WIDEVINE®

Encoding and Packaging

version 1.1

Contents

Version	Date	Description	By
1.0	11/8/2016	Initial revision	Alex Lee
1.1	2/27/2017	Clarification updates	Alex Lee

Summary

This document provides:

- A basic video primer with an emphasis on generating DASH-compatible content.
- Use of DASH and Common Encryption in content packaging.
- Coverage of best practices and recommendations.
- An introduction to the Widevine Shaka Packager.

Contact Us

For any questions, please contact Widevine from our website - www.widevine.com.

Introduction to Encoding

This section is designed to provide a quick layman's understanding of how media is created, what is used for media, common definitions and terminology, and supported codecs for DASH playback.

Let's start with the basic elements of a media file.

Elementary Stream

An elementary stream (ES) is essentially the encoding of media that's perceptible to the user. Every elementary stream contains a single media data type (audio, video, subtitles, captions). The sum of several elementary streams allows for cohesive media playback experience on any given platform.

The content of an elementary stream is dependent on the data format it holds, a codec (**coder-decoder**) for video or audio. Elementary streams are broken down into frames and encoded by codecs. A frame (or media sample) is typically referred to as a still image for video or a few milliseconds of audio. It contains information to render a specific video or audio scene at that specific point in time. A collection of frames would complete a video or audio clip, similar to a flip book.

What this essentially means is audio and video content is stored as frames encoded (compressed) by a codec. Each codec conforms to its own specifications and every codec strives to provide the best media quality for the least amount of resources used (processing, time, efficiency, size).

Since we are referring to audio and video content, the reference for a frame is based on time (in milliseconds). Audio frame durations are typically within the 20-40ms range while video frames vary and are usually expressed in terms for frames/second (fps)..

Codecs

The most common codecs used for video and audio processing in use today are:

Video	Audio
AVC (H264)	AAC
HEVC (H265)	Opus
VP9	Vorbis
AV1 (coming soon)	DTS
	Dolby Digital (AC-3, EAC-3)

Codecs may be closed-source commercial products while others are open-source and free-to-use with or without licensing requirements. Typically, codec selection is based on client platform support. DASH presentations support multiple codecs (i.e. mixing and matching different codecs within the same content).

Containers

Media containers are the grouping of one or more elementary streams into a data stream (in this case, a file). A container is also a format specification that describes elements of the data streams (timing, structure, and media information) that it holds.

The elementary stream metadata that is added to a container include (not a complete list):

- Codec type
- Codec-specific configuration data
- Video height and width
- Video frame rate
- Audio sampling rate
- Audio channels
- Frame timing and ordering information

Container formats

Here are examples of container formats commonly used for streaming media.

MPEG2-TS (.ts)

- Optimized for transmission over a closed network for broadcast systems.

ISO-BMFF (.mp4)

- Designed as a next-generation container format by Apple and adopted by ISO/IEC.
- There are multiple file extensions allowed for this format, however, we are focused on MPEG4 (mp4).
- All data within this format is organized into *boxes*. Each box type represents a different type of data element contained within. Boxes may also contain other boxes.
- Fragmented MP4 is a variant of ISO-BMFF optimized for streaming.

Matroska (.mkv, .webm)

- An open-sourced container format that is designed to be more efficient than ISO-BMFF.
- Uses a more compact version of metadata representation compared to MPEG4, which potentially leads to a faster startup time for playback.

When creating DASH compliant media, the specification requires only a **single elementary stream be present per container**.

Now that we have covered what media files consist of, how are these files created?

Transmuxing and Transcoding

Transcoding is a processing of an elementary stream consisting of decoding followed by re-encoding to the same or another codec, while potentially altering its encoding characteristics (video resolution, bitrate, audio sampling rate, etc.) . The resulting elementary stream may be stored in a different container format than the original.

Transmuxing is the act moving elementary streams from one container format to another without manipulating the actual streams. All audio visual data remains unchanged.

Transmuxing is a less processing-intensive effort compared to transcoding.

Why is transmuxing useful?
To deliver the same content to different platforms that support varying formats.

For example:

To deliver identically encoded streams over different delivery network to separate devices, using MPEG2-TS for a IPTV cable box (i.e. Comcast Cable) and ISO-BMFF for DASH streaming to an Android phone (Comcast Xfinity).

