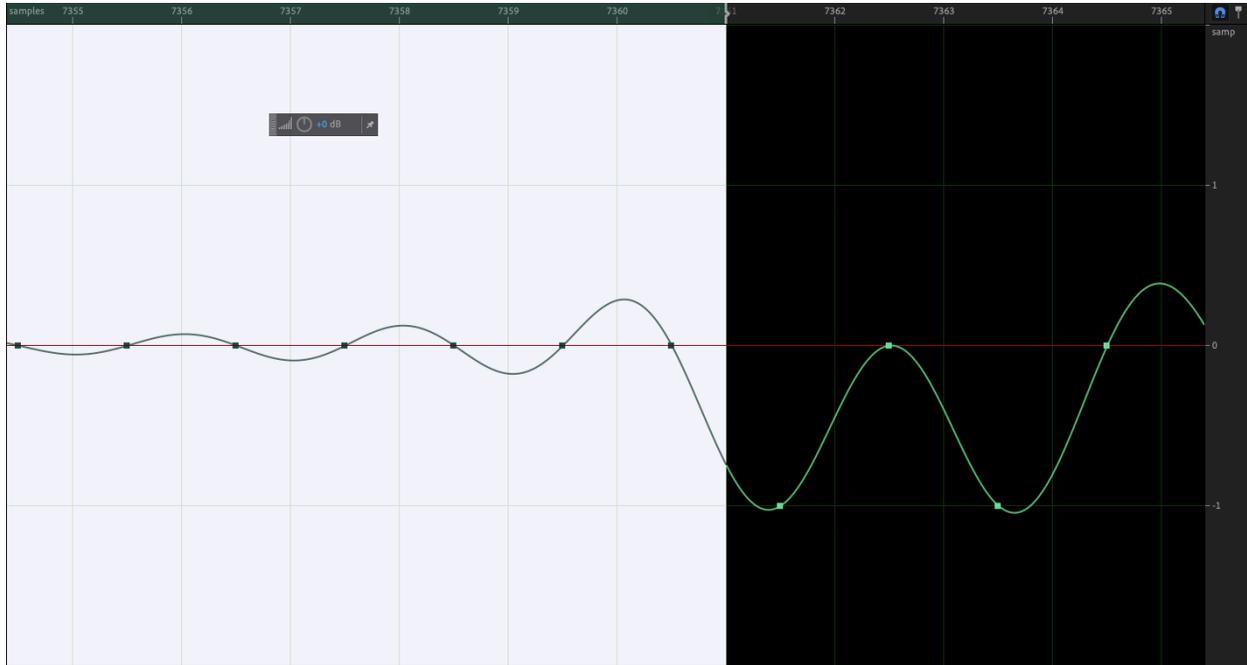


Fig 37: Android Created Audio Adobe Audition Zero Level Samples

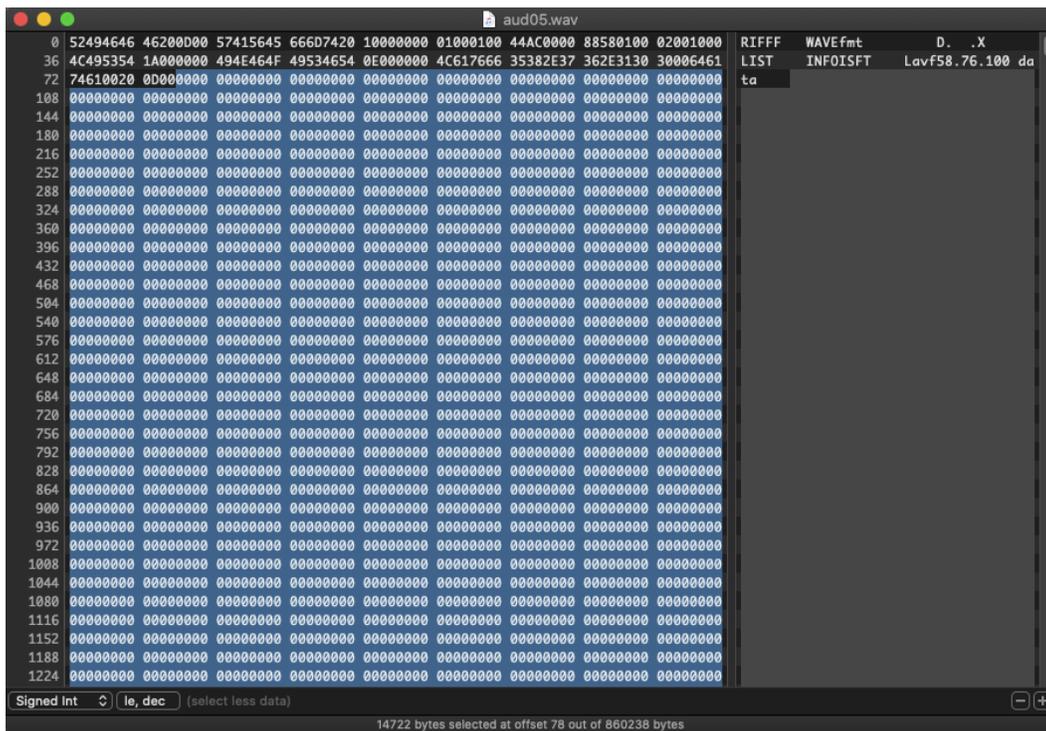
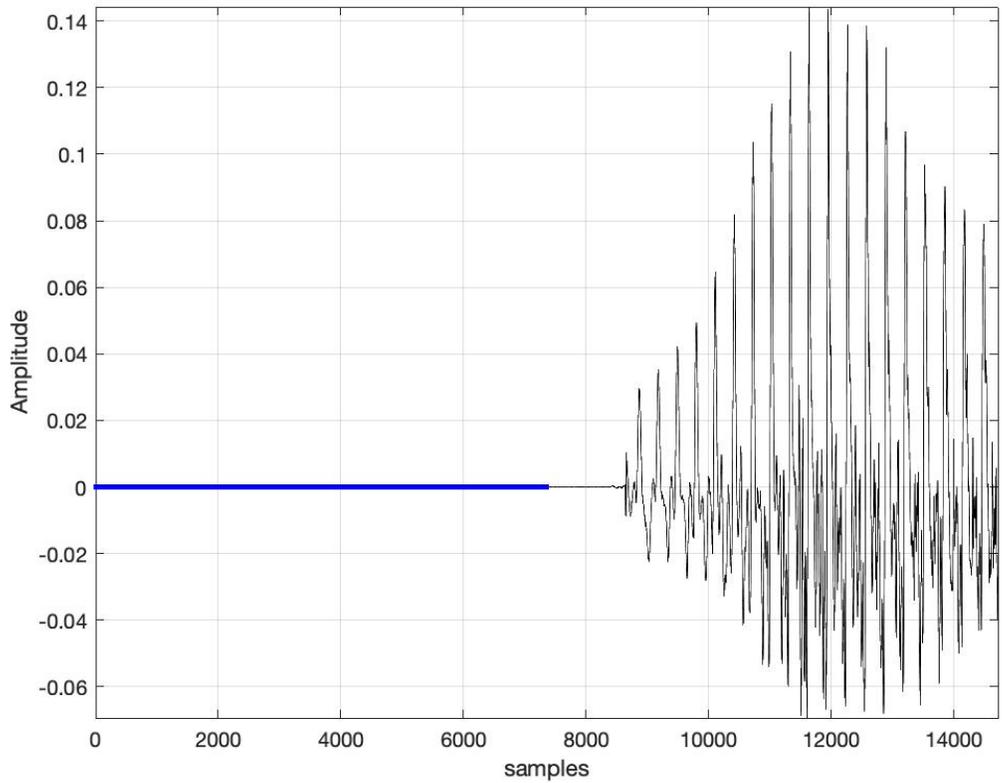


