WV Modular DRM Security Integration Guide for Common Encryption (CENC)

Android Supplement

Version 16 - Android R

Revision History

Version	Date	Description	Author
1	3/4/2013	Initial revision	Jeff Tinker, Fred Gylys-Colwell, Edwin Wong, Rahul Frias, John Bruce
2	3/14/2013	Added RSA Certificate Provisioning	Jeff Tinker, Fred Gylys-Colwell
4	4/2/2013	Added Generic Modular DRM	Jeff Tinker, Fred Gylys-Colwell
5	4/3/2013	Updated Testing section	Edwin Wong
6	4/5/2013	Refactored common information into *Widevine Modular DRM Security Integration Guide for CENC.* This document is now the *Android Supplement*.	Jeff Tinker
7-9		skipped so version number matches main doc.	
10	4/1/2015	Update info about optional features and unit tests	Fred Gylys-Colwell
10.1	4/21/2015	Add section on keybox requests	Fred Gylys-Colwell
11	11/1/2015	Updated for OEMCrypto API v11.	Fred Gylys-Colwell
12	9/9/2016	Update for OEMCrypto API v12 -- Provisioning 3.0	Fred Gylys-Colwell
13	5/19/2017	Update path for liboemcrypto.so	Fred Gylys-Colwell
14	5/14/2018	Updated paths and explained Level 3	Fred Gylys-Colwell
15	6/2/2020	Added HDCP SRM updates, Full Decrypt Path Testing, Sandbox Support	Fred Gylys-Colwell
16	6/16/2020	Updated Build File section and updated the reference implementation section	Edwin Wong, Fred Gylys-Colwell
16	07/20/2020	Removed references to android.hardware.drm@1.0	Edwin Wong

Table of Contents

Terms and Definitions

Trusted Execution Environment (TEE) — The portion of the device that contains security hardware and prevents access by non secure system resources.

References

Widevine Modular DRM Security Integration Guide for Common Encryption (CENC)

WVDN Getting Started Guide

Android DRM API for DASH

DASH - 23009-1 MPD and Segment Formats

DASH - 14496-12 ISO BMFF Amendment

DASH - 23001-7 ISO BMFF Common Encryption

Audience

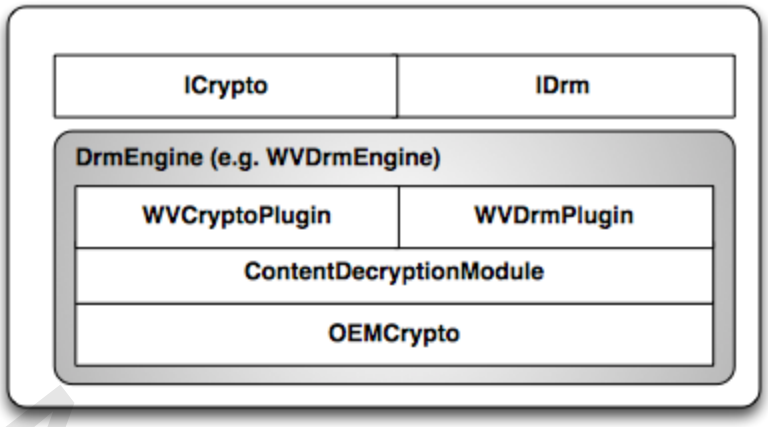
This document is intended for SOC and OEM device manufacturers to integrate with Widevine content protection on android devices.

Purpose

This document defines steps required to build a Widevine DrmEngine component for an android device, and the required functionality of the OEM-provided OEMCrypto library. It contains Android-specific supplemental information for the common document *Widevine Modular DRM Security Integration Guide for Common Encryption (CENC).*

Widevine DrmEngine

The Widevine DrmEngine implements the MediaDrm and Crypto APIs to support content decryption in support of the Android MediaCodec and MediaCrypto APIs. Refer to the “Android DRM API for DASH” document to learn more about how the DrmEngine interacts with these higher level APIs.



The OEMCrypto API defines a hardware abstraction layer to enable the Widevine DrmEngine functionality to be adapted to the underlying hardware feature set.

The remainder of this document defines the OEMCrypto APIs and steps required to build and test the vendor-supplied OEMCrypto implementation library liboemcrypto.so required by the Widevine DrmEngine component on android devices.