Why is transcoding useful?

- Optimize media for better user experience on playback.
- Caters to the adaptive streaming use-case where different media stream qualities are generated to meet varying bandwidth requirements.
- To deal with device playback constraints. The media format applicable for one device may not be compatible with another; as such, the media must be converted.
- To meet business needs
    - Varying levels of quality to meet device streaming capacity
    - Contractual obligations

The next section covers the structure of a video stream.

Video Stream Structure

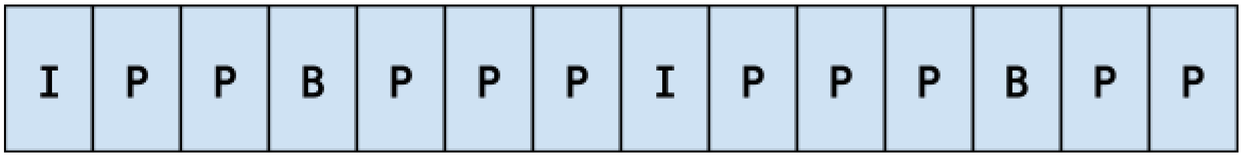

Figure 1 - represents an arbitrary section of a video stream.

As previously stated, video is represented as a sequence of frames over time, where there are groupings of frames that are similar in representation.

Group of Pictures (GOP) Structure

A GOP is:

- a self-contained decodable sequence of frames.
- normally represented as a short sequence of pictures.
- begins with a key frame, and ends before the next key frame..

There are different types of frames contained in a GOP:

- I-frame (Intra)
    - An I-frame (Intra-coded) is a full picture, much like a standard JPEG image file. a.k.a a key frame.
    - **Every GOP starts with an I-frame** as it represents a complete visual representation of a picture.

    - Because it contains the complete picture, an I-frame is usually larger in size compared to other frame types.
    - Also known as a key frame.
- P-frame (Predictive)
    - Contains differences in the picture from previous frames.
- B-frame (Bi-Predictive)
    - Similar to a P-frame, but may be encoded as differences from subsequent pictures. Because of this, it is usually decoded out of order.

If I-frames are so complete, why is there a need for P or B frames? Media consisting of all I-frames would be very large, rendering it unusable for mass consumption. B and P frames are a fraction of the size of an I-frame (since they only display the picture differences).

Why are GOPs important?

- Allows for seek points.
    - Provides the ability to move forward or backward in the video and always have the ability to start by displaying a complete picture (the I-frame). A P-frame or B-frame will only display the visual differences from preceding or successive pictures, not a complete picture.
- Enables higher efficiency for encoding.
    - Inserting an I-frame at scene changes allows for smaller B and P frames, reducing file size.
    - A scene change is a sequence of pictures that has no reference to previous pictures. An example of a scene change is when the camera angle changes.
- Facilitate video adaptation.
    - A video decoder must always start from an I-frame to provide a proper picture reference point.
    - Video adaptation is performed by switching from poor quality media to high quality media (or vice versa). Without using I-frames, the end-user visual experience would be extremely poor because the act of switching video quality will potentially render B or P frames (going from a full picture render to partial).

