Comparison of music streaming services' encryption concepts

Laura Hausmann

April 10, 2019

Contents

1 Preface

(Almost) everyone is using music streaming services nowadays. Spotify being the most popular, offering over 12 million tracks on demand, for a flat fee of $9.99 a month. But contrary to buying CDs or digital music in stores like 7digital or iTunes, you don't own the music. As soon as you stop paying for the subscription, or the company goes bankrupt, all of it is gone. But what exactly is preventing you from "just downloading" the tracks from all these services? It's time to talk about DRM.

2 DRM

DRM, or Digital Rights Management, is a form of copy protection or, as used in streaming services, a method of restricting usage rights. Instead of files containing the raw audio stream in a standardized container like MP3, AAC, M4A, FLAC or ALAC they are either a completely proprietary format, containing a reference to the user/account and their respective license/usage restrictions (e.g. old iTunes Downloads use M4B, an encrypted M4A container), or, as used in most streaming services, a simple encryption scheme over a standardized container.

Originally iTunes and the like implemented DRM for purchased music as well, although they dropped that practice in 2009, after numerous antitrust lawsuits regarding their FairPlay/M4B DRM, which was only playable on their own iPod players.[1]

3 Streaming Services

DRM strategies differ a lot between the different services. Some (mainly YouTube Music) skip DRM altogether, relying on security by obscurity in changing the website up a bit every few days, making automation harder. This can, however, be bypassed with relative ease by using a browser extension that dynamically extracts loaded media sources.

[1] The Guardian. *Apple drops DRM copy protection from millions of iTunes songs.* 2009. URL: `https://www.theguardian.com/technology/2009/jan/06/apple-drops-itunes-copy-protection`.

Spotify is an outlier, in that there is no publicly known bypass for their encryption system, which makes use of AES-128-encrypted AAC streams, with an additional layer of Widevine DRM to obscure the encryption keys. This is most likely the reason for the encryption still not being public. For a limited time, it was possible to use the "App API" to download audio directly, which was both limited to premium users and required an API key that, while not being easy to get from Spotify directly, was shared on sites like GitHub Gist many times. This ultimately led to the discontinuation of that service, being replaced with the much more restricted "Web API". The most popular currently working method of extracting audio from Spotify is by combining the music control API of the respective operating system with the raw sound output, reencoded to usually a MP3 stream. This has obvious problems, such as other system audio being recorded with the music, as well as quality loss due to reencoding the compressed audio stream.

3.1 Deezer

The first streaming service that was cracked fully was Deezer. They use several encryption algorithms for different parts of their website, but the actual audio data is encrypted using Blowfish in CBC (Cipher Block Chaining). When using the web player, even free users are allowed to play audio, and media extraction extensions can download a file with .mp3 extension, which upon closer inspection contains a normal MP3 stream, stripped of all tags. When played however, it becomes obvious that a form of encryption is still being used: there are a lot of audio glitches to be heard. It was soon figured out that only every third audioframe/block (2048 bytes) is actually encrypted. Since the streams are playable by a free user, and widevine or similar is not being used, the encryption key was soon found in the obfuscated JavaScript. Every track has a unique encryption key, that can be calculated using the ASCII-MD5 hash of the track ID, and a static secret. The algorithm used is also contained in the clientside javascript.

3.1.1 Summary of Flaws

Deezer made a few mistakes over the last few years. This is a (likely incomplete) list of them and ideas on how to avoid them.

1. Encryption keys are only pseudo-dynamic. In this case, they are generated with the trackId (in the url, locatable via the *public, unauthenticated* API and a static master key, which can be found in the web player's JavaScript.
   - Authenticated, logged, rate-limited API endpoint /v2/getTrackKey. Possibly user-specific keys to avoid a central database.
2. Any user is able to get the streaming URIs in **any** quality (up to 16/44.1 FLAC) from their authenticated API. Including free users with unconfirmed emails.
   - At the very least check the user's subscription status before returning the URI. As can be seen above, encryption isn't stopping downloads.
3. The official API, while offering authentication, has almost no endpoints that actually check for authentication.
   - If ignoring authentication, at least use some form of rate-limiting. You can easily pull 5 Gigabit/s from their CDN. For hours.

3.2 Qobuz

Qobuz is an outlier. Once a site to purchase Hi-Res music, it now offers streaming subscriptions as well. But that's not the only odd thing. They have an official API, documented on GitHub, which includes an endpoint (/track/getFileUrl) that returns a URL pointing to an unencrypted FLAC stream.[2] Combined with flawed authentication, this becomes a problem. The endpoint is authenticated with:

- AppID and AppSecret
- Username and Password ⇒ User Auth Token

Again, this leads to a few flaws:

[2]URL: `https://github.com/Qobuz/api-documentation`.

3.2.1 Summary of Flaws

1. The web player uses the same API. Combined with the great documentation, it is very easy to find the AppID and AppSecret in the clientside JavaScript, no painful reversing needed.
    - There is not much that can be done here, other than an adding an obscurity layer or rate-limiting the API (again, it possible to pull over 5 Gigabit/s)

3.3 Tidal

Tidal was one of the first streaming services to offer FLAC-streaming, now with up to 24/384 (packaged in MQA) resolution. They are also the only service in question that employs rate limiting. Libraries that support getting MP3 or FLAC are readily available, although the only program I could find that can download MQA is closed-source. But reversing the desktop client wasn't too hard and I soon found the parameters for getting the MQA stream. Tidal uses AES-128 in CBC mode as well, returning an encrypted file key including a nonce when requesting the stream. This is a great design choice, because it allows them to patch the download interface in the future.

3.3.1 Summary of Flaws

1. The current encryption method and nonce use a single master key, which has been known for a few years now. This could have easily been patched, as the rest of the server infrastructure is pretty secure.