Level 3 OEMCrypto Library

For Android, Widevine provides a level 3 OEMCrypto library. The Level 3 library is an obfuscated version of OEMCrypto, and is used as a fall-back for devices that do not provide a level 1, hardware protected, library. You can find the level 3 library in the files vendor/widevine/libwvdrmengine/level3/*/libl3oemcrypto.cpp. The vendor should not have to modify any build files or this file in order to include the level 3 library. The vendor should only provide a level 1 oemcrypto library.

File System

The system mediadrm service stores drm certificates and off-line license information in subdirectories of /data/vendor/mediadrm. You might see two subdirectories: IDM0 and IDM1013. The directory IDM0 is owned by root and is created when you run the unit tests as root -- this directory will not be created on production devices. The directory IDM1013 is owned by media, and will be created by the service at initialization. The directory IDM1013 will be present on a production device.

The location of files has moved from Android O to Android P. If your device will field upgrade from Android O to Android P or Q, you will need to migrate data. If you do not, offline license will not be preserved. See the document “Upgrading Widevine Drm to Pi” for instructions on migrating data.

Build Files

Starting with Android 10 release, Widevine provides an updated manifest to be included as a LOCAL_VINTF_FRAGMENTS so we do not have to modify device-specific manifests. At build time, this manifest is added to the device. Adding an entry here is exactly like adding an entry in the device’s main manifest.

This section describes the steps required to integrate Widevine DRM on Android R devices. The steps are: adding the Widevine service to the device’s build files and setting SELinux permissions.

Adding Widevine to device build files

The <device>/device.mk file for the device must include the following product packages:

```
PRODUCT_PACKAGES += \
    android.hardware.drm@1.3-service.widevine \
    android.hardware.drm@1.3-service.clearkey
```

Setting SELinux permissions

1. Add to <device>/sepolicy/vendor/file.te

```
type mediadrm_vendor_data_file, file_type, data_file_type;
```

2. Add to <device>/sepolicy/vendor/file_contexts

```
/vendor/bin/hw/android\.hardware\.drm@1\.3-service\.widevine
u:object_r:hal_drm_widevine_exec:s0

/data/vendor/mediadrm(/.*)? u:object_r:mediadrm_vendor_data_file:s0
```

3. Add to <device>/sepolicy/vendor/hal_drm_widevine.te

```
allow hal_drm_widevine mediadrm_vendor_data_file:dir create_dir_perms;
allow hal_drm_widevine mediadrm_vendor_data_file:file create_file_perms;
```

Svelte devices -- low-memory footprint

Svelte devices are low memory devices. For example, Android Go devices are svelte. For svelte devices the DRM service should only be loaded when it is needed. Changes to the files above should be:

1. In the device makefile <device>/device.mk, replace the strings **android.hardware.drm@1.3-service.*** with the strings **android.hardware.drm@1.3-service-lazy.***

2. In the file <device>/sepolicy/vendor/file_contexts, replace the permissions specified above with the following line

```
/vendor/bin/hw/android\.hardware\.drm@1\.3-service-lazy\.widevine
u:object_r:hal_drm_widevine_exec:s0
```

Deliverables

The OEMCrypto API implementation should be performed by the vendor. The API is to be implemented in the shared library liboemcrypto.so, which should be placed in /vendor/lib on the device. Alternately, the library may also be placed in one of these directories:

- /system/lib/liboemcrypto.so
- /system/lib64/liboemcrypto.so
- /vendor/lib/liboemcrypto.so
- /vendor/lib64/liboemcrypto.so
- /odm/lib/liboemcrypto.so
- /odm/lib64/liboemcrypto.so

Additional Requirements

In the document **Widevine Modular DRM Security Integration Guide for Common Encryption (CENC),** several features are listed as optional. All android devices must be provisioned with a production keybox or an OEM certificate. The Session Usage Table is not optional for Android devices. An Android device will not pass GTS testing as a Level 1 device unless it can support the Session Usage Table API. The Generic Crypto API functions are not optional. These functions must be implemented in the secure processor such that keys are not visible to the unsecure OS. Certificate functionality is not optional. Devices must be able to load a different “app specific” RSA certificate in each session.

The only functions described in **Widevine Modular DRM Security Integration Guide for CENC** that are optional are the “Very Optional Features”:

- **OEMCrypto_WrapKeyboxOrOEMCert** - not used by CDM code. OEMs may wish to implement this to facilitate provisioning the device.