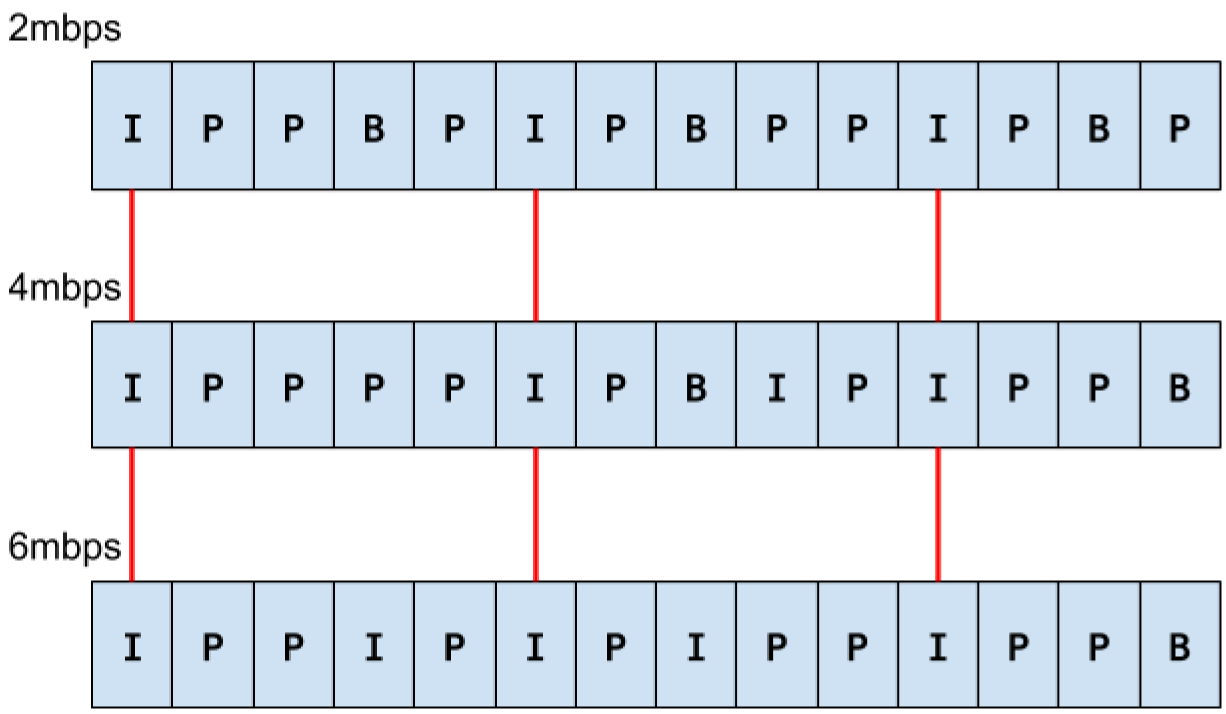
2mbps
I P P B P I P B P P I P B P
4mbps
I P P P P I P B I P I P P B
6mbps
I P P I P I P I P P I P P B

Figure 2 - represents 3 video streams with aligned GOPs (i-frames).

There are 2 types of GOP.

- Open
    - Open GOPs start with a B-frame that is able to look at the last P-frame from the preceding GOP as well as the first I-frame of its own GOP.
    - An open GOP has a reliance on the preceding GOP.
- Closed
    - Closed GOPs cannot contain any frame that refers to a frame in the previous or next GOP.
    - A closed GOP is **usually required** when generating media to allow a smoother adaptation experience.

A GOP length or size is dependent on the number of frames per GOP. A longer GOP is more efficient (a larger grouping of similar pictures), however, provides fewer seek or adaptation points.

