WIDEVINE®

Widevine DRM
Architecture Overview

Table of Contents

Contact Widevine

For more information about Widevine, contact us at http://www.widevine.com/contact.html.

Related Documentation

For content provider partners, please look at the Getting Started guide.
For device partners, please look at the Getting Started for Devices guide.

Widevine DRM Architecture Overview

Google's Widevine DRM architecture provides a market-leading platform for delivering protected premium content at the highest possible quality to the largest number of devices. The Widevine DRM platform uses standards-based royalty-free solutions for encryption, adaptive streaming, transport, and player software. It also includes free, open source tools for content preparation and media playback, enabling openness and innovation at every level.

Widevine DRM gives content partners easy, effective, and inexpensive methods for streaming video over the Internet, accelerating the transition away from proprietary and legacy systems to give billions of video consumers better access to the next generation of media experiences.

This overview will introduce the nine core components of the Widevine DRM architecture and explain how the components work together to create a secure playback system that starts with the Shaka Packager and finishes with a choice of players on multiple client platforms and devices, including HTML5, Android, and iOS.

Architecture Components List

The Widevine DRM architecture consists of the following nine core components:

Name	**Description**
Common Encryption	W3C Standard encryption protocol
Encrypted Media Extensions	W3C Standard for encrypted playback
Media Source Extensions	W3C Standard for media streaming
Dynamic Adaptive Streaming over HTTP	W3C Standard streaming protocol
Shaka Packager	Open source content packaging
Widevine License Server	Provides WV license information
Video Players (HTML5, Android, iOS, OEM)	Secure playback on various platforms
Content Decryption Module	Device-specific decryption
OEMCrypto Module	Trusted Hardware decryption

Architecture Component Relationships

The Widevine DRM components work together to create an end-to-end platform for protecting premium video content, and provides all the tools you need to go from the content preparation stage to the final delivery on any device. The flow process begins by preparing your media with

Common Encryption and the Shaka Packager for adaptive streaming. Once prepared, the content is encrypted with licenses which are stored on the Widevine License Server. Later, when the encrypted content streams to the Player via a Content Delivery Network, the License Server provides license information to a supported media player. The encrypted content is then passed to the device's Content Decryption Module, enabling secure playback with the OEMCrypto Module. Diagram 1-1 shows how the Widevine DRM components flow together as a platform.

Components Overview

This section introduces the nine core components and explains the role they play in the Widevine DRM platform.

Common Encryption

The Widevine DRM uses the Common Encryption (CENC) open standard for encryption and also recommends the ISO Base Media File Format (BMFF) standard with CENC, but note that the WEBM format is also supported and that other formats may be supported in the future. You can read more about CENC and BMFF on the W3C site at w3c.github.io/encrypted-media/cenc-format.html. Note that the Widevine DRM CENC (Common Encryption), Generic CENC, Generic FairPlay Streaming encryption formats are currently supported.

Encrypted Media Extensions

The Encrypted Media Extensions (EME) use the Common Encryption format as part of the Widevine DRM end-to-end security protocol, making sure no content escapes into the wild. The Encrypted Media extension to the HTML5 Media element specification can be found at http://www.w3.org/TR/encrypted-media/. There is a good overview of EME at http://www.html5rocks.com/en/tutorials/eme/basics/.

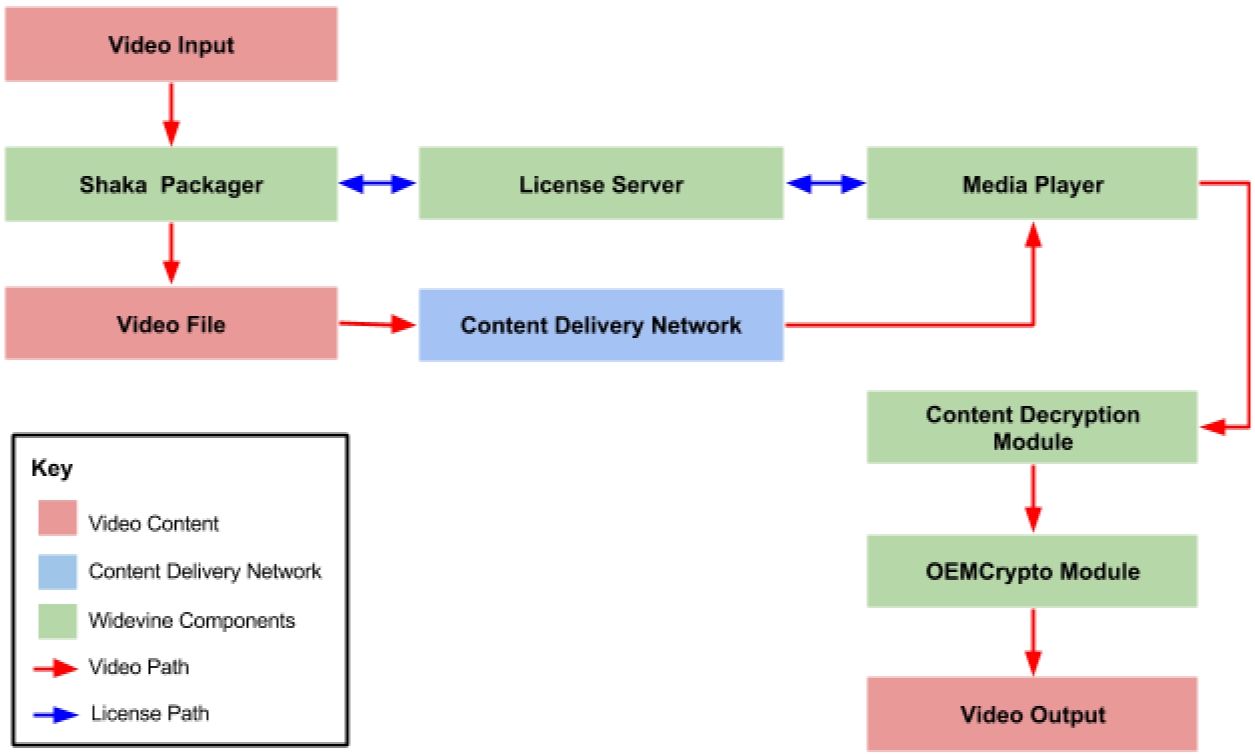

Diagram 1-1: Widevine DRM Component Flow

Media Source Extensions

Media Source Extensions (MSE) are used to parse the incoming DASH-based media streams and pass them to the playback hardware. The Media Source extension to the HTML5 Media element specification can be found at http://www.w3.org/TR/media-source/.

