Marist College

Audio Encryption with AES Milestone

Historical Context and AES Implementation in Java for Modern Applications

Tom Mackinson
MSCS 630 Cryptography
Aaron Kippins
May 9th, 2021

Abstract

The goal of this paper is first to provide a historical background on audio encryption, to provide some more modern approaches used by current applications. Audio encryption and piracy date back to the early days of the internet, and with the rise of CDs and music sharing applications the need for encryption became more and more necessary. As time went on, measures taken to protect intellectual property grew more and more stringent, and certain companies even took illegal actions in order to stop users from copying CDs. These measures would eventually lead to music streaming as an even more convenient alternative to music piracy, however streaming would lead to the need for encryption, much like how CDs used varying forms of encryption to protect their contents. Finally, I will show how I implemented AES encryption using java as one method of protecting audio content, starting from lab work done in class, and finally reading an mp3 file into java and encrypting the individual bytes.

Introduction

In the early 2000s, P2P music sharing exploded onto the internet thanks to sites such as Napster and Limewire. Music sharing over the internet had a serious impact on the sales of CDs at the time, with studies showing sales of CDs were down anywhere from 14 to 30 percent (Michel: The impact of Digital File Sharing…). With this loss of sales due to illegal file sharing online, the music industry took measures to protect their copyrights and prevent more music from being shared illegally. From taking legal action against Napster in attempts to shutdown the website (Kravets: RIAA sues Napster), to installing security measures on user's computers, even without asking for consent or providing a license agreement (Brown: Sony BMG Rootkit scandal), music companies have gone to great lengths to try and protect their intellectual property.

Background/Related Work

Throughout this course I have worked on various types of encryption in the labs. For the final two lab assignments, I implemented AES encryption, both generating keys and the actual steps of encryption with those keys. For this project I decided to use that process and encrypt audio with the same algorithm. So far I have an application that can convert a file into an array of bytes. The only problem from this, is that the algorithm we wrote works on hex strings, with pairs of hex values representing 16 bytes in total.

Methodology

To actually go about encrypting an mp3 file, I started by using the code we created in labs. Our last two labs were to write the AES encryption algorithm and have it encrypt hex pairs using hex keys. This worked fine for the labs, but leads to a few problems when working with audio files. My initial program was designed simply to copy a file. This means read the file in as input, and then output a new file with a different name. I started with this as a first goal since these are the same basic steps my final program would need to perform. The only difference is that I would have to add encryption and decryption in between reading and writing, however I figured that if I could get the actual reading and writing working, the rest would be as simple as connecting my encryption algorithm and writing a decryption algorithm. I also wanted this program to run in the command line. Since the initial idea for this project was to mimic the encryption performed in the back end by streaming apps, this task is supposed to be automated and easily scripted. As such creating a GUI or front end for it seemed to be counter productive, it would take time away from actual research and would go against what this

project was trying to simulate.

The first major issue was actually loading the file into java. That was something I've never worked with before, generally any input for a program I've written has simply been a string input. At first I tried to go about reading the file in the way I would normally, with a scanner. Scanner works fine for basic strings or short input streams, but loading an entire non text document this way did not work as planned. My test mp3 file was a roughly 3 minute long song with a file size of about 6MB. Copying this file took a few minutes with a scanner, and lead to a text file that had a slightly smaller file size than the original mp3. This file was unreadable, and unplayable by my music player application. I did some more research and quickly saw that scanner was not suitable for this purpose. I left scanner in so the user could enter both a file name and an encryption key, which again, goes along with the idea that this application should be easy to write scripts for, however for actually reading files I had to do some more research. Looking through java documentation and other resources online, I eventually came across input and output streams. These provided me with a way to read a file in and store it as an array of bytes. Converting my program from scanner to input streams was not a difficult task, and supplying the scanner with the inputted file name meant that it could load any file I wanted. Using input and output streams, my initial mp3 file was copied to a new file which played as expected in a music player.

With this first issue out of the way, I then started to look into the actual encryption and decryption process for audio files. Since I was working with a byte array and expected hexadecimal pairs, I decided to work on converting the bytes into hex. If I could find a way to quickly convert between the two types, then the encryption algorithm would have the expected input, and the decryption algorithm could be based around the same code without making major modifications. This conversion from bytes to strings however was too time consuming. Since this is supposed to mimic audio streaming apps which have very quick times to load a file, I needed to make some changes. Converting my initial 6MB mp3 file from bytes to hexadecimal took over 2 hours in my first attempt. This would not work for a finished project, so I will have to make some further changes to the code. This is where I stopped before working on this milestone, however I will write down some of the ideas of how I will speed up this process. My first idea is to simply use a smaller file. While this means I won't be working with a full song like streaming applications do, I will be able to rapidly test and compare my code, and eventually build it up to a full audio file. This might alleviate some of the problems with this process, however it is not a permanent solution.