Fig 38: Android Created Audio Hex Fiend Zero Level Samples



*Fig 39: iOS Received Audio MATLAB Zero Level Samples*



*Fig 40: iOS Received Audio Adobe Audition Zero Level Samples*

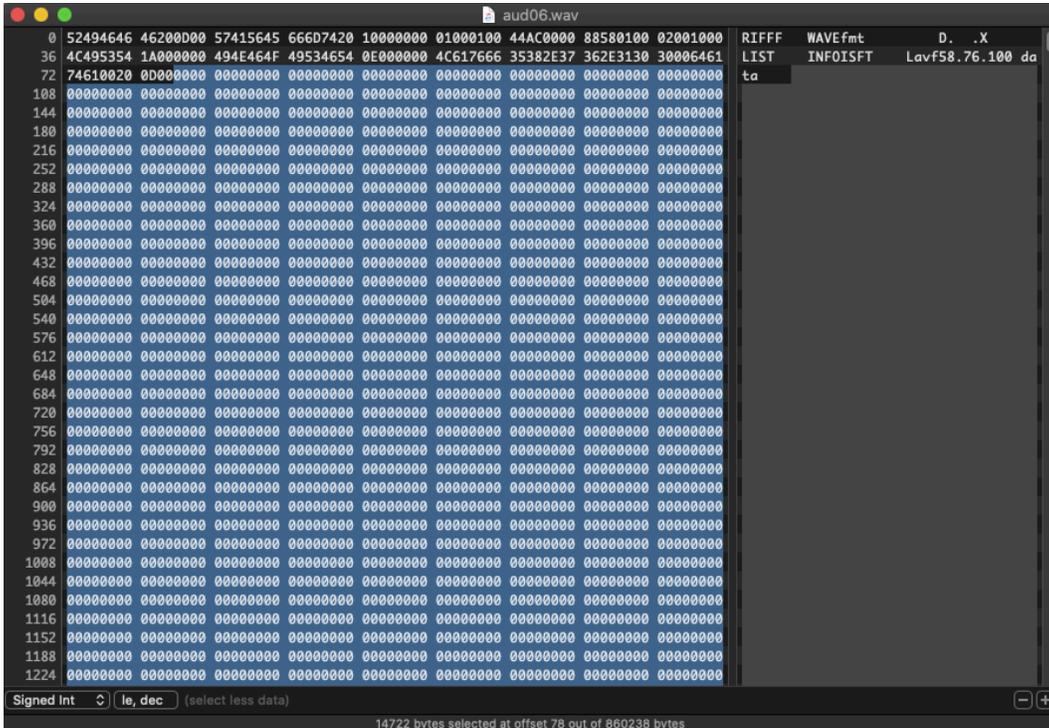


Fig 41: iOS Received Audio Hex Fiend Zero Level Samples

#### Test 4: Android to Android

The final test conducted was sending snaps between two different android devices. The interesting aspects of these results was that, like the previous test, the data was the same. Both the created and received audio files were exact copies of each other. Matching hash sets, similar metadata, and MATLAB tests. The SHA265 is as follows

Created SHA256:

91c92809c4cc38312448cc9883fc2589593dc880398993efa2916562dc1b052f

Received SHA256:

91c92809c4cc38312448cc9883fc2589593dc880398993efa2916562dc1b052f

Much like in the previous exam, the metadata on both files remained consistent, this includes the file ID, Codec ID, and file Duration. The one aspect that does continue to change is

the stream size percentage that decreases from 21% on the created device to 15% on the received device.