Dynamic Adaptive Streaming over HTTP

In order to provide a solution to the common problem of varying bandwidth in an unknown environment, Widevine DRM uses the open standard, Dynamic Adaptive Streaming over HTTP (DASH). You can learn more about DASH at http://dashif.org/.

The DASH protocol is used because it solves two important content delivery problems: *bandwidth* and *transport*.

Bandwidth: Dynamic Adaptive Streaming

In an ideal streaming scenario, video is sent to the viewer at the highest rate and quality. However, in a typical urban neighborhood, the delivery bandwidth may change over time for a variety of reasons. For example, at 8:00PM in an apartment complex, when everyone starts watching movies at the same time, the bandwidth decreases and the stream needs to play at a lower bitrate to preserve continuity.

DASH solves this problem by taking video encoded at different resolution and bitrates, converting it into fragmented MP4 files, which are divided up into segments of the same length. When the device receives encrypted video, it checks the bandwidth and requests the appropriate segment of the fragmented MP4 file from the server. The bandwidth is checked again on a repeating basis and a higher or lower quality segment is chosen to play next if the average bandwidth has changed significantly. Because the quality of the stream is dynamically adapted when the circumstances change, the viewer has an optimum playback experience. Note that a "fragment" in fragmented MP4 may have more than one segment.

Diagram 2-1 shows how dynamic adaptive streaming works. The highlighted areas indicate which alternate chunk (segment) will play at a specific time, determined by current bandwidth. These are only sample numbers, and the numbers and times will vary depending on bandwidth and quality desired.

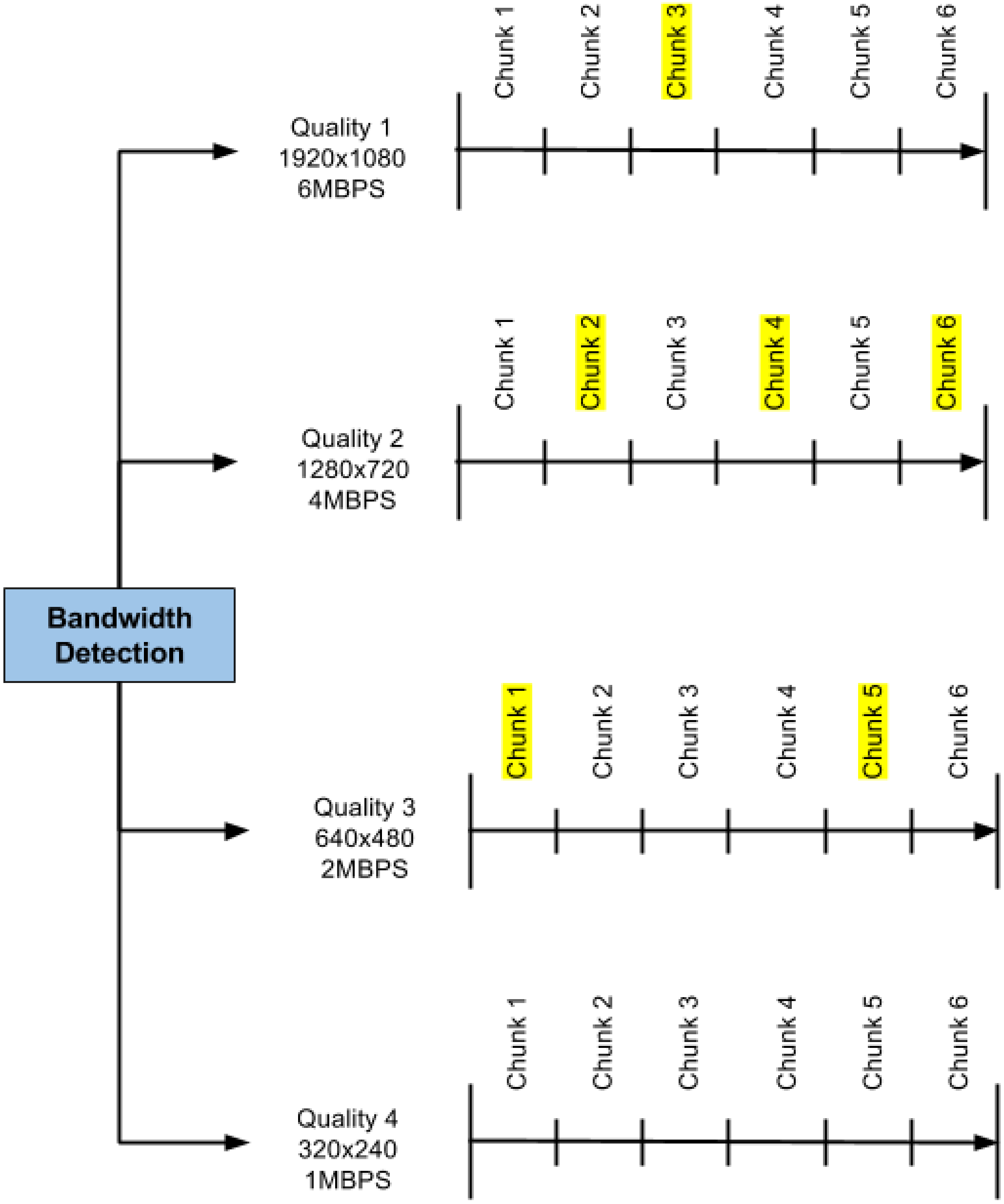

Diagram 1-2: Dynamic Adaptive Streaming

Transport: Streaming over HTTP

Widevine DRM uses the open standard HTTP protocol because it is included in all browsers and browsers are now part of most devices.

HLS Streaming

Widevine also supports the Apple HLS Live adaptive streaming format.

Media Packaging

Widevine DRM provides a complete open-source DASH packaging system called Shaka Packager. The packager converts files of different resolutions and bandwidth to fragmented MP4, defining equal-length segments for all desired files. A manifest (MPD) is prepared that describes the different resolutions and bandwidth for each file. When a player receives encrypted content, it requests the appropriate segment that will provide optimal viewing at that moment, but can request different segments as bandwidth conditions change. The Shaka Packager is covered in more detail in the Shaka Content Packager section.

