Media encryption

- Clear Key encryption
  - Create a key
  - Create an IV
  - Encrypt with Clear Key
  - Create a key information file
  - Encrypt for HLS
- Widevine encryption
  - Encrypt with Widevine
- Media conversion sequence
  - DASH/WebM
  - DASH/MP4
  - HLS/MP4

In this section we are going to cover two different strategies for encrypting your media, and some practical examples on how to use them with FFmpeg and Shaka Packager. The two strategies for encryption we'll discuss are Clear Key and using a service like Google Widevine. Both strategies are a form of digital rights management (DRM) to control what users can do with your media. However, one is inherently less secure than the other due to the way keys are passed for authentication and is why a DRM service might make more sense.

The primary DRM services for the web are Google Widevine, Microsoft PlayReady and Apple FairPlay, but we will not be covering all of them in this article. However, if you are targeting all the modern browsers you are likely going to be using all three DRM services.

Conversion and encryption is done with these applications:

- Shaka Packager (#shaka on Stack Overflow)
- FFmpeg (#ffmpeg on Stack Overflow)
- OpenSSL (#openssl on Stack Overflow)

Clear Key encryption

First, you should have a good understanding of what Clear Key is and isn't before using it. When you do ***not*** want to use an existing DRM service and feel basic encryption of you media is a viable option, you would use Clear Key. But, keep in mind that this type of encryption does not provide the same level of security as using one of the DRM services. This is because the key value pair is not encrypted under another key, unlike encrypted keys which are generated by a decryption key that is stored on a licence server. Additionally, Clear Key sends the key value pair as plain text, so while you are encrypting your media the key to decrypt it is not a secret.

Create a key

You can use the same method to create a key for both DASH and HLS. Do this using OpenSSL. The following will create an encryption key made of 16 hex values.

```
openssl rand -hex 16 > media.key
```

This command creates a file that could contain white space and new line characters, which are not allowed by Shaka Packager. You'll need to open the key file and manually remove all whitespace including the final carriage return.

Create an IV

Next we can generate an initialization vector (IV).

```
openssl rand -hex 16
6143b5373a51cb46209cfed0d747da66
```

Make a note of the output, you'll need this later.

Encrypt with Clear Key

The following example uses Shaka Packager with raw keys, where `keys` and `key_ids` are provided to Shaka Packager directly. Read the documentation for more examples.

For the `key` flag, use the key created earlier, which is stored in the `media.key` file. However, when entering it on the command line, be sure you've removed its whitespace. For the `key_id` flag, repeat the `media.id` value or use the IV value generated above.

```
packager \
  input=glocken.mp4,stream=audio,output=glockena.m4a \
  input=glocken.mp4,stream=video,output=glockenv.mp4 \
  --enable_fixed_key_encryption \
  --keys
label=audio:key=INSERT_AUDIO_KEY_HERE:key_id=INSERT_AUDIO_KEY_ID_HERE,label=video:key=INSERT_V
IDEO_KEY_HERE:key_id=INSERT_VIDEO_KEY_ID_HERE
```

Note that we can also encrypt both `video` and `audio` streams using the same `keys` by omitting the label argument instead of defining a different set of `keys` per-stream. For example, `--keys label=:key=INSERT_KEY_HERE:key_id=INSERT_KEY_ID_HERE`. However, this would decrease the level of security when protecting your streams.

Create a key information file

To encrypt for HLS you need a key information file in addition to a key file. A key information file is a text file with the format below. It should have the extension `.keyinfo`. For example: `encrypt.keyinfo`.

```
key URI
key file path
private key
```

The key URI is where the `media.key` (created above will be located on your server. The key file path is its location relative to the key information file. Finally, the private key is the contents of the `media.key` file itself, or the IV you created before. For example:

```
https://example.com/keys/media.key
/path/to/media.key
6143b5373a51cb46209cfed0d747da66
```

Encrypt for HLS

```
packager \

'input=input.mp4,stream=video,segment_template=output$Number$.ts,playlist_name=video_playlist.
m3u8' \

'input=input.mp4,stream=audio,segment_template=output_audio$Number$.ts,playlist_name=audio_pla
```

```
ylist.m3u8,hls_group_id=audio,hls_name=ENGLISH' \
  --hls_master_playlist_output="master_playlist.m3u8" \
  --hls_base_url="http://localhost:5000/"
```

This command will accept a key with either 16 or 32 characters.

```
ffmpeg -i myvideo.mov -c:v libx264 -c:a aac -hls_key_info_file encrypt.keyinfo myvideo.m3u8
```

Widevine encryption

Now you know what Clear Key encryption is and when to use it. But, when should you use a DRM service for additional security? This is where Widevine, or another service, would be used to securely encrypt and decrypt your media. Widevine supports MPEG-DASH and HLS and is a DRM from Google. Widevine is used by the Google Chrome and Firefox web browsers, Android MediaDRM, Android TV, and other consumer electronics devices that use Encrypted Media Extensions and Media Source Extensions, where Widevine decrypts content.

Encrypt with Widevine

Most of the examples in this article used Clear Key encryption. However, for Widevine you will want to replace the following options.

```
--enable_fixed_key_encryption \
--enable_fixed_key_decryption \
--keys label=:key=INSERT_KEY_HERE:key_id=INSERT_KEY_ID_HERE
```

Everything in the demultiplexer (demux) command except the name of your files and the `--content-id` flag should be copied exactly from the example. The `--content-id` is 16 or 32 random hex digits. Use the keys provided here instead of your own. Read the Shaka Packager documentation on using the Widevine Key Server for more examples.

