Modular DRM
iOS CDM User Guide v1.5

iOS SDK v3.0.10

User Guide for Widevine iOS SDK

This document provides an overview of the Widevine iOS SDK and each of the required components. It contains iOS-specific supplemental information for the common document *Widevine Modular DRM Security Integration Guide for Common Encryption (CENC).*

Introduction

The Widevine iOS SDK includes the following components that create a customizable solution for the application developer interested in integrating Widevine Modular DRM into their iOS app. These components include:

- Software components
- User Guide (this document)
- An API Reference (coming soon)

Software Requirements

The following software must be installed on your development machine:

- OSX 10.10 or higher
- XCode 6.4 or higher

The following iOS targets are supported:

- iOS 8
- iOS 9
- iOS 10

Software Components

The software components are packaged in a single zip file for download. When you unzip this file, the following packages will be available:

- CDM Host Interface
- CDM Reference Player
- iOS CDM SDK Dynamic Library

In addition, the following XCode project files will be included:

- CocoaAsync Socket
- Cocoa HTTP Server

Software Component Details

The following table provides more details about the downloaded software:

Component	Details
CDM Host Interface	CDM API Interface for programming your own player
CDM Reference Player	Reference player for testing
CDM Dylib	Dynamic library (contains **Content Decryption Module** (CDM), **OEMCrypto Module** (OEMC), and **Universal Dash Transmuxer** (UDT))

Additional Details about Each Software Component

The following additional details are provided about each software component:

iOS Host

The CDM uses a "Host" interface to communicate with the upper layers of the system. Application-level events and certain system-level services (e.g timers and file I/O) are delegated to the Host object. The CDM makes calls to the Host object to notify the application of events and to access system-level services. The CDM source code contains a simple implementation of the Host interface. In the reference player, a sample iOS Host has been created as an example. **Note**: The actual CDM is included in the CDM dynamic library.

XML Parser

When a manifest file (MPD) is requested by an iOS application, the operating system will be unable to handle it as a video element. The data will need to be parsed and the resulting output used to create HLS playlists as well as fed into the the Universal Dash Transmuxer. Multiple open source projects are available that can be used, such as TBXML, NSXMLParser, TouchXML and others. **Note**: For your convenience, the TBXML XML Parser is downloaded and installed by the setup shell file.

Local Web Server

The Universal DASH Transmuxer (UDT) will create HLS segments and the application will be responsible for creating appropriate playlists to support the newly created content. In order to stream HLS content, the segments must be passed via an HTTP stream. A local web server will be responsible for passing the HLS segment data to the video player. The reference player uses CocoaHTTPServer, but many other options are available. **Note**:

For your convenience, the CocoaHTTPServer is downloaded and installed by the setup shell file.

Video Player Application

The application will connect each of the components listed above. It is the responsibility of the application to initiate the license and content request and pass the the data to the appropriate components. A reference player has been included and built to demonstrate the process of requesting a license through playback.

CDM Dynamic Library (dylib)

The Widevine iOS SDK provides an iOS dylib to ease integration. All assets must be registered with the dylib prior to use. Please use the development dylib or simulator if you need to run a debugger with jailbroken devices. The production release dylib is protected and obfuscated and does not support jailbroken devices.

Library Components

The library contains the following three components:

- Content Decryption Module
- OEMCrypto Library
- Universal DASH Transmuxer

These components will be covered later in the *Dynamic Library Components* section after details about library variations.

Library Variations

The following two dynamic library versions are supported:

Developer Version (Dev)

This version is designed to work using the Widevine test license service (license.uat.widevine.com). The following two builds for developer versions are provided:

- Insecure: This build contains basic obfuscation and is NOT for distribution.
- SIM: Built to work with the simulator, however it will NOT run on a device.

Release Version (Release)

This version is designed to work using the Widevine production license servers. Only the EIT version will be provided.

- Symbols removed with additional security. Will NOT run with debugger or on JailBroken devices.

Note: Unprotected dylibs (using the dev release) will not work in the Widevine production license service environment and can not be commercially released.

Dynamic Library Components

The following components are encrypted, obfuscated, and placed inside the CDM dynamic library:

Content Decryption Module (CDM)

The CDM implements methods and callbacks that are compliant with Common Encryption (CENC) standards. The CDM is responsible for acquiring keys from the license server and handles license exchanges in a secure manner to enable playback of encrypted content[1].

OEMCrypto Library

The purpose of OEMCrypto is to provide an additional layer of security while handling key exchange. The CDM uses OEMCrypto for performing secure operations.

This interface defines a standard set of functions that are needed to securely perform various license protocol, certificate protocol and key handling operations.

Universal Dash Transmuxer (UDT)

The open source DASH transmuxer is used to enable DASH content to be converted to HLS streams that are playable on iOS devices. To enable this functionality the individual DASH files must be passed to the UDT. This process requires a few steps to prepare the DASH content for ingestion [2].