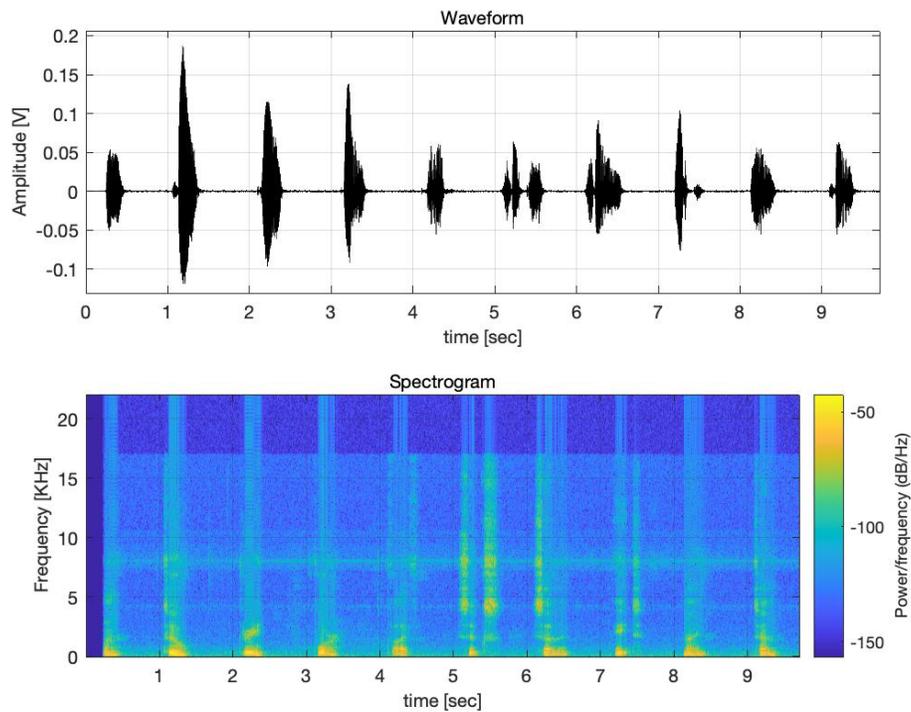
<b>Audio</b>		<b>Audio</b>	
<i>ID :</i>	256	<i>ID :</i>	256
<i>Format :</i>	AAC LC	<i>Format :</i>	AAC LC
<i>Format/Info :</i>	Advanced Audio Codec	<i>Format/Info :</i>	Advanced Audio Codec
<i>Codec ID :</i>	mp4a-40-2	<i>Codec ID :</i>	mp4a-40-2
<i>Duration :</i>	9 s 705 ms	<i>Duration :</i>	9 s 705 ms
<i>Bit rate mode :</i>	Constant	<i>Bit rate mode :</i>	Constant
<i>Bit rate :</i>	132 kb/s	<i>Bit rate :</i>	132 kb/s
<i>Channel(s) :</i>	1 channel	<i>Channel(s) :</i>	1 channel
<i>Channel layout :</i>	C	<i>Channel layout :</i>	C
<i>Sampling rate :</i>	44.1 kHz	<i>Sampling rate :</i>	44.1 kHz
<i>Frame rate :</i>	43.066 FPS (1024 SPF)	<i>Frame rate :</i>	43.066 FPS (1024 SPF)
<i>Compression mode :</i>	Lossy	<i>Compression mode :</i>	Lossy
<i>Stream size :</i>	155 KiB (21%)	<i>Stream size :</i>	155 KiB (15%)
<i>Title :</i>	Snap Audio	<i>Title :</i>	Snap Audio
<i>Language :</i>	English	<i>Language :</i>	English
<i>Encoded date :</i>	UTC 2021-09-29 22:56:43	<i>Encoded date :</i>	UTC 2021-09-29 22:56:52
<i>Tagged date :</i>	UTC 2021-09-29 22:56:43	<i>Tagged date :</i>	UTC 2021-09-29 22:56:52

*Fig 42: Android Created to Android Received Audio Metadata*

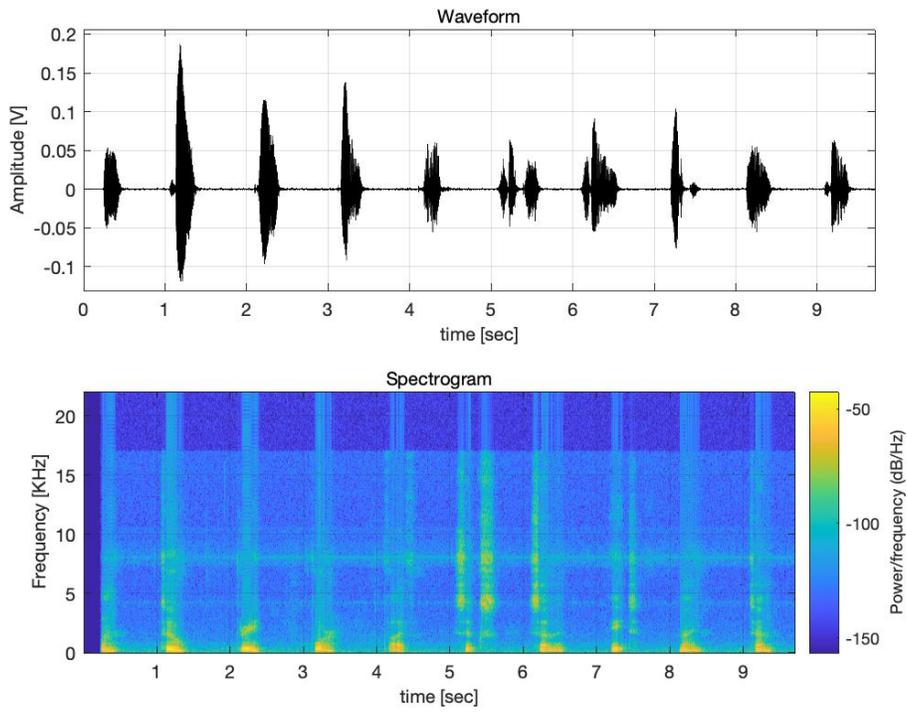
One minor detail that separates this test from the Android to iOS test is that the Codec ID is the same. In the previous test, the receiving audio on the iOS device had the name mp4a-40-2-2. This could very much signify that the received audio on that test is a copy of the original, meanwhile here the received audio is that exact file sent. A final note on the metadata is that with an Android device sending the snap, the title of the audio includes Snap Audio yet again letting the analyzer know where the file was generated from.