Encoding media

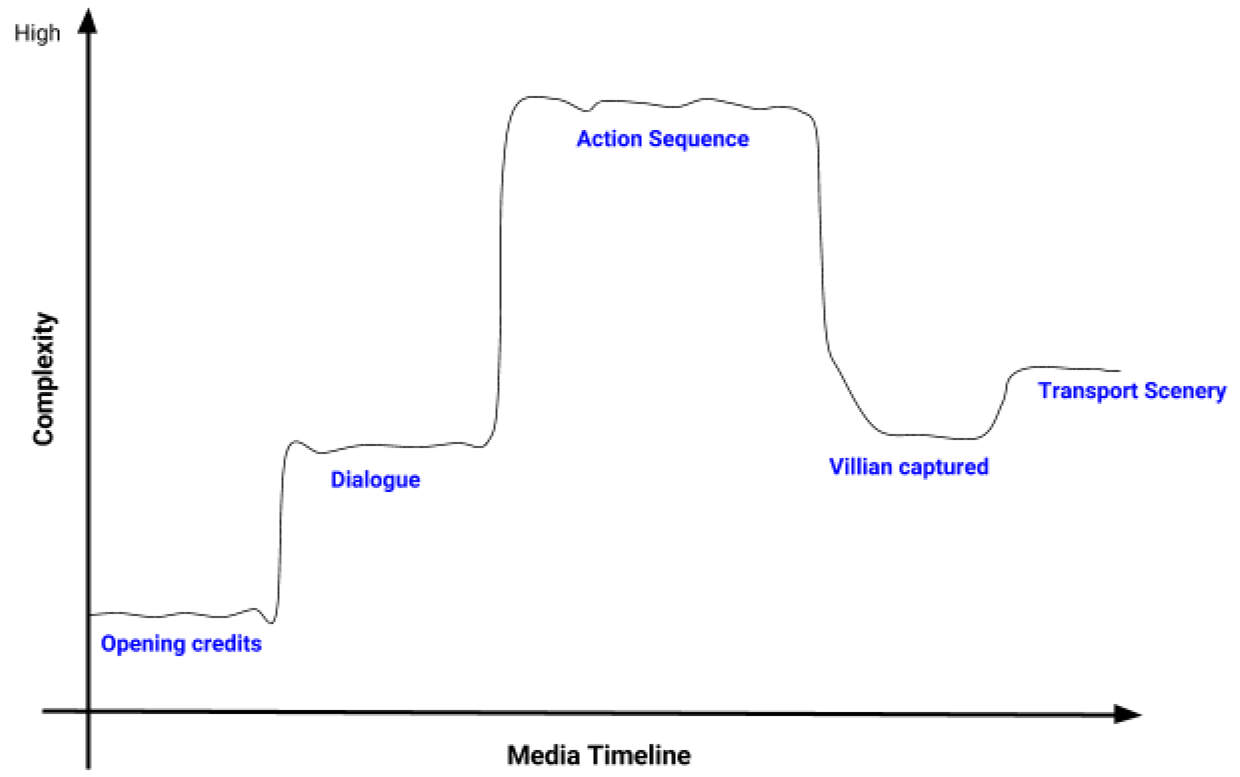

Figure 3 - represents examples of varying complexity based on media timeline.

Media is not uniform. For example, a movie will contain many scenes - action sequences, dialogues, romance and more. The amount of data required to encode a scene depends on how much the frames change between one and the next (complexity). An action sequence would require more data to encode correctly compared to a scene of a sleeping baby.

There are many considerations for encoding, however, the primary decision first comes down to bitrate.

Constant Bitrate (CBR)

CBR refers to content that is encoded at a specific bitrate, uniformly across its entirety.

e.g. 6 Mbit/sec = 1 second of media requires 6MB

For a low-complexity scene, a CBR configuration will force the codec to add padding (empty data) to meet the desired bitrate since it has to be constant.

For a high-complexity scene, a CBR configuration will force the codec to compromise on scene quality if the required data to render the scene exceeds the CBR limit. The scene is too complex to store in the amount of information allowed, resulting in graininess, artifacts, skipping.

Variable Bitrate (VBR)

VBR allows the codec to use fewer bits when it is not needed, saving them for more complex scenes.

The basic principle is to set a target bitrate and allow for some level of variance over the length of the media, to achieve the overall target bitrate.

VBR is generally recommended as it results in higher-quality encodings compared to CBR encodings at the same bit rate.

What happens when the bitrate exceeds the target for extended periods of time? The end-user device will be unable to display the frames correctly since it's expecting X bitrate but receiving X+Y bitrate instead. This can be shown as stuttering and artifacts (pixelation).

To work around these limitations, a video buffer verifier (VBV) is specified. The VBV manages VBR variance. It specifies a maximum bitrate over a rolling buffer.

In our tests, for the majority of current devices, the VBV should be set to twice the target bitrate.

Aspect Ratio

Aside from bitrate, there are other parameters to control the display or viewing of pictures - namely, the aspect ratio to preserve the original picture, preventing visual distortion.

There are 3 distinctions of aspect ratio - Sample Aspect Ratio, Pixel Aspect Ratio and Display Aspect Ratio.

The display aspect ratio is most understood and referred to when discussing aspect ratios. It is a reflection of what we see in media. References to terms like 4:3 or 16:9 refer to the display aspect ratio. A common method is to define the display aspect ratio (DAR) and let your encoder software calculate the SAR and PAR accordingly.

Using encoding profiles

Encoding profiles refer to a set of parameters that generates media with specific properties. For example, an encoding profile for SD vs HD content would not be identical as there would be, at minimum, a change in the display resolution.

These profiles vary per codec. Every codec defines its own specification and features - for example: H264 uses Baseline, Main and High profiles. Even more importantly, encoding profile support varies from device to device. Therefore, it is paramount to ensure that a device is a capable of playback for any given encoding profile.