License Server

In order to provide license information for encrypting and decrypting media securely, Widevine DRM provides a Cloud-based license service. When media is prepared, information about the media is sent to the Widevine License Server for later use. Then when the stream is received by the Player, the License Server is called and license information is provided. The license protocol used to communicate with the License Server is a simple request-response over HTTPS. The License Request and License Response messages are constructed and parsed using Google Protocol Buffers.

Video Players

Widevine DRM supports player technologies for HTML5, Android, and iOS. In addition, it also supports individual OEM devices on a licensed request basis. Specific details for developing with HTML5, Android, and iOS player technologies are covered in later sections.

Content Decryption Module

Widevine installs a Content Decryption Module (CDM) on every device that plays back encrypted content. The module is unique for each type of device and has three major functions:

1. When the Player determines that the content is encrypted, it tells the CDM to generate a license request. The MPD and PSSH (Protection System Specific Header) information is retrieved from the content and parsed in order to determine the DRM system to be used.
2. The CDM then creates an encrypted license request object which it passes back to the Player. The Player will then pass the encrypted license object to the License Server.
3. Next, when the License Server responds to the Player request, it sends an encrypted object containing license information to the Player. Finally, the Player passes the encrypted object to the CDM, which in turn passes it to the OEMCrypto Module for decryption.

OEMCrypto Module

The OEMCrypto Module decrypts the content using information passed to it from the Player (and the License Server). The OEMCrypto Module is in the Trusted Layer of the device and is tied into the device hardware. It uses the encrypted license information to decrypt the media, and the media sent to the video stack.

Widevine DRM Security Model

The DRM security model provides secure decryption using a sequence of exchanges between the Widevine License Server and the Content Decryption Module. The Player acts as a facilitator for these exchanges, but cannot read the encrypted license information or content. Only the License Server and the Content Decryption Module can work directly with the license information, but neither are open source. This unique mix of open source and protected source enables Widevine DRM to make it easy to create custom playback applications that are encrypted and secure.

The following eight steps explain the sequence of security exchanges for decryption:

1. Receive Media from Content Delivery Network (CDN)

The browser's media engine determines that the media is encrypted. The initialization data (**InitData**) is extracted from the content by the browser and sent to the Player as an event. User authorization is up to the client, not Widevine.

2. Pass Data to the Content Decryption Module (CDM)

The Player passes the **initData** to the Content Decryption Module. The Player is not able to decrypt anything.

3. CDM Passes License Request to the Player

Having received data from the Player, the CDM creates a license request and passes the request information back to the Player.

4. Player Passes License Request to License Server (via proxy)

Once the Player has the License Request, it can now pass that request to the Widevine License Server (via proxy). But the Player cannot read the request itself or take any action except to pass it on.

5. License Server Passes License to Player

When the License Server receives the request, it responds to the request by sending the license back to the Player. The license is an encrypted message that includes license information data.

6. The Player Passes the License to the Content Decryption Module

The Player passes the encrypted license to the Content Decryption Module.

7. The CDM Passes Data to the OEMCrypto Module

Because the CDM is not in the Trusted Layer of the device, it must pass the information to the OEMCrypto Module, which does reside in the Trusted layer of the device. The actual decryption takes place in the OEMCrypto Module. In some implementations, the decoding takes place there also. The Browser does the actual parsing of the container.

8. OEMCrypto Sends Video Chunks to the Screen

Once the content is demultiplexed, decrypted, and decoded, it is sent to the screen in small chunks, and is not stored anywhere on the device.

Diagram 1-3 shows the playback security model's sequence of exchanges between the playback components.

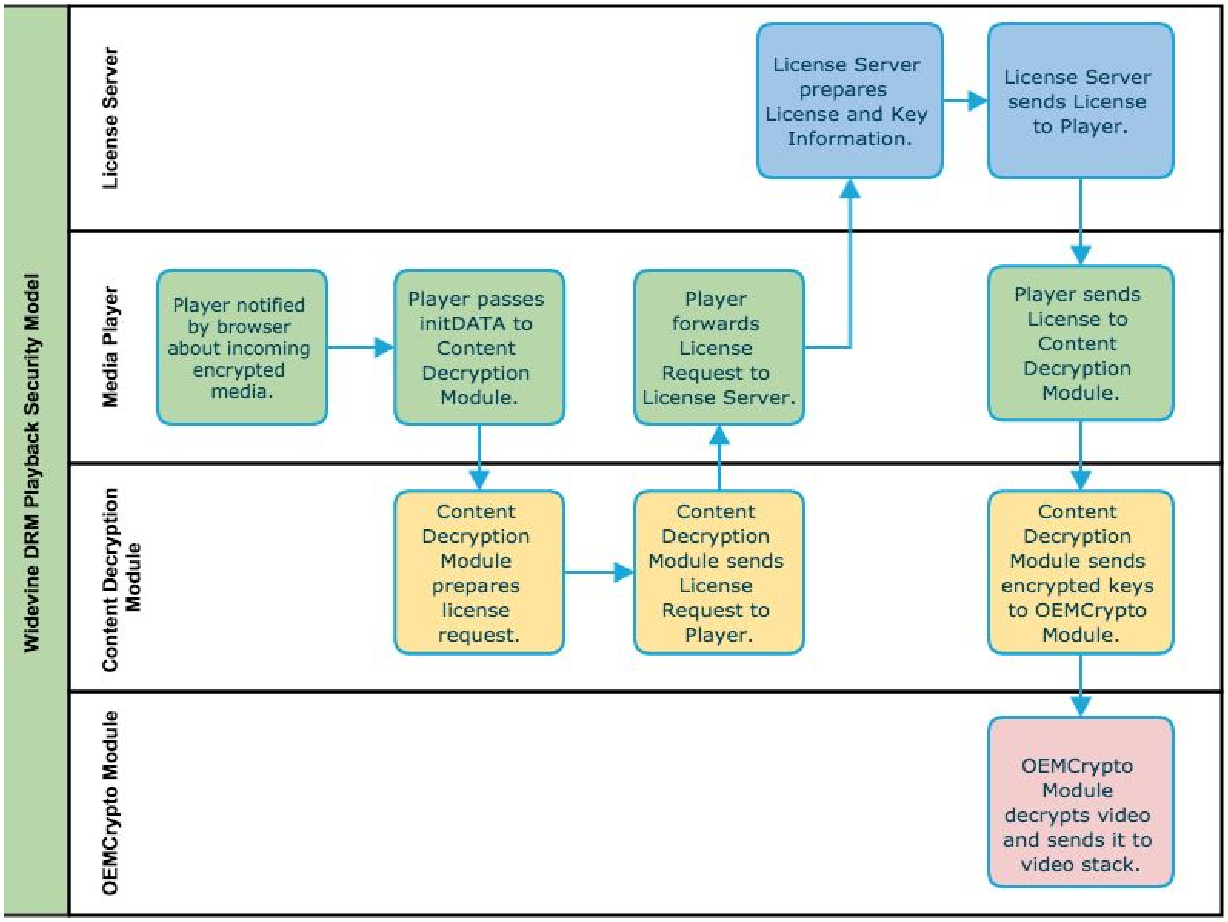
Widevine DRM Playback Security Model
License Server
Media Player
Content Decryption Module
OEMCrypto Module
Player notified by browser about incoming encrypted media.
Player passes initDATA to Content Decryption Module.
Content Decryption Module prepares license request.
Content Decryption Module sends License Request to Player.
Player forwards License Request to License Server.
License Server prepares License and Key Information.
License Server sends License to Player.
Player sends License to Content Decryption Module.
Content Decryption Module sends encrypted keys to OEMCrypto Module.
OEMCrypto Module decrypts video and sends it to video stack.