As was the case in the previous tests, the same MATLAB tests were used on the android-to-android scenario. The results alone from the hash and metadata set the tone for what was to be expected, much like in the previous tests. All the frequency analysis exams from the spectrogram, spectrum, FFT, and zero level samples provided results that indicate the created

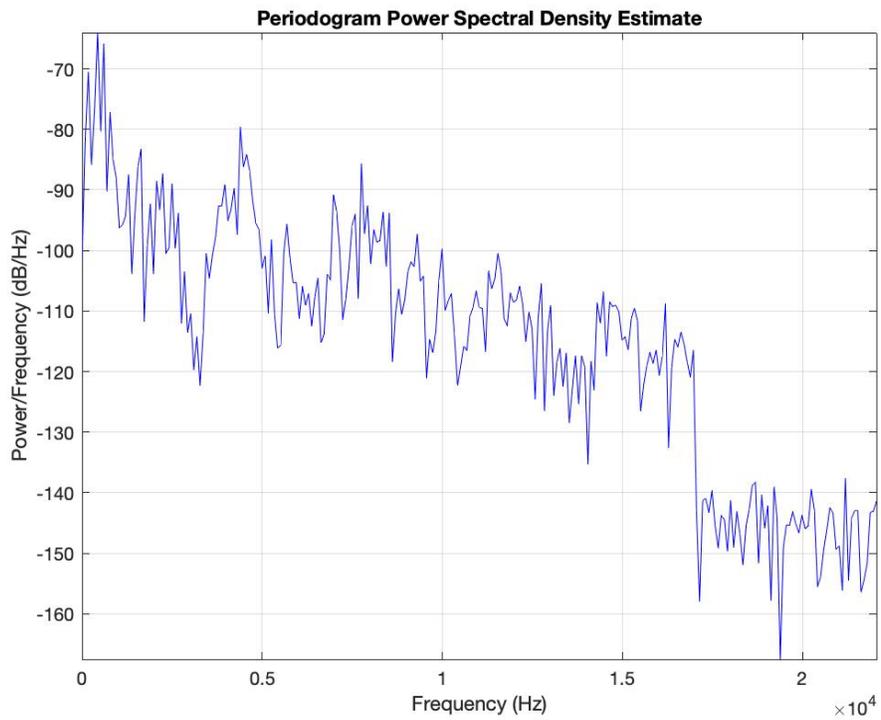
and received files are the same. The spectrogram tests display the waveform as well as the spectrogram with seemingly similar images. The spectrum PSD graph displays both audio frequency levels even though it seems as though one frequency level is being displayed. The reasoning behind this is that the two files are identical to each other. This is also made apparent by the CC value of 1, the highest possible correlation value. The MDQ is negative infinite like it was in the android to iOS test. Again, the reasoning behind the calculation of negative infinite is because the denominator in the quadratic equation is 0. The FFT of both audio files were again examined with a high FFT order for frequency analysis. To the naked eye the difference between the graphs holds no difference.



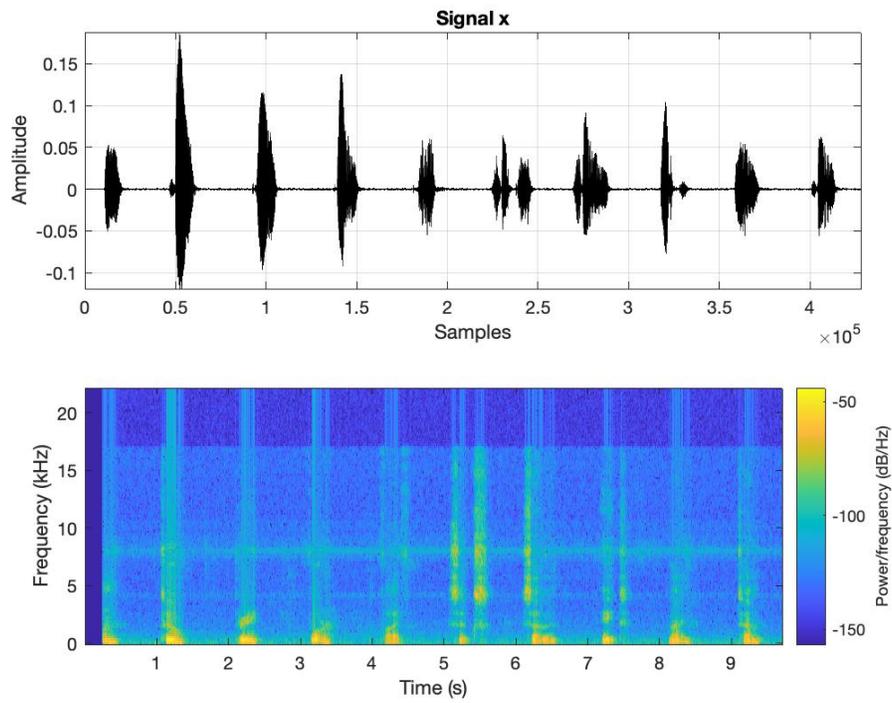
*Fig 43: Android Created Audio Spectrogram*