Common encoding profiles

Codec	Profiles
AAC	AAC-LC AAC-HE (SBR - spectral band replication) AAC-HEv2 (SBR and PS - parametric stereo)
VP9	0 (8-bit) 1 (8-bit) 2 (10-bit) (12-bit) 3 (10-bit) (12-bit)
H264	Baseline, Main, High
H265	Main, Main10

Best Practices

The section outlines a series of recommendations and best practices from encoding to encryption and enabling playback on various client platforms. These recommendations serve as guidance on the design and implementation of EME, CENC, DASH support from Widevine for both server and client pieces. To ensure maximum playback compatibility across all client platforms, the lowest common denominator for encoding profiles should be used.

The table below illustrates the most common audio and video codec combinations for various video resolutions:

Resolution	Widevine Client Security Level	Video Codec	Audio Codec
SD	L3	H264 (MP4)	AAC
HD	L1	H264 (MP4)	AAC DTS Dolby Digital
UHD	L1	HEVC (MP4) VP9 (WebM)	AAC DTS Dolby Digital

General recommendations

- Use variable bitrate (VBR) tracks with a reasonable video buffer verifier (VBV) value.
    - VBV = 1.5 - 2x target bitrate.
- All files must have closed GOPs and identical IDR frame structure.
- IDR frame separation – approx 3 seconds (lowest value consistent with good image quality).
- Key frames must be at the same exact intervals across all track types.
- **DASH REQUIREMENT** - 'moov' atom should immediately follow the 'ftyp' atom.
- ISO-BMFF Chunks should contain no more than 1 second worth of sample data.

H264 Encoding Profiles

(referenced from the Android Compatibility Definition Document)

Video Resolution	Profile and Level	Bitrate	FPS	GOP
240p	Baseline 3.0	800 kbps	24	3s
480p	Main 3.1	2000 kbps	24	3s
720p	Main 4.0	8000 kbps	24	3s
1080p	High 4.2	20000 kbps	24	3s
4K / UHD	High 5.1	30000 kbps	24	3s

Example encoding syntax using ffmpeg

```
ffmpeg -i <source_file> -an \
-vf "scale=1280:trunc(ow/a/2)*2" \
-c:v libx264 -profile:v main -level:v 4.0 \
-x264opts scenecut=0:open_gop=0:min-keyint=72:keyint=72 \
-movflags +faststart \
-minrate 2M -maxrate 2M -bufsize 3M -b:v 2M  <output_file>
```

Ffmpeg parameters

Parameter	Description
-an	No audio, video only
-vf	The options used here will resize the video to the desired resolution while maintaining the original aspect ratio
-c:v libx264	Selects the codec library
-profile	Specifies the codec profile
-level	Specified the codec level
-x264opts	Declares additional options: Enforce closed GOPs Specify 3s GOP intervals (72 for 24fps source content)
-movflags +faststart	Optimizes the output MP4 file format for streaming

-minrate	Minimum bitrate
-maxrate	Maximum bitrate
-b:v	Target bitrate
-bufsize	Video buffer size

HEVC Encoding Profiles

(referenced from the Android Compatibility Definition Document)

Video Resolution	Profile and Level	Bitrate	FPS	GOP
240p	Main 3.0	600 kbps	24	3s
360p	Main 3.0	1600 kbps	24	3s
480p	Main 3.0	3000 kbps	24	3s
720p	Main 3.1	4000 kbps	24	3s
1080p	Main 4.1	10000 kbps	24	3s
4K / UHD	Main10 5.1	20000 kbps	24	3s

Example encoding syntax using ffmpeg

```
ffmpeg -i <source_file> -an \
-vf "scale=1280:trunc(ow/a/2)*2" \
-c:v libx265 \
-x265-params
level=4.0:scenecut=0:open_gop=0:min-keyint=72:keyint=72 \
-movflags +faststart \
-minrate 4M -maxrate 4M -bufsize 6M -b:v 4M  <output_file>
```

Ffmpeg parameters

Parameter	Description
-an	No audio, video only
-vf	The options used here will resize the video to the desired resolution while maintaining the original aspect ratio
-c:v libx265	Selects the codec library
-x265-params	Declares additional options: Specify HEVC level Enforce closed GOPs Specify 3s GOP intervals (72 for 24fps source content)
-movflags +faststart	Optimizes the output MP4 file format for streaming