Diagram 1-3: Widevine DRM Playback Security Model

Security Levels Provided by Widevine DRM

Widevine devices provide different security levels, depending on the device used.

Security Level Definitions.

The following security level definitions are used by Widevine:

Security Level 1

All content processing, cryptography, and control is performed within the Trusted Execution Environment (TEE). In some implementation models, security processing may be performed in different chips.

Security Level 2

Performs cryptography (but not video processing) within the TEE: decrypted buffers are returned to the application domain and processed through separate video hardware or software. At level 2, however, cryptographic information is still processed only within the trusted execution environment.

Security Level 3

Does not have a TEE on the device. Appropriate measures may be taken to protect the cryptographic information and decrypted content on host operating system. A Level 3 implementation may also include a hardware cryptographic engine, but that only enhances performance, not security.

Security Level for Widevine Devices

The following table describes the security for Widevine devices:

Device Type	**Security Level**
Chrome on desktops	L3
ChromeOS	L1, L3
Android	L1, L3
Embedded devices	Varies depending on implementation

Shaka Packager

Widevine DRM provides the Shaka Packager so that you can take content and transmux the content into segments suitable for adaptive streaming using DASH and HLS. In addition, Shaka Packager encrypts the content using the Widevine DRM CENC (Common Encryption), Generic CENC, Generic FairPlay Streaming encryption formats. The packager also prepares a DASH manifest (MPD) that identifies media streams and their respective URLs.

Packaging Steps

These are the steps for packaging your content for DASH streaming:

1. Encode the video using the VP9 or H.264 codec. MPEG2-TS and Widevine 1.0 (WVM) are also supported and other encoding formats may be supported in the future. You can use any encoding tools you want as long as they follow the standards for the media you are encoding
2. Separate files must be encoded for each resolution and bitrate that you want your player to support. For example, 1920x1080 at 6MBPS might be a high-quality choice and a low-quality would be 320x240 at 1MBPS.
3. The following three steps are then performed concurrently by Shaka Packager:

    a. It converts each encoded file to a fragmented MP4 or WebM container format with defined segments. Each segment will be the same time-coded length. Shaka Packager will produce a fragmented MP4 file for each specific resolution and bitrate file you feed it.
    b. It encrypts each file with CENC using license information obtained from the Widevine License Server.
    c. It creates a manifest (MPD) describing each fragmented MP4 file and all the details necessary to play it back. Additionally the system allows for the injection of a CENC PSSH (Protection System Specific Header) to identify the DRM system used.

Once Shaka Packager is finished, copy the different bitrate/resolution files and the manifest to a folder on your Content Delivery Network. The URL pointing to the MPD will be used to begin the streaming.

Diagram 2-1 shows the process flow for using Shaka Packager.

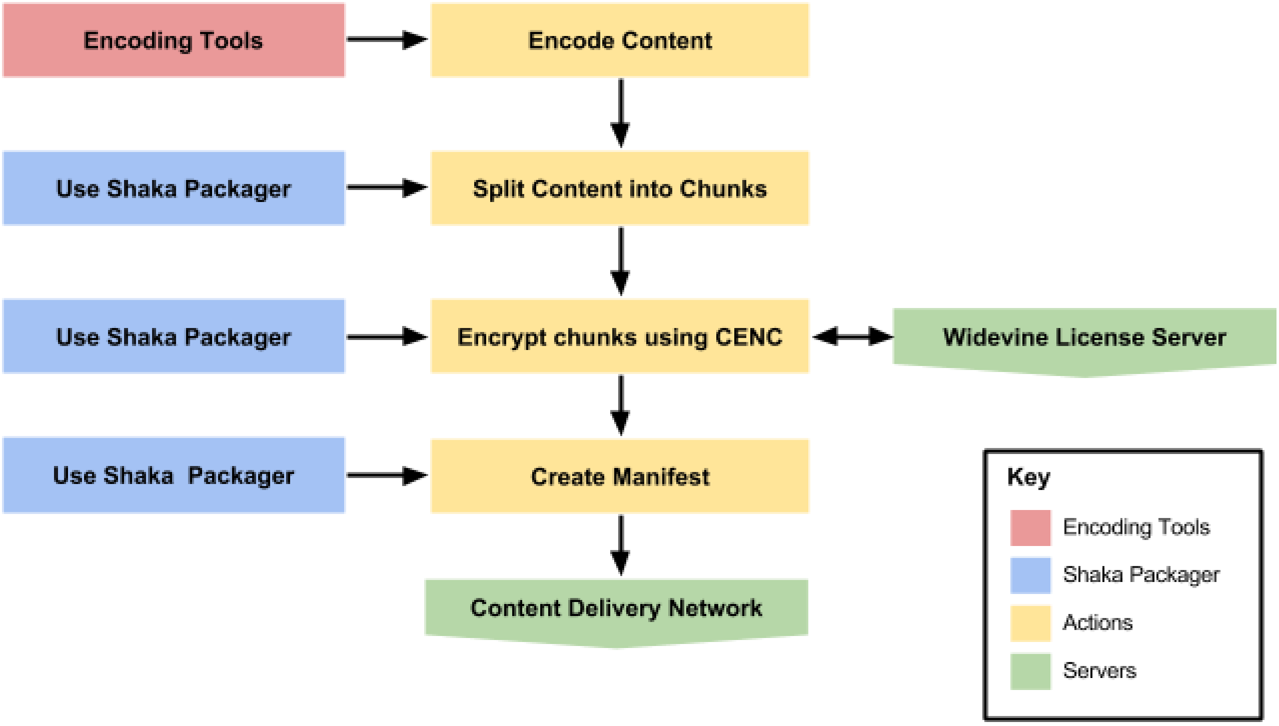

Diagram 2-1: Shaka Packager Content Flow

Working with Shaka Packager

The following resources will help you work with Shaka Packager more easily and efficiently.