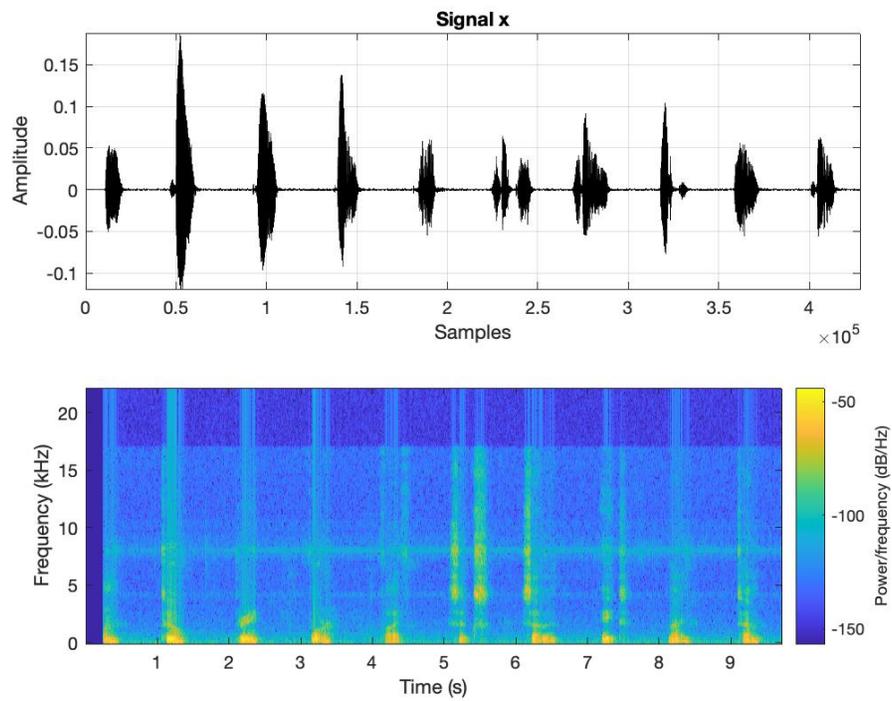
*Fig 44: Android Received Audio Spectrogram*



*Fig 45: Android to Android Audio PSD*

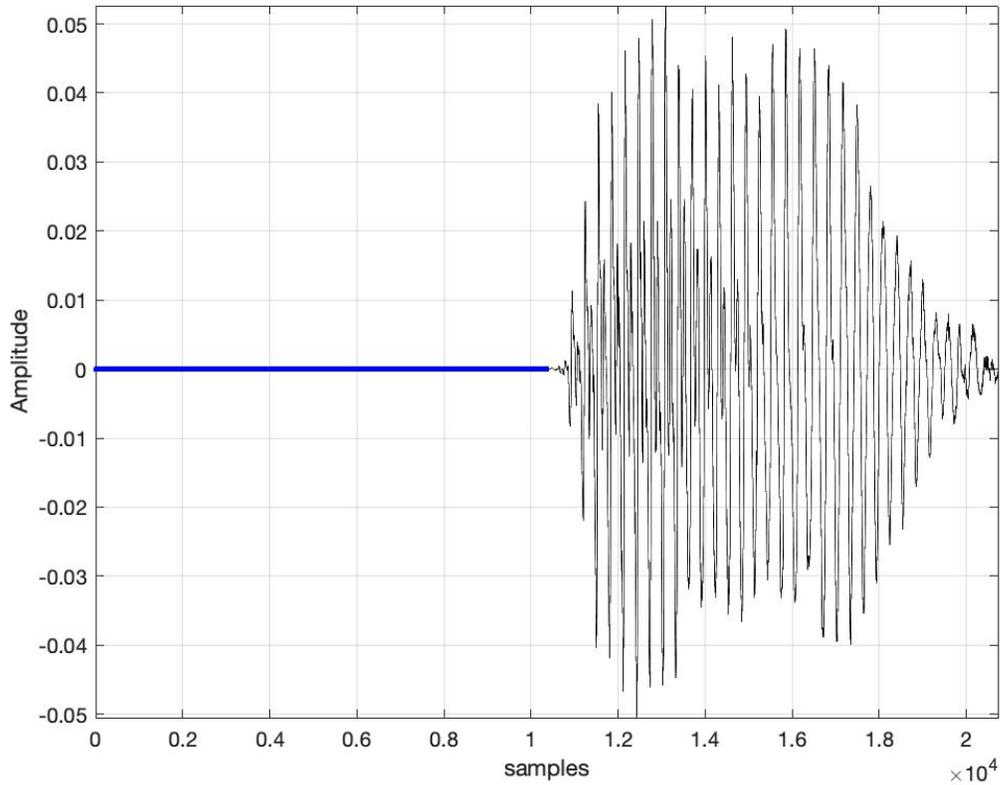


*Fig 46: Android Created Audio FFT*



*Fig 47: Android Received Audio FFT*

The results from the consecutive zero level samples from the android-to-android testing proved to be the highest number of padding occurring among the four tests performed. MATLAB reported a number of 10360 consecutive zeros on both files with that number increasing to 10362 when manually searching for them in Audition and Hex Fiend.