-minrate	Minimum bitrate
-maxrate	Maximum bitrate
-b:v	Target bitrate
-bufsize	Video buffer size

VP9 Encoding Profiles

(referenced from the Android Compatibility Definition Document)

Video Resolution	Profile and Level	Bitrate	FPS	GOP
240p	Profile 0 (8-bit)	600 kbps	24	3s
360p	Profile 0 (8-bit)	1600 kbps	24	3s
720p	Profile 0 (8-bit)	4000 kbps	24	3s
1080p	Profile 0 (8-bit)	5000 kbps	24	3s
4K / UHD	Profile 2 (10-bit)	20000 kbps	24	3s

Example encoding syntax using ffmpeg

```
ffmpeg -i <source_file> \
-vf "scale=1280:trunc(ow/a/2)*2" \
-c:v libvpx-vp9 -keyint_min 72 -g 72 -profile:v 0 \
-threads 4 -tile-columns 6 -frame-parallel 1 \
-speed 1 -auto-alt-ref 1 -lag-in-frames 25 \
-an -minrate 4M -maxrate 4M -bufsize 4M -b:v 4M \
-f webm -dash 1 <output_file>
```

Ffmpeg parameters

Parameter	Description
-an	No audio, video only
-vf	The options used here will resize the video to the desired resolution while maintaining the original aspect ratio
-c:v libvpx-vp9	Selects the codec library
-profile:v	Specifies VP9 profile
-f webm	Specifies webm container output
-keyint_min	Specifies minimum keyframe interval
-g	Specifies keyframe interval
-threads 4 -tile-columns 6	libvpx options optimized for VOD streaming media

-frame-parallel 1 -speed 1 -auto-alt-ref 1 -lag-in-frames 25 -dash 1	See the VP9 Encoding Guide.
-minrate	Minimum bitrate
-maxrate	Maximum bitrate
-b:v	Target bitrate
-bufsize	Video buffer size

Content Encryption

Encryption best practices can be distilled into a simple statement: separate keys should be used across different types of content (audio, video, resolution).

Encryption Recommendations from Least Secure to Most Secure

Audio	Video
No encryption	Single content key for all tracks
Separate content key for audio tracks	Separate content key for each video resolution group (SD, HD, UHD)
Separate content key for audio tracks	Separate content key for each video track

Playback Security Levels for Chrome and iOS

The recommended security level setting for VIDEO tracks will be to specify SW_SECURE_DECODE. The only supported security level setting for AUDIO tracks is SW_SECURE_CRYPTO. Security level settings are specified by your license proxy implementation on a per track basis. The table below provides the recommended security level settings per Chrome platform.

Platform	Video	Audio
Browser (PC, Mac, Linux) iOS	SW_SECURE_DECODE (L3)	SW_SECURE_CRYPTO (L3)
ChromeOS	SW_SECURE_DECODE (L3) HW_SECURE_ALL (L1)	SW_SECURE_CRYPTO (L3)

Using the Widevine Shaka Packager

Widevine provides a reference open-source CENC packaging solution - Shaka Packager. The GiHub page provides documentation on how to build and use the packager. We highly recommend joining the GitHub user group to keep up with the latest information and to ask questions.

The following sections covers how to install and use the Shaka Packager.

Install dependencies

The Shaka Packager supports a variety of OSes, this section of the document will focus on using Ubuntu Linux (14.04 or higher). For the sake of simplicity, all commands are executed as `user=root`.

To ensure you have the basic build environment, run the following commands as `user=root` :

```
# apt-get install -y build-essential gcc wget git g++ subversion
```

It should also prompt to install any additional dependencies, say `Yes`.

Build the Shaka Packager

You may now execute the commands as a normal Linux user.

1. Packager source is located at https://github.com/google/shaka-packager
2. Pull gclient and ninja from Chrome Depot Tools:

```
$ git clone
https://chromium.googlesource.com/chromium/tools/depot_tools.git
```

3. Add depot_tools to your PATH:

```
$ export PATH=$PATH:`pwd`/depot_tools
```

   a. Note that the above command contains a ` (back quote).
   b. You may want to add this to your .bashrc file or your shell's equivalent so that you don’t need to reset your $PATH manually each time you open a new shell.