Downloading Shaka Packager

You can download the Shaka packager source code at https://GitHub.com/google/shaka-packager.

Building Shaka Packager

Use Git or Subversion to download the source source code. The build tools are designed for Linux, but you can use Docker to build on an Apple Mac. If you want to build the packager on Windows, you can use Cygwin from https://www.cygwin.com. Follow these steps to build the package:

1. Install the Chromium Depot tools. The Chromium Depot tools can be found at https://www.chromium.org/developers/how-tos/get-the-code.
2. Get the Shaka Packager source code from GitHub.
3. Build the Shaka Packager using the Ninja build tool. Ninja is a utility similar to the make build tool, but is faster. Ninja are included in the Chromium depot tools.

Once you have built Shaka Packager, you can use it to multiplex and encrypt media for transmission to the Content Delivery Network of your choice.

Shaka Packager Community

You can participate in Shaka Packager community activity at https://GitHub.com/google/shaka-packager/issues.

ISO BMFF

Shaka Packager supports common media containers: WebM, ISO BMFF, MPEG-2 TS, and Widevine Media 1.0 (WVM). You can learn more about ISO BMFF at http://www.w3.org/2013/12/byte-stream-format-registry/isobmff-byte-stream-format.html. The current version supported is *ISO/IEC 23001-7 Common Encryption in ISO base media file format files - version 2*.

The video must be encoded in VP9 or H.264 format but other formats may be supported at a later time. You are free to use any tools to encode video in those formats, but the Shaka Packager must be used to prepare files for DASH or HLS streaming.

Shaka HTML5 Video Player

Widevine DRM provides the Shaka Player for platforms supporting HTML5. Shaka Player makes it possible to have secure playback of streaming video over the Internet. It does this by first receiving a DASH (Dynamic Adaptive Streaming over HTTP) multiplexed stream, determining the current bandwidth of the user, and selecting the appropriate segment for optimal playback. Segments will be the same length but may have different resolutions and bitrates. Next Shaka Player prepares the chosen segment for decrypting and decoding by the device Content Decryption Module and underlying video hardware. At this point, the video is ready to play.

DASH, CENC, EME, MSE Standards

The DASH and CENC standards were discussed earlier, but Shaka Player also uses two other open standards: Encrypted Media (EME) and Media Source (MSE) extensions to the HTML5 Media Element.

Player Components

The playback of secure video content through Shaka Player involves these five major components:

- Widevine License Server
- Shaka Player Library
- Encrypted Media Extensions
- Media Source Extensions

- Content Decryption Module

Widevine License Server

The Widevine License Server has been covered earlier and serves license information to Shaka Player.

Shaka Player Library

Shaka Player Library is a free and open-source library that facilitates secure playback of encrypted media in an HTML5 environment. The Shaka Player library is written in JavaScript and compiled using the Closure library for greater efficiency, quality, and security. The core of Shaka Player uses the twin HTML5 media standards of Encrypted Media Extensions (EME) and Media Source Extensions (MSE) to ensure that the streams are demultiplexed, decrypted, and decoded securely.

Encrypted Media Extensions

Shaka Player uses the open source Encrypted Media Extensions "under the hood" to implement content security. You don't need to do any additional programming to make it work, but because Shaka Player is open-source, you can always modify the details. Encrypted Media Extensions is covered earlier in this document.

Media Source Extensions

Shaka Player uses the open source Media Source Extensions to enable DASH streaming, but does not require additional programming since it is embedded in the Shaka Player library. Media Source Extensions is covered earlier in this document.

Content Decryption Module

The Content Decryption Module is a vital part of the Widevine DRM stack and has been covered earlier. Chrome installs this module to decrypt the media in a trusted manner.

Component Stack for Shaka Player

The Diagram 3-1 shows Shaka Player components and how they relate to each other in a stack to enable media demultiplexing, decrypting, and decoding. The flow starts with Shaka Player receiving content from the Content Delivery Network. The Shaka Player library works with the License Server to pass encrypted license information to the Content Decryption Module. Media Source Extensions and Encrypted Media Extensions work alongside of DASH and CENC to enable the Content Decryption Module to play secure content.