1. Demux (separate) the audio and video, encrypt the new files, and output a media presentation description (MPD) file.

   ```
   packager \
     input=tmp_glocken.mp4,stream=video,output=glocken_video.mp4 \
     input=tmp_glocken.mp4,stream=audio,output=glocken_audio.m4a \
     --enable_widevine_encryption \
     --key_server_url
   ```

```
"https://license.uat.widevine.com/cenc/getcontentkey/widevine_test" \
  --content_id "fd385d9f9a14bb09" \
  --signer "widevine_test" \
  --aes_signing_key
"1ae8ccd0e7985cc0b6203a55855a1034afc252980e970ca90e5202689f947ab9" \
  --aes_signing_iv "d58ce954203b7c9a9a9d467f59839249"
```

2. Remux (combine) the audio and video streams. If you're using a video framework, you may not need to do this.

```
ffmpeg -i glocken_video.mp4 -i glocken_audio.m4a -c copy glocke.mp4
```

The default pixel format `yuv420p` is used in all the `ffmpeg` remux examples in this article because one isn't supplied in the command line. The `ffmpeg` command will give you an error message that it is deprecated. We've chosen not to override the default because, though deprecated `yuv420p` is still the most widely supported.

Media conversion sequence

This section shows in order commands needed to get from a raw `.mov` file to encrypted assets packaged for DASH or HLS. For the sake of having a goal to illustrate, we're converting a source file to a bitrate of 8Mbs at a resolution of 1080p (1920 x 1080). Adjust these values as your needs dictate.

Not all steps are possible with Shaka Packager alone, so we'll also use FFmpeg when needed.

DASH/WebM

1. Convert the file type and codec.
   For this command you can use either `liborbis` or `libopus` for the audio codec.

```
ffmpeg -i glocken.mov -c:v libvpx-vp9 -c:a libvorbis -b:v 8M -vf setsar=1:1 -f webm
tmp_glocken.webm
```

2. Create a Clear Key encryption key.

```
openssl rand -hex 16 > media.key
```

3. Demux (separate) the audio and video, encrypt the new files, and output a media presentation description (MPD) file.

```
packager \
  input=tmp_glocken.webm,stream=video,output=glocken_video.webm \
  input=tmp_glocken.webm,stream=audio,output=glocken_audio.webm \
  --enable_fixed_key_encryption \
  --enable_fixed_key_decryption \
  --keys label=:key=INSERT_KEY_HERE:key_id=INSERT_KEY_ID_HERE \
  --mpd_output glocken_webm_vod.mpd
```

4. Remux (combine) the audio and video streams. If you're using a video framework, you may not need to do this.

```
ffmpeg -i glocken_video.webm -i glocken_audio.webm -c copy glocken.webm
```

DASH/MP4

1. Convert the file type, video codec and bitrate.

```
ffmpeg -i glocken.mov -c:v libx264 -c:a aac -b:v 8M -strict -2 tmp_glocken.mp4
```

2. Create a Clear Key encryption key.

```
openssl rand -hex 16 > media.key
```

3. Demux (separate) the audio and video, encrypt the new files, and output a media presentation description (MPD) file.

```
packager \
  input=tmp_glocken.mp4,stream=video,output=glocken_video.mp4 \
  input=tmp_glocken.mp4,stream=audio,output=glocken_audio.m4a \
  --enable_fixed_key_encryption \
  --enable_fixed_key_decryption \
  --keys label=:key=INSERT_KEY_HERE:key_id=INSERT_KEY_ID_HERE \
  --mpd_output glocken_mp4_vod.mpd
```

4. Remux (combine) the audio and video streams. If you're using a video framework, you may not need to do this.

```
ffmpeg -i glocken_video.mp4 -i glocken_audio.m4a -c copy glocken.mp4
```

HLS/MP4

HLS only supports MP4, so first you'll need to convert to the MP4 container and supported codecs.

1. Convert the file type, video codec, and bitrate.

```
ffmpeg -i glocken.mov -c:v libx264 -c:a aac -b:v 8M -strict -2 glocken.mp4
```

2. Create a Clear Key encryption key.

```
openssl rand -hex 16 > media.key
```

3. Create a key information file

```
packager \

'input=glocken.mp4,stream=video,segment_template=output$Number$.ts,playlist_name=vide
o_playlist.m3u8' \

'input=glocken.mp4,stream=audio,segment_template=output_audio$Number$.ts,playlist_nam
e=audio_playlist.m3u8,hls_group_id=audio,hls_name=ENGLISH' \
  --hls_master_playlist_output="master_playlist.m3u8" \
  --hls_base_url="http://localhost:5000/" \
  --enable_fixed_key_encryption \
  --enable_fixed_key_decryption \
  --keys label=:key=INSERT_KEY_HERE:key_id=INSERT_KEY_ID_HERE
```

That was a lot to digest, but hopefully you are now able to encrypt you media with confidence. Next we'll show you how to add media to your site.

□□□ #3
□ □□□ □□□ 2026-10-31 22:56:44 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:33 UTC