4. Run the following commands in sequence.

```
$ mkdir shaka-packager
$ cd shaka-packager
$ gclient config https://www.github.com/google/shaka-packager.git
--name=src
$ gclient sync
```

5. Run the following to verify the directories were created:

```
$ ll src/
```

6. Build using ninja to create the reference binaries for use. This takes a while, grab a cup of coffee.

```
$ ninja -C src/out/Release
```

7. Verify the binaries exist.

```
$ cd src/out/Release && ls
```

The following binary files will be used:

a. **packager**
   i. Used to analyze and encrypt media files
b. **mpd_generator**
   i. Used to generate playlist files

For ease of use, you can add the Release directory to your PATH or copy the 2 binaries to a directory in your PATH:

```
$ export PATH=$PATH:$HOME/shaka-packager/src/out/Release
```

Getting Help with Shaka Packager

To view available options, run:

```
$ packager --help
$ mpd_generator --help
```

Audio and Video Stream Analysis

Use the `--dump_stream_info` to analyze your audio and video streams

Example

```
$ packager input=~/llama_h264_main_480p_1000.mp4 --dump_stream_info
[0921/220412:INFO:demuxer.cc(58)] Initialize Demuxer for file
'llama_h264_main_480p_1000.mp4'.

File "llama_h264_main_480p_1000.mp4":
Found 1 stream(s).
Stream [0] type: Video
 codec_string: avc1.4d401f
 time_scale: 12288
 duration: 1843712 (150.0 seconds)
 is_encrypted: false
 codec: H264
 width: 858
 height: 482
 pixel_aspect_ratio: 3856:3861
 trick_play_rate: 0
 nalu_length_size: 4

Packaging completed successfully.
```

Understanding the output:

Found <#> stream(s)	Number of streams found in the content file For DASH compliance, you'll want this value to be 1.
Stream [0] type:	Specifies the stream number and if it is video or audio
codec_string:	Indicates video codec and profile. This will match the MPD data.
time_scale:	The number of time units that pass per second in its time coordinate system. A time coordinate system that measures time in sixtieths of a second. See ISO-BMFF spec for additional details.
duration	Expressed in seconds

language:	language of the video
is_encrypted:	Determines if content is encrypted
codec:	video codec
width:	Video width
height:	Video size
nalu_length_size_:	Network Abstraction Layer Units - typically expressed in 1, 2, or 4 bytes

Encrypt content

Shaka Packager allows for single and multi-file encryption. It's optimal to package all your content in a single command - encrypt all tracks and generate the MPD.

To encrypt using the Widevine Cloud License Service, you will need

a. A Content ID - a value that identifies the media that is being packaged.

To generate a random content ID (in hex)

```
$ echo -n <random alphanumeric string> | xxd -p
```

b. The proper credentials to access the License Service.

The `widevine_test` credentials in the Widevine License Service Test environment is available for testing purposes.

```
Provider = widevine_test
IV = d58ce954203b7c9a9a9d467f59839249
KEY = 1ae8ccd0e7985cc0b6203a55855a1034afc252980e970ca90e5202689f947ab9
```

Syntax for H264 - VOD

```
$ packager \
input=/root/llama_audio_aac_128k.mp4,stream=audio,output=enc_llama_audio.mp4
\
input=/root/llama_h264_baseline_360p_600.mp4,stream=video,output=enc_llama_h
264_360p.mp4 \
input=/root/llama_h264_main_480p_1000.mp4,stream=video,output=enc_llama_h264
_480p.mp4 \
input=/root/llama_h264_main_720p_3000.mp4,stream=video,output=enc_llama_h264
```

```
_720p.mp4 \
input=/root/llama_h264_high_1080p_6000.mp4,stream=video,output=enc_llama_h26
4_1080p.mp4 \
--enable_widevine_encryption \
--key_server_url
"https://license.uat.widevine.com/cenc/getcontentkey/widevine_test" \
--content_id "<hex output>" \
--signer "widevine_test" \
--aes_signing_key
"1ae8ccd0e7985cc0b6203a55855a1034afc252980e970ca90e5202689f947ab9" \
--aes_signing_iv "d58ce954203b7c9a9a9d467f59839249" \
--crypto_period_duration 0 \
--mpd_output llama_h264.mpd
```