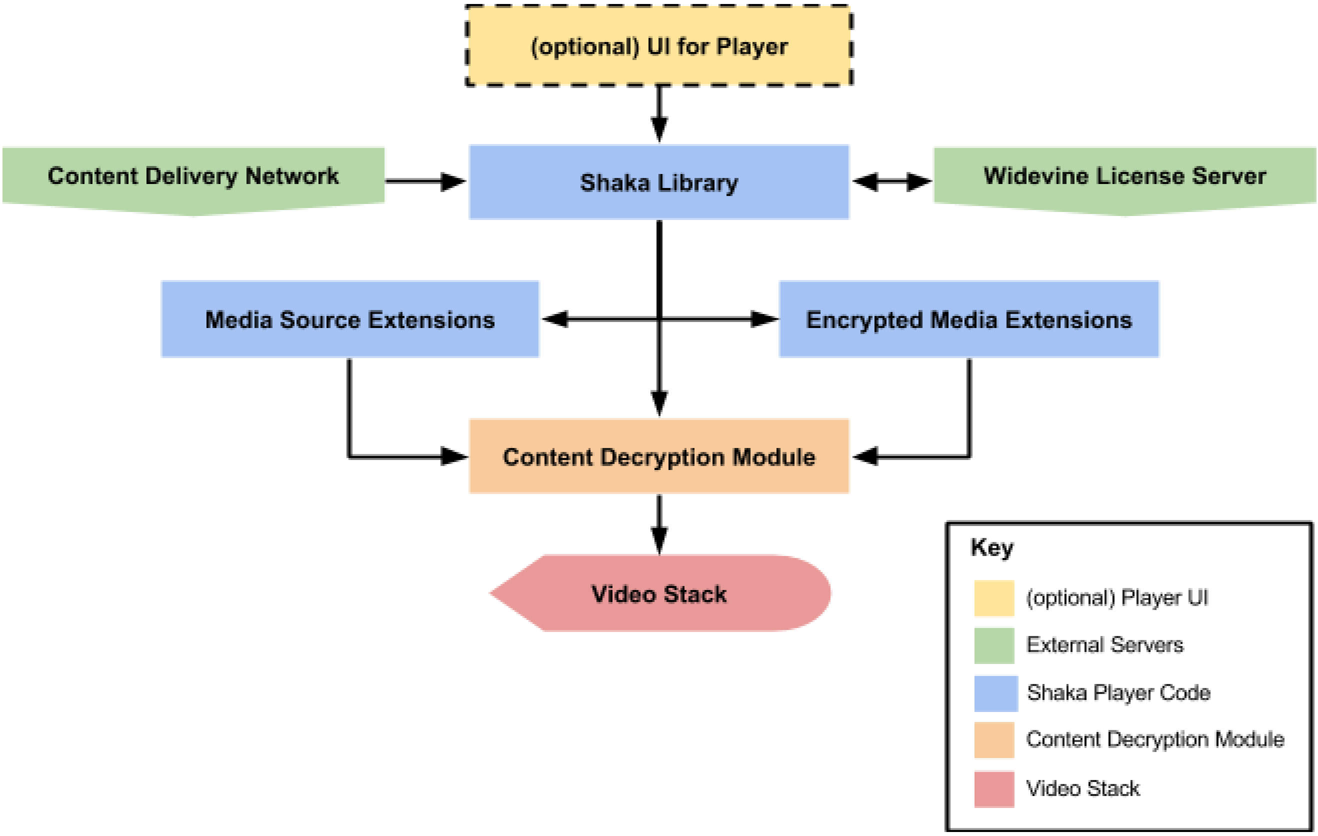

Diagram 3-1: Shaka Player Component Stack

Chrome Components Needed by Shaka Player

When Chrome is installed on a device, it also installs an additional Widevine component: the Content Decryption Module (CDM). Shaka Player requires this component for decryption. Because the CDM is the heart of the decryption process, it is not open-source but uses the protocols of the Common Encryption standard. If this component needs to be updated, Chrome will update it automatically.

App Development with Shaka Player

Shaka provides a simple JavaScript API that only requires a small amount of code to configure the player for your specific needs. All you need to do is input the following details:

1. Initialize the player.
2. Define the video source and manifest.
3. Configure the DRM type.

Diagram 3-2 shows the Shaka API programming requirements:

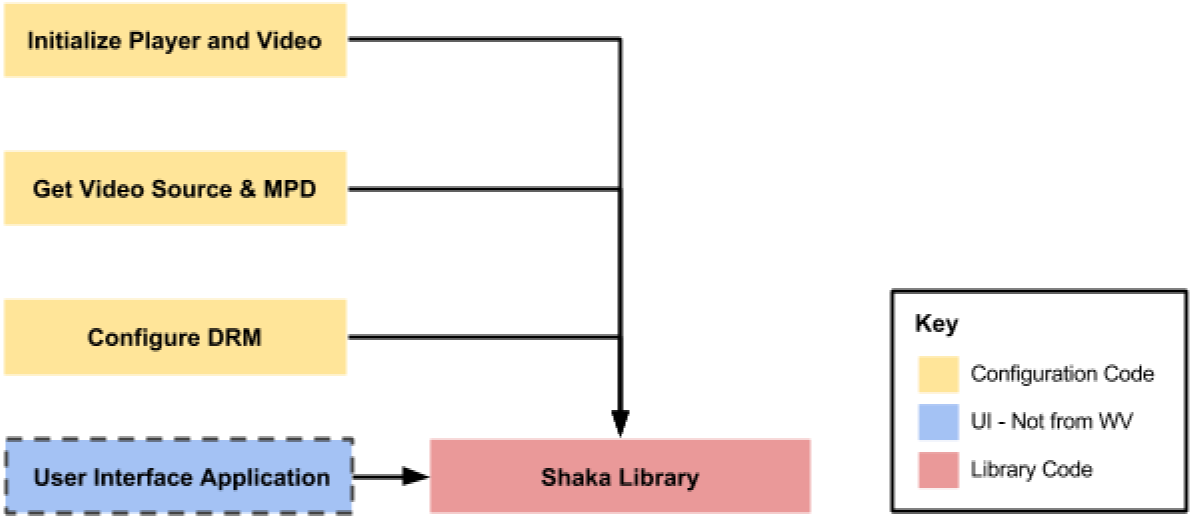

Diagram 3-2: Programming the Shaka API

Working with Shaka Player

The following requirements and resources will help you to work with the player more easily and efficiently.

Downloading Shaka

You can download the latest version of Shaka at https://GitHub.com/google/shaka-player. More information about Shaka Player is available from g.co/shakainfo

Shaka Requirements

Shaka requires an HTML5-compliant browser (i.e. Chrome 33 or higher) that supports EME and MSE.

Shaka Community

There is an online community of participants in the Shaka open source GitHub project at https://GitHub.com/google/shaka-player/issues. At this site you can meet with other users and Google developers who can answer questions, make suggestions, and discuss programming options. There is also a mailing list you can join for further discussion of Shaka issues at https://groups.google.com/forum/#!forum/shaka-player-users.

Building Shaka

In order to get the most out of Shaka Player, you need to build it from source code provided by Widevine on GitHub. This code is updated frequently, ensuring that you can get the latest build. The build uses the Google Closure compiler on the Shaka JavaScript source code to minify, optimize, obfuscate, and check for many different types of errors. You can run Shaka in the uncompiled state but this is not recommended for production environments because the compiled version is faster, smaller, and more robust. Instructions for downloading and building

Shaka Player on Windows, Mac, and Linux are at http://shaka-player-demo.appspot.com/docs/tutorial-dev.html.

Pre-compiled Builds

Pre-compiled release builds of Shaka Player are available from CDNJS.com for your convenience. For more information, visit https://cdnjs.com/libraries/shaka-player.

Test Application

Shaka provides a test app you can use for functional testing. You can tune this app by adjusting specific URL parameters if needed. By using this app, you can determine that your own media application running on top of Shaka is working correctly. For more information, see http://shaka-player-demo.appspot.com/docs/tutorial-dev.html.

Android Player for Widevine DRM

You can develop a native Android media player application using the Java Platform APIs or a web application using HTML5 and JavaScript APIs. A native Android application must use low-level Android API methods. DASH and CENC are open standards that the Android player supports.