Step 1: Parse Manifest

An XML parser is required to read the request MPD and parse each element. The resultant XML code will be an HLS playlist, needed as input for the UDT. To do this conversion, the reference player uses an open source utility, TBXML, however any XML Parser should be sufficient.

Step 2: Create HLS Playlists

The process to create the playlist will be dependent on the complexity of the incoming MPD. A separate .m3u8 playlist will need to be created for each audio and video track; i.e., an MPD containing 4 separate bitrates of audio and video would result in 8 playlists.

Step 3: Stream Segments

Once the HLS playlists have been created, a localhost URL will need to be sent to video player to initiate the standard HTTP request. The request will need to be intercepted by a locally running proxy and return the transmuxed TS segments back to the player from the UDT. The reference player uses CocoaHTTPServer as an example.

[1] See *Widevine Modular DRM Security Integration Guide for more details.*
[2] *See Universal Dash Transmuxer Project Documentation for more details.*

Recommended Development Path

The following development path is recommended:

1. Use simulator for initial development.
2. Validate against device using Dev build.
3. Once verified on a device, move to the release version of the eval library.

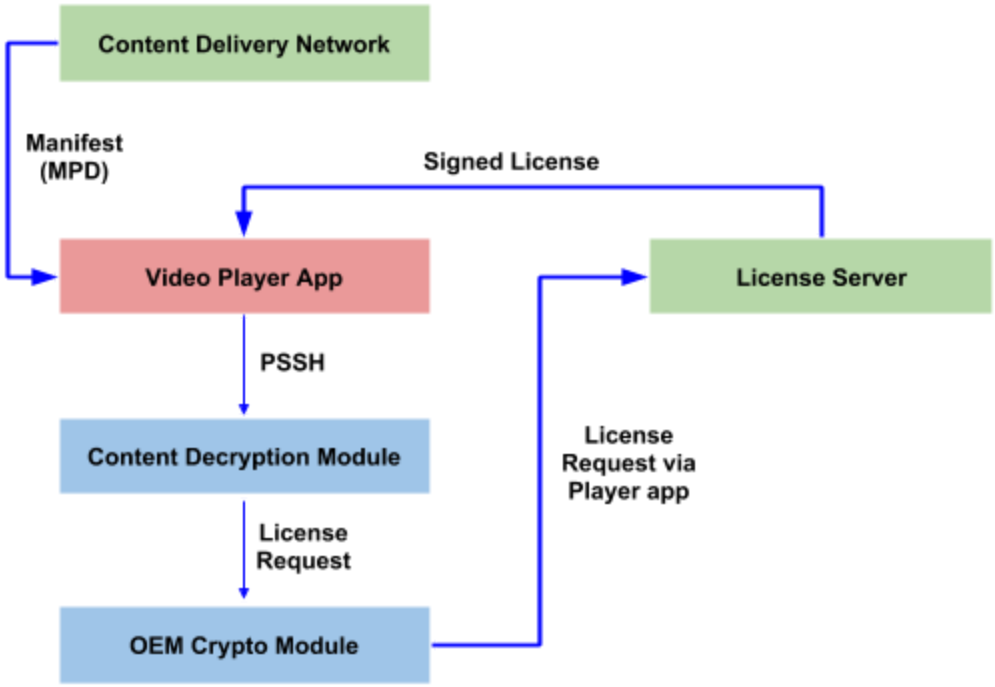
Widevine iOS Encrypted DASH to HLS Workflow

The following diagrams show the flow of the conversion from encrypted DASH content to decrypted HLS content for a player on an iOS device. Because the decryption and demuxing is complicated, the flow is broken into 4 major steps.

1. Get a Signed License
2. Create an HLS Playlist
3. Prepare to Download Content
4. Download and Play Content

Step 1: Getting a Signed License

The Player app requests and receives a manifest from the content delivery network. A PSSH is extracted from the manifest and sent to the CDM. The PSSH is used to generated a license request to the OEMCrypto module, which then passes the request to the Widevine License Server. The License Server then sends a signed license to the Player app. The signed license and the manifest are saved for later steps.



