Guilty PlAESures

Michael Pepsin

Final Write-Up

5/14/2020

1. Abstract

Guilty PlAESures is a web app that uses AES encryption to encrypt MP3 files. Originally, the concept included anonymous emailing of the ciphertext and key to an email address input along with the MP3 file. Once these data are fed back into the web app, the user receives the original file. This paper outlines the methodology, motivation, and results of the project.

2. Motivation

My inspiration came from a song from Spongebob Squarepants. I stumbled across this song while letting my Spotify play some recommended tracks. When the song “Gary Come Home” came on, I actually enjoyed it quite a bit and was a little embarrassed to listen to it repeatedly several times in the days to come, even adding it to my main playlist. This project was conceived as a way to anonymously share music, requiring only an MP3 file of a song and the email of the intended recipient.

3. Related Work

A quick Google search of the phrase “mp3 encryption” rendered a result for MP3Crypt, a console mode application available on www.compuphase.com. This program is very old and uses H04x0 encryption, which is suited for audio. However, this means using a CD or USB to store the encryption keys and the MP3Crypt itself. There is also a GitHub repository that encrypts MP3 files and uses open ssl for decryption, but by its nature, the keys are stored client-side as plaintext, making it effectively useless against hackers. Guilty PlAESures is more oriented towards sharing music compared to MP3Crypt, and since the key it generates is unique to the MP3 file, it is more secure than the GitHub repo Encrypt/Decrypt.

4. Methodology

Guilty PlAESures uses AES encryption with keys of size 128. The web app is serviced by a Tomcat servlet for handling data from the HTML form. The servlet stores the file in temporary memory while it is encrypted. The java servlet file uses a suite of encryption and decryption methods in another file, mp3Driver.java, which uses the java.security and javax.crypto libraries.

The user of Guilty PlAESures supplies an MP3 file, which is converted to a byte array using the Apache Commons IO library. This byte array is encrypted, and output as a String for the user to copy. The encryption key is randomly generated, and output as a String with the ciphertext. The web app uses the HttpServlet method doPost to handle the MP3 file, because there are no limits on how large the data handled by doPost can be. This is opposed to doGet, which has a cap, and handles the decryption since it is processing only Strings. The encrypted String is encoded with the ISO-8859-1 charset. This is because it is output as a String and re-entered as a String. The ciphertext input for decryption must be converted back into a byte array before it is decrypted and reassembled as a file. Using any other charset breaks the decryption. Once the ciphertext is transformed back into a byte array, it is decrypted and written as a file to the disk. The only languages used were HTML for the web app and Java for all functionality.

5. Analysis

The web app successfully encrypts and decrypts MP3 files, but some of the original implementation concepts are missing. First, attempting to configure the Tomcat servlet to handle emails resulted in the servlet breaking twice. This, along with concerns of the approaching deadline caused me to abandon this function. In addition, the decryption part of the HTML form does not handle the ciphertext. Even for a small MP3 file, the ciphertext is still too long to be handled by the form. Perhaps it is possible to use doPost for handling both parts of the form since the data limit of doGet is potentially to blame. However, since doPost already handles the encryption form, its code would quickly become messy. Regardless of these issues, the web app can still be used to encrypt MP3 files, and the implementation of mp3Driver.java is very useful. Another program, testEncryption.java, was implemented to test the encryption and decryption. It uses the same mp3Driver methods to encrypt and decrypt MP3 files. Used as a Java application (as opposed to the Tomcat-serviced web app), it can be of personal use for encrypting and decrypting.

6. Conclusion

Certain core facets of the Guilty PlAESures concept are not implemented. Central among these is the email functionality, which is key to the anonymous sharing aspect of Guilty PlAESures. Nonetheless, I am proud to have gotten the encryption and decryption working my first time using Tomcat. The Tomcat servlet turned out to be an excellent method. It allowed me to avoid using JavaScript, a language I am not experienced in. The absence of JavaScript also kept the project “cleaner”, because functionality was not split between Java and JS unnecessarily. The Tomcat servlet also significantly simplified the process of servicing the web app, as a full-fledged Apache server would have been more work to implement.

I may also have a further use for mp3Driver.java someday. It is fully equipped to handle MP3 encryption, and contains a method mp3Parser which can be used to parse MP3 files into byte arrays. This method is deprecated by the Tomcat servlet, but mp3Driver.mp3Parser() can be

used in the absence of the servlet, resulting in a robust suite for encryption and decryption. Overall, the complete concept for Guilty PlAESures is not implemented, but it is definitively equipped to handle its main functionality, the encryption and decryption of music files.

Sources:

https://www.compuphase.com/mp3/mp3crypt.htm

https://gist.github.com/mmcc/5862775