Syntax for VP9 - VOD

```
$ packager \
input=/root/llama_audio_aac_128k.mp4,stream=audio,output=enc_llama_audio.mp4
\
input=/root/llama_vp9_360p_300.webm,stream=video,output=enc_llama_vp9_360p.w
ebm \
input=/root/llama_vp9_480p_500.webm,stream=video,output=enc_llama_vp9_480p.w
ebm \
input=/root/llama_vp9_720p_1500.webm,stream=video,output=enc_llama_vp9_720p.
webm \
input=/root/llama_vp9_1080p_3000.webm,stream=video,output=enc_llama_vp9_1080
p.webm \
--enable_widevine_encryption \
--key_server_url
"https://license.uat.widevine.com/cenc/getcontentkey/widevine_test" \
--content_id "<hex output>" \
--signer "widevine_test" \
--aes_signing_key
"1ae8ccd0e7985cc0b6203a55855a1034afc252980e970ca90e5202689f947ab9" \
--aes_signing_iv "d58ce954203b7c9a9a9d467f59839249" \
--crypto_period_duration 0 \
--mpd_output llama_vp9.mpd
```

Sample MPD

Below is a sample MPD for Shaka Packager encrypted content.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Generated with https://github.com/google/edash-packager version
```

```
e0e0925-release-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 DASH-MPD.xsd"
xmlns:cenc="urn:mpeg:cenc:2013" minBufferTime="PT2S" type="static"
profiles="urn:mpeg:dash:profile:isoff-on-demand:2011"
mediaPresentationDuration="PT734.166687011718 8S">
  <Period id="0">
    <AdaptationSet id="0" contentType="audio" lang="en">
      <Representation id="0" bandwidth="132252" codecs="mp4a.40.2"
mimeType="audio/mp4" audioSamplingRate="44100">
        <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011"
value="2"/>
        <ContentProtection value="cenc"
schemeIdUri="urn:mpeg:dash:mp4protection:2011"
cenc:default_KID="815f3b3b-c165-5415-886b-e6b837f7663f"/>
        <ContentProtection
schemeIdUri="urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">

<cenc:pssh>AAAAaHBzc2gAAAAA7e+LqXnWSs6jyCfc1R0h7QAAAEgIARIQJPZ2phZEUvO+FnPYDj
XwBIQW2j5Y6rUXreKuHqFkSvFSxIQgV87O8FlVBWIa+a4N/dmPxoFY3dpcDEiBRI0VmeIMgA=</ce
c:pssh>
        </ContentProtection>
        <BaseURL>enc_tears_audio.mp4</BaseURL>
        <SegmentBase indexRange="1001-1920" timescale="44100">
          <Initialization range="0-1000"/>
        </SegmentBase>
      </Representation>
    </AdaptationSet>
    <AdaptationSet id="1" contentType="video" width="720" height="300"
frameRate="12288/512" par="12:5">
      <Representation id="1" bandwidth="686572" codecs="avc1.42c01e"
mimeType="video/mp4" sar="1:1">
        <ContentProtection value="cenc"
schemeIdUri="urn:mpeg:dash:mp4protection:2011"
cenc:default_KID="24f676a6-1644-52f3-be16-73d80e3517c0"/>
        <ContentProtection
schemeIdUri="urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">

<cenc:pssh>AAAAaHBzc2gAAAAA7e+LqXnWSs6jyCfc1R0h7QAAAEgIARIQJPZ2phZEUvO+FnPYDj
XwBIQW2j5Y6rUXreKuHqFkSvFSxIQgV87O8FlVBWIa+a4N/dmPxoFY3dpcDEiBRI0VmeIMgA=</ce
c:pssh>
        </ContentProtection>
        <BaseURL>enc_tears_video.mp4</BaseURL>
        <SegmentBase indexRange="1127-1902" timescale="12288">
```

```
           <Initialization range="0-1126"/>
         </SegmentBase>
       </Representation>
     </AdaptationSet>
   </Period>
</MPD>
```

Content Playback

Once the encrypted media is staged on a web-accessible URL, it is available for playback. Widevine's open-source reference Shaka Player is recommended to test and validate playback.

The Shaka Player public demo is configured to automatically enable playback of encrypted content (using the `widevine_test` credentials).