*Fig 48: Android Created Audio MATLAB Zero Level Samples*

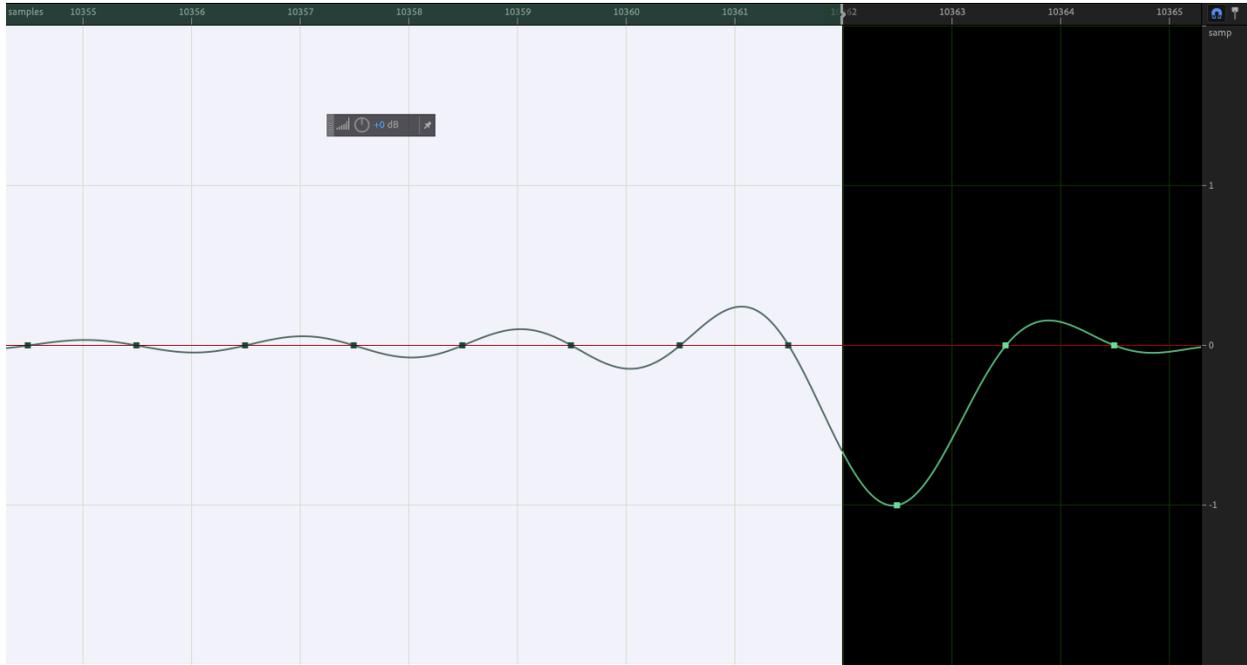


Fig 49: Android Created Audio Adobe Audition Zero Level Samples