Components for Android in Widevine DRM

The components for Android follow the general pattern for Widevine DRM player applications. The main difference is that you must use low-level Android API methods and you have the option of using an open-source project, ExoPlayer, as a sample application, library, or reference player for A/B testing.

Diagram 4-1 shows the components for Android in Widevine DRM and how they relate to each other.

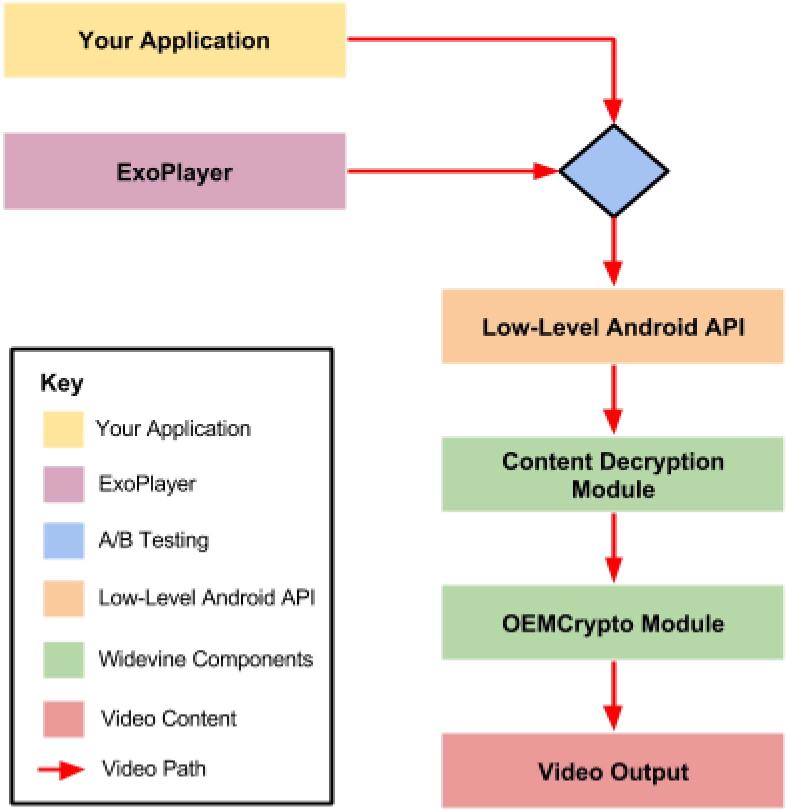

Diagram 4-1: Android Player Component Stack

Native Android Application

Android has Widevine technology built into the Android libraries. Android media players are normally created using the **MediaPlayer** interface at http://developer.android.com/reference/android/media/MediaPlayer.html. However, **MediaPlayer** does not support DASH at a high-level. Instead, you will want to use the following Android interfaces to provide DASH support:

Interface	**Android SDK Location**
MediaExtractor	http://developer.android.com/reference/android/media/MediaExtractor.html
MediaCodec	http://developer.android.com/reference/android/media/MediaCodec.html
MediaCrypto	http://developer.android.com/reference/android/media/MediaCrypto.html
MediaDrm	https://developer.android.com/reference/android/media/MediaDrm.html

For an example of how to use these interfaces to play DASH video, see the ExoPlayer open-source project at http://google.GitHub.io/ExoPlayer.

ExoPlayer

The ExoPlayer project can be useful when creating native applications in the following ways:

- Reference Player
- Source Code Study
- Library Wrapper

Reference Player

The ExoPlayer can be used as an A/B reference to test your native player application. You will be able to see if your application is functional by switching back and forth between your player and ExoPlayer.

Source Code

You can study the ExoPlayer source code to see how the Android libraries are called.

Library Wrapper

The ExoPlayer exports methods that you can call to simplify your Android development. The methods provide a wrapper for complicated method calls, saving you time and trouble.

HTML5 Chrome in Android

Android 5+ includes a version of Chrome that supports HTML5. You can implement an HTML5 DASH and CENC video application using Encrypted Media Extensions (EME) and Media Source Extensions (MSE). Shaka Player supports Chrome on Android. See Shaka Player section for more information on HTML5 playback.

Supported Android Versions

The Widevine CDM and OEMCrypto modules are supported in Android 4.4 (API Level 16) or higher. ExoPlayer is also supported for those versions.

Widevine DRM SDK for iOS

Apple iOS does not natively support Dynamic Adaptive Streaming over HTTP (DASH) or Common Encryption (CENC). For this reason, Widevine DRM has created an SDK for iOS developers who want to stream video using DASH with CENC. Since iOS uses the HTTP Live Streaming (HLS) protocol instead of DASH, Widevine DRM transmuxes DASH to HLS on-the-fly, while keeping the content protected. Widevine DRM provides the CDM Dynamic Library to facilitate the transmuxing process.

CDM Dynamic Library

Widevine provides the CDM Dynamic Library (CDL) to incorporate Widevine video into your own iOS application. CDL contains components that use DASH and CENC to make it possible to view video content securely on an iOS device. The following components make up the dynamic library:

- Universal DASH Transmuxer
- Content Decryption Module
- OEMCrypto API

Universal DASH Transmuxer

The Universal DASH Transmuxer (UDT) enables DASH content to be converted to HLS streams on-the-fly without any perceptible loss of playback fidelity. The UDT workflow contains the following stages:

- Parse the Manifest
- Create an HLS Playlist
- Stream the Segments

Parse the Manifest

An XML parser is required to read the DASH manifest (MPD) and then parse each element for input for HLS playlist creation. The **TBXML** utility is recommended and used by the included reference player, but XML parsers such as the native **libxml2** or others, like **NSXMLParser** or **TouchXML,** can be used.

Create an HLS Playlist

The input to create an HLS playlist will be dependent on the complexity of the incoming parsed manifest (MPD). Once parsed, the parameters will be passed to the UDT and a separate **.m3u8** playlist will be generated for each audio and video track. For example, an MPD containing 4 separate bitrates of audio and video would result with 8 playlists.

Stream Segments

Once the HLS playlists have been created, a localhost URL must be sent to the video player to initiate the standard HTTP request. The request must be intercepted by a locally running proxy and must return the transmuxed TS segments back to the player from the UDT. The **CocoaHTTPServer** is recommended and is used by the reference player, but any localhost server can be used.