To speed up the application even further, I will simply remove the conversion from bytes to hex. This will require me to do one of two things. I will either have to break down the byte array into a series of length 16 byte arrays, and then convert it into hex just before sending it into the encryption algorithm, or I will have to convert the algorithm to expect bytes instead of hex strings. The first option, converting to strings just before encryption would be easiest to implement out of these two. I would simply take the plaintext, create a length 16 hex array, then pass it to the encryption application. I can even store the encrypted result as hex since the decryption algorithm won't need to convert back to bytes until after the ciphertext has been decrypted. The other option, while more difficult to implement, I believe would be much faster in total. This removes the problem of converting the bytes into strings, and I wouldn't even need to create byte arrays. I simply would work through a for loop, and pass 16 bytes at a time into the encryption algorithm. The problem with this method is that I have to make major changes to the encryption algorithm. The lookup tables either have to be changed to store bytes or need to convert the hex that they return into unsigned bytes at the return statement. Any other string conversions must be changed to unsigned bytes. With those changes to the code, the decryption algorithm would still be very similar to the current encryption algorithm, just with certain functions reversed as necessary to undo the encryption steps.

From where I am in the project, the only code I have left to work on is modifying the encryption algorithm and then storing the encrypted file. I then have to write the decryption algorithm, and the

decryption driver. The decryption algorithm will have to work backwards through the steps of the encryption algorithm, however certain steps will also have to be inverted. Finally, the decryption driver will have to be written. This program will start the same as the encryption driver, reading a file name and a key in, then loading the file and sending it several bytes at a time to the decryption algorithm. The result returned by decryption will also need to be stored, and then written to a new file. This decrypted file will simply be compared with the original both by listening, and with a verification program. The verification program will simply read the two files in, and compare the bytes found within. This process will simply verify that the two files match exactly, however if any changes were to occur then it would be unlikely that the decrypted file would even be able to be read by an audio application, as it would most likely not be a valid mp3 file anymore.

Discussion

This is simply the work that I have completed thus far for this project. I still need to do some more research on audio encryption methods, as well as CD encryption methods as that provides some interesting background that shows why this whole application is necessary. The serious impact of music piracy on the industry in the early 2000s lead to an extreme response form the music industry. This response would eventually lead to music streaming as we know it today, however streaming itself still requires security. That need for security in streaming apps is why I am working on implementing AES encryption for audio files, with the understanding that my application should be as quick as possible with AES encryption, while also being easy to script and automate. I still have a bit more work to do, some more research and better sources for the historical background sections, as well as the actual programming itself, however this has been an interesting look into the topic of audio and music encryption.

Works Cited

Brown, Bob. "Sony BMG Rootkit Scandal: 10 Years Later." *CSO Online*, 28 Oct. 2015, www.csoonline.com/article/2998952/sony-bmg-rootkit-scandal-10-years-later.html.

Dowling, Stephen. "Napster Turns 20: How It Changed the Music Industry." *BBC Culture*, BBC, 31 May 2019, www.bbc.com/culture/article/20190531-napster-turns-20-how-it-changed-the-music-industry#:%7E:text=On%201%20June%201999%2C%20a,the%20boardrooms%20of%20record%20companies.

Kravets, David. "Dec. 7, 1999: RIAA Sues Napster." *Wired*, 7 Dec. 2009, www.wired.com/2009/12/1207riaa-sues-napster/#:%7E:text=With%20a%20bankrupt%20Napster%20unable,with%20loans%20totaling%20%2485%20million.

Michel, Norbert J. "The Impact of Digital File Sharing on the Music Industry: An Empirical Analysis." *Topics in Economic Analysis & Policy*, vol. 6, no. 1, 2006. *Crossref*, doi:10.2202/1538-0653.1549.

(Preliminary, will have more formal citations soon)

https://www.bbc.com/culture/article/20190531-napster-turns-20-how-it-changed-the-music-industry#:~:text=On%201%20June%201999%2C%20a,the%20boardrooms%20of%20record%20companies.
https://electronics.howstuffworks.com/gadgets/audio-music/spotify3.htm
https://pdos.csail.mit.edu/6.824/papers/pouwelse-btmeasure.pdf\
https://www.riaa.com/wp-content/uploads/2004/01/art-the-impact-of-digital-file-sharing-on-the-music-industry-michel-2006.pdf
https://www.csoonline.com/article/2998952/sony-bmg-rootkit-scandal-10-years-later.html
https://www.wired.com/2009/12/1207riaa-sues-napster/#:~:text=With%20a%20bankrupt%20Napster%20unable,with%20loans%20totaling%20%2485%20million.

https://ieeexplore.ieee.org/abstract/document/911304
https://ieeexplore.ieee.org/abstract/document/925318