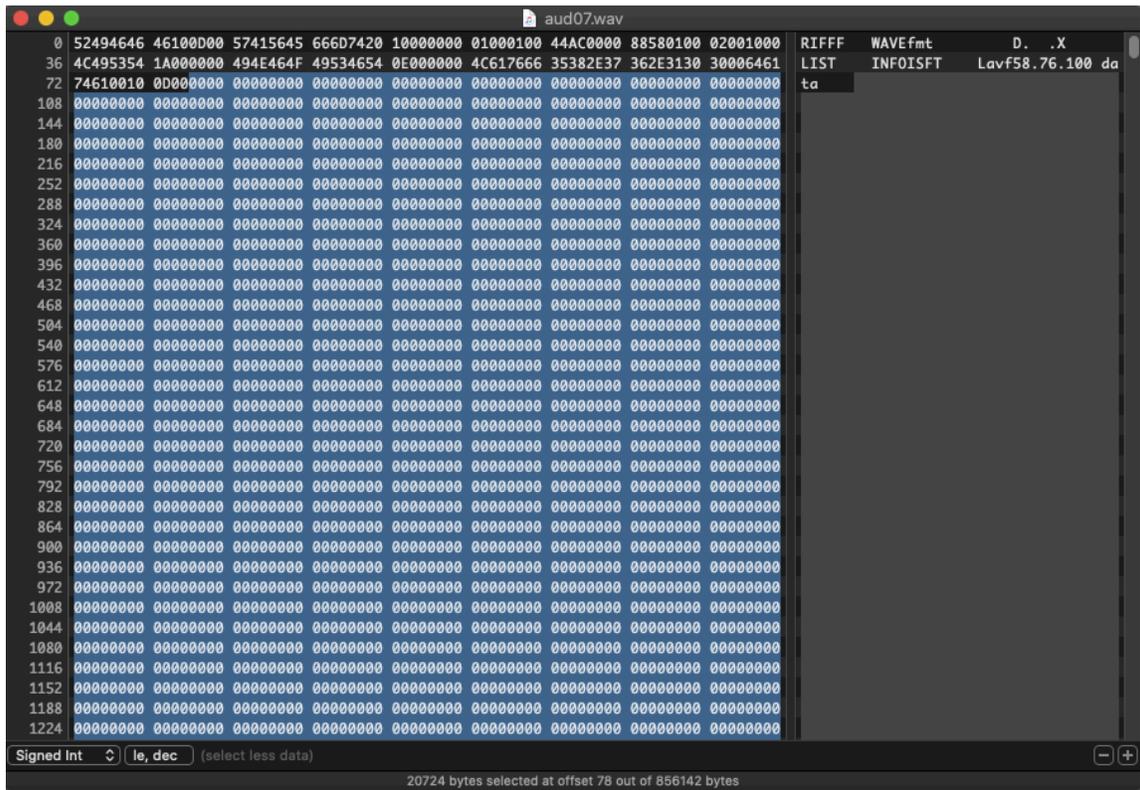
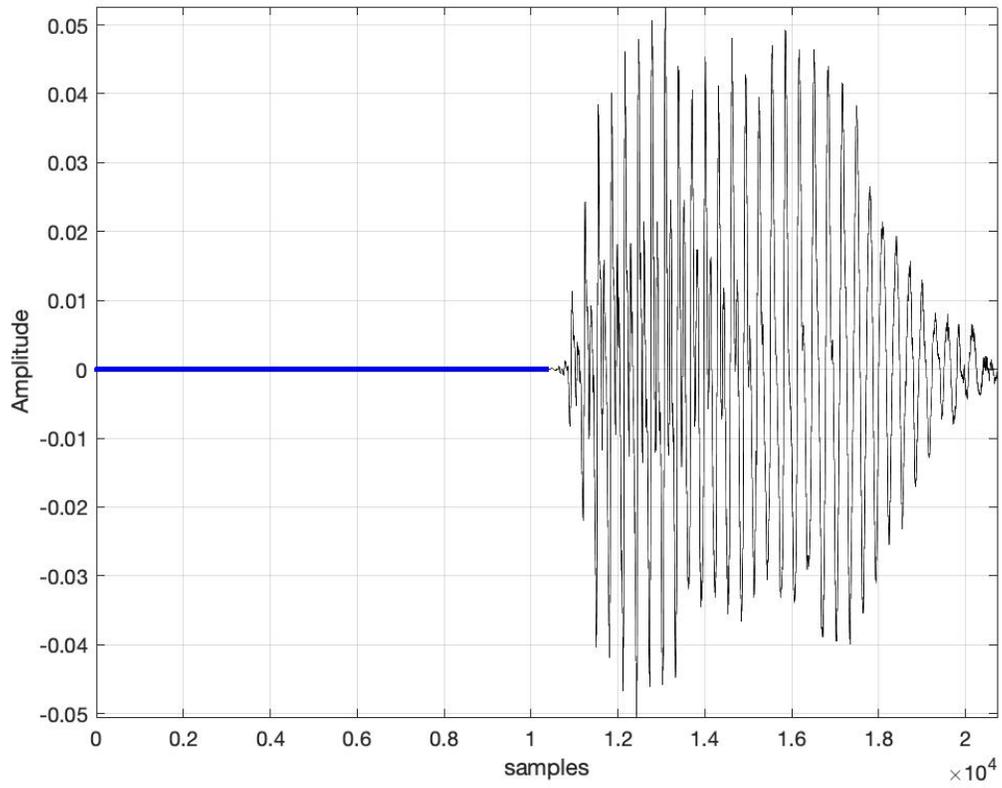
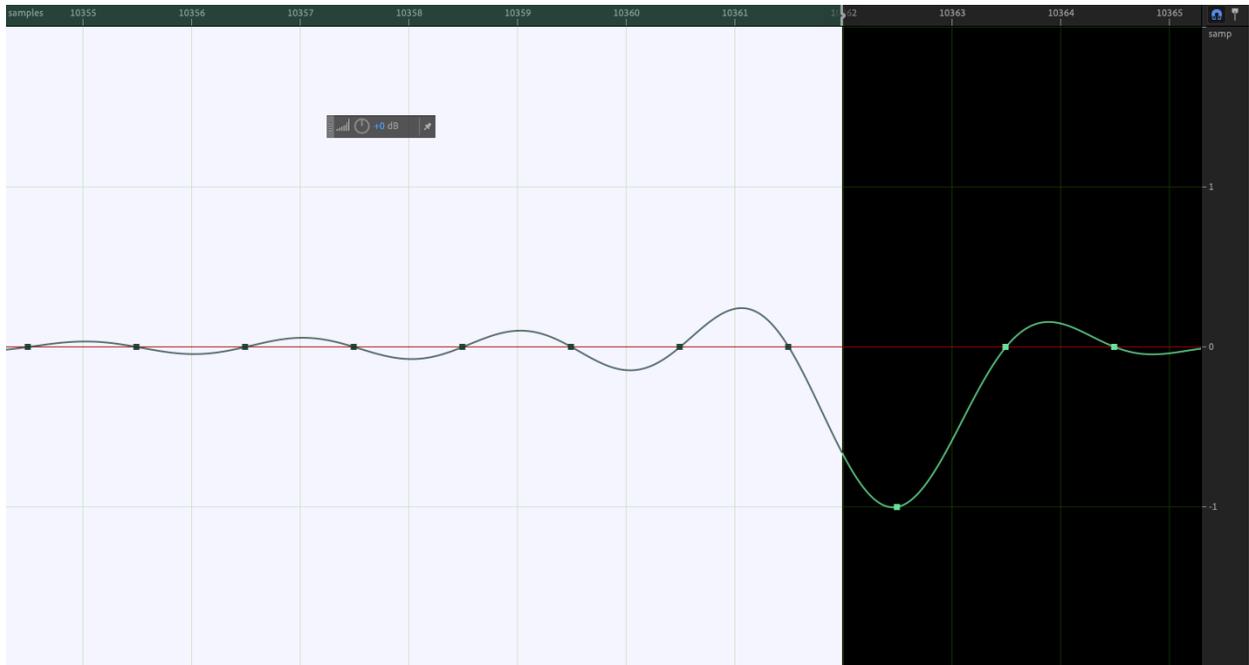