Content Decryption Module

The Content Decryption Module (CDM) implements methods and callbacks that are compliant with Common Encryption (CENC) standards. The CDM is responsible for acquiring license information from the Widevine License server and handles license exchanges in a secure manner to enable playback of encrypted content. The CDM solution also contains the following utilities for a successful operation:

- iOS Host
- Protocol Buffers
- String Encoders

iOS Host

The Host interface communicates with the upper layers of the iOS system and processes application-level events and system-level services.

Protocol Buffers

The license protocol used to communicate with the License Server is a simple request-response over HTTPS. The License Request and License Response messages are constructed and parsed using Google Protocol Buffers.

String Encoders

A collection of high performance c-string transformations used for base64 strings.

OEMCrypto API

The purpose of OEMCrypto API is to provide an additional layer of security while handling license information exchange. This interface defines a standard set of functions that are needed to securely perform various license protocol operations.

Widevine Reference Player for iOS SDK

Widevine provides a reference player for testing the Widevine components on iOS. The Reference Player is minimal and only for testing purposes. The Reference Player can be used in parallel to your application to perform A/B testing.

Diagram 5-1 shows the Widevine SDK for iOS components and their relationship to the Reference Player.

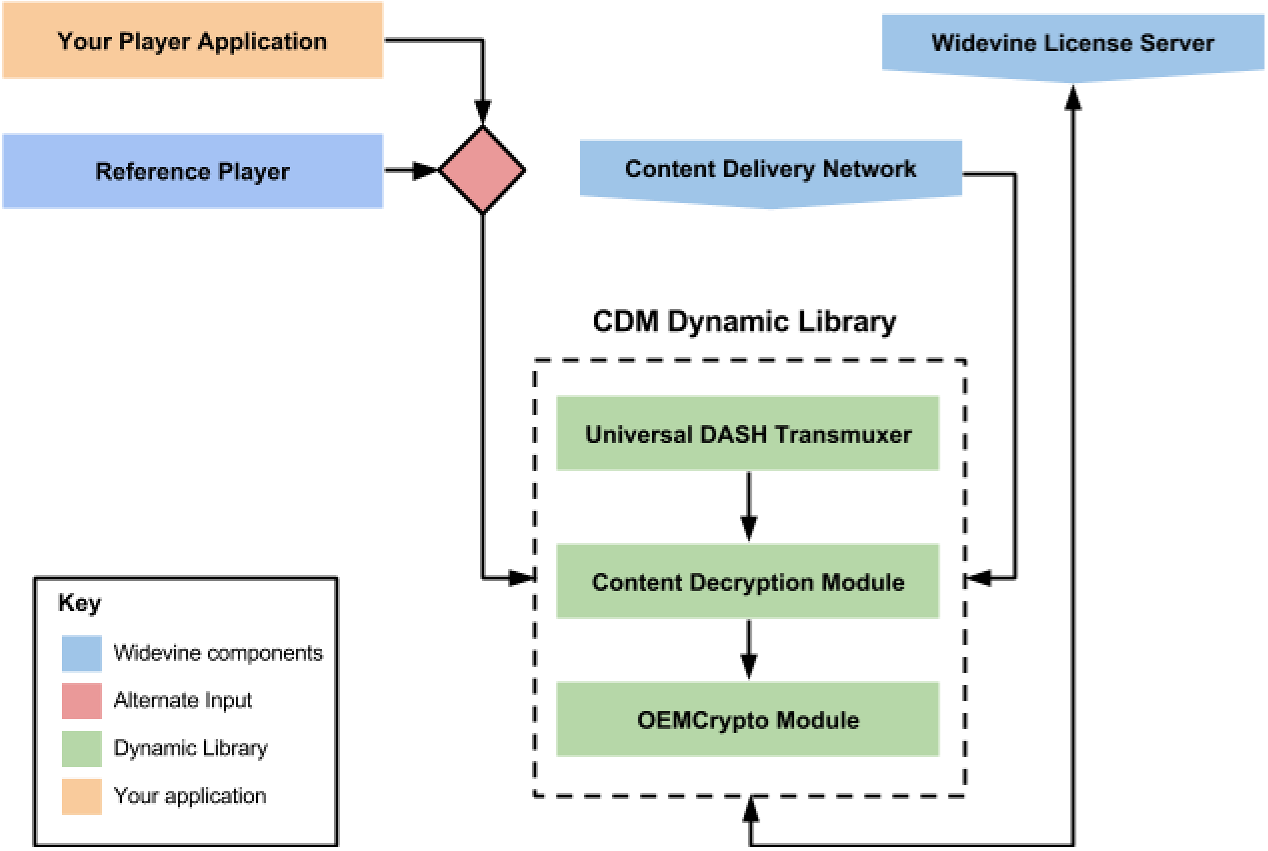

Diagram 5-1: Widevine SDK for iOS Components

Working with the Widevine SDK for iOS

The following requirements and resources will help you work with the Widevine SDK for iOS more easily and efficiently.

Widevine iOS SDK Requirements

Apple iOS versions 7 through 9 are supported with 64-bit app capability. Xcode version 6.3 or higher is required when building for iOS7 due to restrictions with embedded libraries

Widevine CDM Dynamic Library Versions

The dynamic library comes in two release versions: production and development.

Production Releases

The production release dynamic library is protected and obfuscated and does not support jailbroken devices or run with a debugger attached to the build (including XCode). Symbols are removed and you can only work with Widevine production License Servers.

Development Releases

If you are working with a development release, you must use the Widevine test License Server (license.uat.widevine.com). There are two Development builds: EIT and SIM. EIT contains obfuscation and will only run on devices; SIM will only work on simulators, not devices.

Widevine iOS Community

There is a Google Group community for discussing the Widevine iOS SDK. You can find it at widevine-ios-discuss@googlegroups.com.

Version History

Version	Date	Summary
1.0	9/11/2016	Creation
1.1	7/14/2016	Removal of MediaCloud service; change eDASH Packager to Shaka Packager; updates for codecs, packages, and streaming formats supported; minor edits.
1.2	3/6/2017	Updated Contact Widevine, Related Documentation

No express or implied warranties are provided for herein. All specifications are subject to change and any expected future products, features or functionality will be provided on an if and when available basis. Note that the descriptions of Google's patents and other intellectual property herein are intended to provide illustrative, non-exhaustive examples of some of the areas to which the patents and applications are currently believed to pertain, and is not intended for use in a legal proceeding to interpret or limit the scope or meaning of the patents or their claims, or indicate that a Google patent claim(s) is materially required to perform or implement any of the listed items.