Step 2: Creating an HLS Playlist

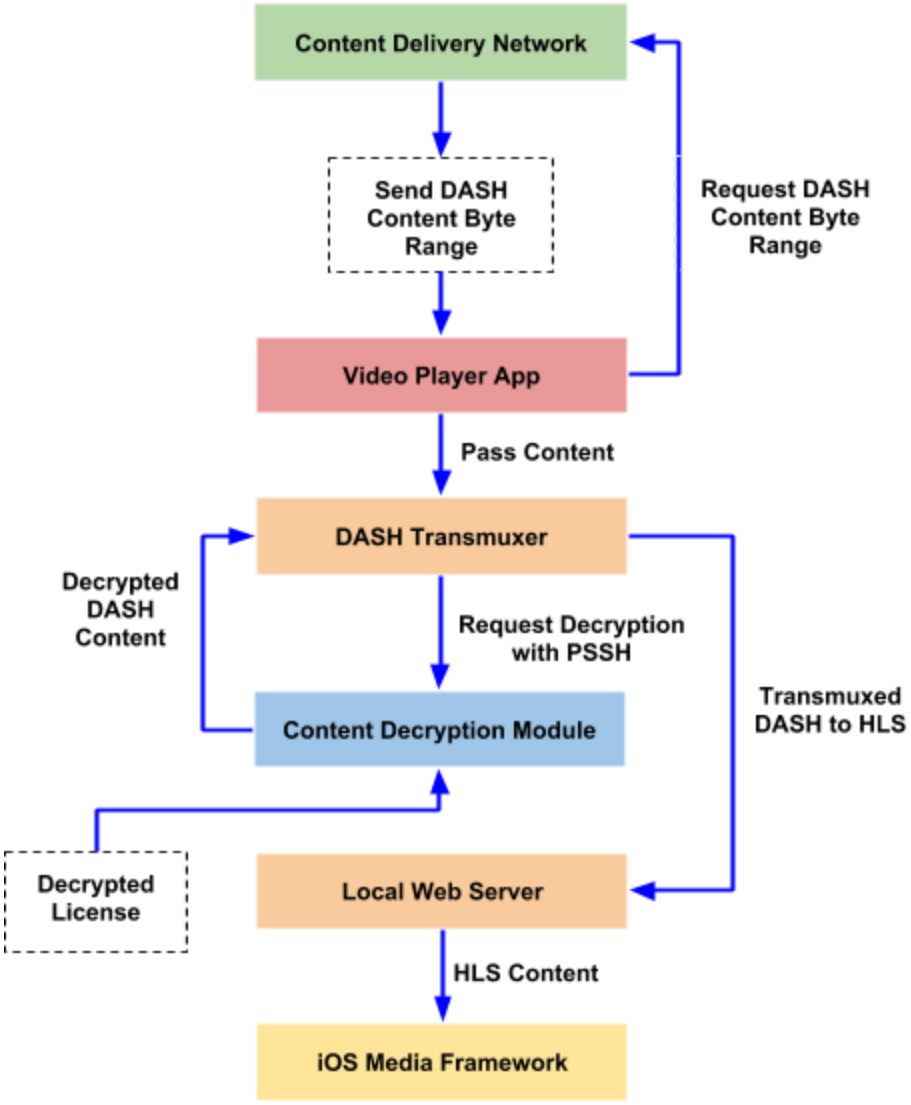
Using the signed license obtained in Step 1, the Player app passes the license to the CDM, which in turn passes a request to decrypt the license to the OEMCrypto module. The decrypted playlist is saved for a later step. Then the Player app sends the manifest to the XML Parser of your choice. The parser converts the manifest to an HLS playlist and saves it for a later step.

Step 3: Preparing to Download Content

Now that the Player app has an HLS playlist, it can send that playlist to the iOS Media Framework. The iOS Media Framework will pass HLS request segments to the local web server, which will then pass segment information to the DASH transmuxer (UDT). The HLS segment information is then mapped to a byte range of DASH content. The Player app is now ready to download content from the Content Delivery Network.

Step 4: Downloading and Playing the Content

The Player app requests a DASH content byte range request to the Content Delivery Network. The CDN then sends the DASH content byte range back to the Player app. Then the content is passed to the DASH Transmuxer (UDT). But before the UDT can transmux the content to HLS, it must verify the content with the Content Decryption Module (CDM) which has the decrypted license to prove that the content has the proper digital rights management. Once the content is verified, the UDT sends the transmuxed HLS content to the local web server, which in turn passes the HLS video to the iOS Media Framework.



Running the Reference Player

The iOS SDK reference player is designed to be a basic implementation to aid in the understanding of the how the CDM operates on iOS using the UDT. The example provided is intended to help developers understand how to build a CDM implementation. The project itself is not intended for commercial release.

The example project contains static links that can be modified.

Steps

- Open the CDMPlayer.xcodeproj in XCode. Do the following:
    - Change Target to CDMPlayer app in XCode at the top of the menu
    - Set output device as iPhone or Simulator

Troubleshooting

- **iOSCdmHost::CreateSession Crash**
  When linking the iOS SDK into your own application, be sure to disable Dead Code stripping which can remove the symbols needed by the CDM to function properly.

- **App Transport Security (ATS)**
  The Reference Player uses a local web server and by default uses HTTP. This is an example implementation and in order to comply with Apple's App store rules, adding a SSL cert for local traffic that enables HTTPS would be the preferred way to integrate the solution. However, using the following values may be useful if using HTTP only for local traffic.

```
<key>NSAppTransportSecurity</key>
<dict>
  <!--To support localhost -->
  <key>NSAllowsLocalNetworking</key>
  <true/>
  <!--To continue to work for iOS 9 -->
  <key>NSAllowsArbitraryLoads</key>
  <true/>
</dict>
```

- **Keychain Sharing**
  If running into OSStatus Error -34018, this is likely due to not enabling Keychain Sharing via the Capabilities section in XCode. As of iOS10, this is now a requirement as the CDM access the keychain to secure key data.