Fig 50: Android Created Audio Hex Fiend Zero Level Samples



*Fig 51: Android Received Audio MATLAB Zero Level Samples*



*Fig 52: Android Received Audio Adobe Audition Zero Level Samples*

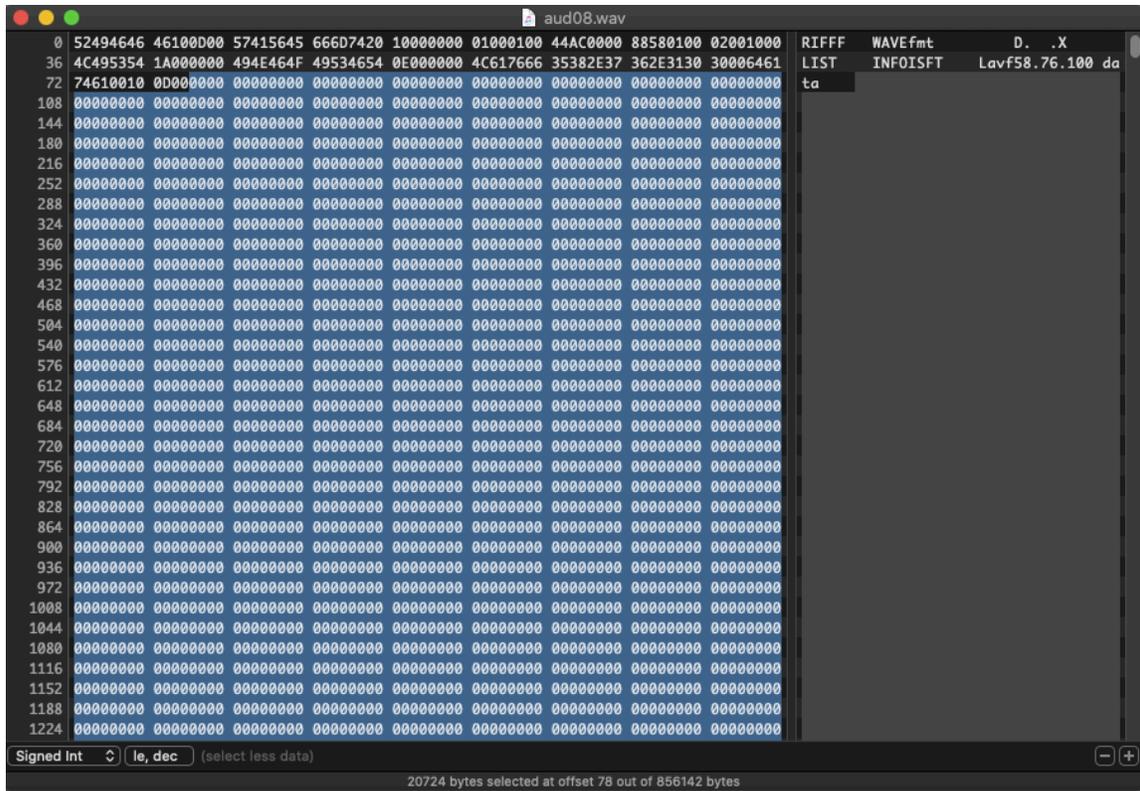


Fig 53: Android Received Audio Hex Fiend Zero Level Samples

## **CHAPTER V**

### **CONCLUSIONS**

The results from the tests can be separated into two separate parts, the first being an iOS device as the creator and the second using an android device as the creator. Using an android device to send a snap to another device, regardless of what OS that device is running, produces an exact copy of the original. Both files are consistent with one another, and they even have the same hash value. Using an iOS device on the other hand leads to varying results whether it be padding the audio with zero level samples or slightly changing the duration of the file. Sending a snap from an iOS device will result in a new file being created rather than duplicating the original snap. These findings are significant because the changes seem to occur when uploading and downloading from the Snapchat server, but only for iOS devices.

#### **Future Research**

As with any piece of software out on the market today, Snapchat is always undergoing maintenance and performance upgrades. Snapchat was on version 11.44.0.37 when the testing began but has since updated to version 11.55.1.37. A possible area of research is to study Snapchat's algorithm changes as new versions are released and compare them to previous versions. This would give insight into where the transcoding occurs, whether it happens as the snap is being uploaded onto the Snapchat servers or after it was downloaded from the server. A record can be kept as to what changes were done from the previous version to the next. This would be important because many people hold off on updates and this leads to some users running the newer software and others running the older software.

## REFERENCES

1. Aji, Mukhlis Prasetyo & Riadi, Imam. The Digital Forensic Analysis of Snapchat Application using XML records. *Journal of Theoretical and Applied Information Technology*. October 2017
2. Alyahya, T., & Kausar, F. (2017). Snapchat Analysis to Discover Digital Forensic Artifacts on Android Smartphone. *Elsevier, Science Direct Procedia Computer Science*.109C 1035-1040. Doi: 10.1016/j.procs.2017.05.421.
3. Malley, Angela Rae. A Comparison Analysis of Saved Snapchat Video Files on Android VS iPhones. *University of Colorado at Denver*. 2021.
4. SWGDE Technical Notes on FFMPEG Version: 2.0 (November 20, 2018)