- **OEMCrypto_InstallKeyboxOrOEMCert** - OEMs may wish to implement this to facilitate provisioning the device -- in particular, at initialization, if IsKeyboxOrOEMCertValid() returns

false, the widevine code will look for a file called /factory/wv.keys and call OEMCrypto_InstallKeyboxOrOEMCert with that file. This code does not check the provisioning method first -- we expect the code path to work whether the device supports a keybox or an OEM certificate. It is OEMCrypto's responsibility to verify the file passed into this function.

- **HDCP SRM Updates** - SRM Updates are used to install a new HDCP device revocation list. This is not required by any content providers at this time. Some studios are requesting it.

- **Full Decrypt Path Testing** - This is recommended for testing only. Not required for any production device.

- **Sandbox Support** - Only used on specialized devices, such as Virtual Machines or desktop applications. Not required by any content providers. Several in-vehicle entertainment systems are planning to use this to distinguish between separate seats in a car.

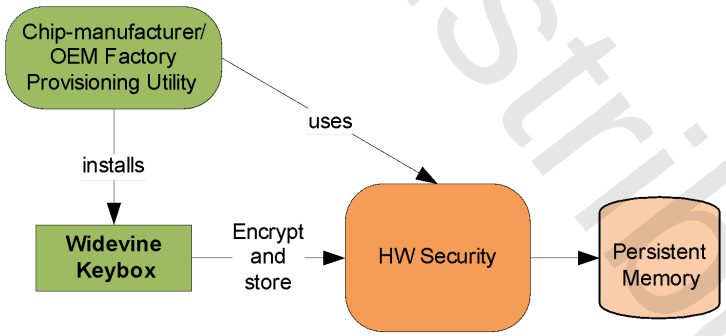
Keybox Requests and Installation Process

Factory Provisioning 3.0

Level 1 Android devices are required to be either factory provisioned with a keybox or factory provisioned with on OEM certificate. For a complete description of OEM certificates, please see the document Widevine_DRM_Device_Provisioning_Models.pdf, found in the docs directory .

Keybox Factory Provisioning

In Keybox Factory provisioning, the manufacturer obtains keyboxes from Widevine, which are then installed on devices during manufacturing. The keybox must be installed in a partition or region of persistent memory that cannot be erased due to a factory reset or other software operation.



Keybox or OEM Certificate Requests and Installation Process

Keybox or OEM Certificate Requests

If you currently work with the Android team directly and have an Android TAM, please use APFE through the partner.android.com portal for managing your Android devices and keyboxes.

If you are not working with and Android TAM, you may use the Widevine Developer Network to request keyboxes or OEM certificates. See the document "WVDN Getting Started Guide" for details. You may contact Widevine support through https://support.google.com/widevine/contact/wv_oemcf.

Keybox XML File Format

An example keybox file is shown below:

```
<?xml version="1.0"?>
<Widevine>
<NumberOfKeyboxes>2</NumberOfKeyboxes>
<Keybox DeviceID="mfg_mod123_0000001"><Key>c5f5cf3c2cb2ce175f2f5337a2f8f8ab</Key>
<ID>9d56e4931762b52aa21e4e590df477b5c81c683e0579f041ffa21f875c4c5e4a1cd4c2331e27e3f4a4
9352fb432557336f63b1cb62549fddc9224b84d0c0364c827365fc217d9cb0</ID>
<Magic>6b626f78</Magic>
<CRC>0b11b841</CRC>
</Keybox>
<Keybox DeviceID="mfg_mod123_0000002"><Key>73e38eb4f313e4fce8a5ab547cc7e2c0</Key>
<ID>215a40a9d13da3a9648335081a182869cbe78f607ce3ceb7506f351a22f411ae3f324ab5f5bfb7c542
ffcd38ec09438e7f92855149b02921463153c441332d7a21f875c4c5e4a1cd </ID>
<Magic>6b626f78</Magic>
<CRC>2b4c5e9f</CRC>
</Keybox>
</Widevine>
```

Keybox Installation

The utility for installing a keybox on the device during manufacturing needs to be defined and implemented by the manufacturer. To assist with this process, Widevine provides sample source code for translating a keybox in XML file format into a byte sequence that can be installed on the device.

The keybox must be encrypted with an OEM root key, sometimes called a "key encryption key" using AES-128 or stronger encryption. Once encrypted, the keybox must be stored in a non-erasable persistent memory region or file on the device. The keybox is accessed using the OEMCrypto Keybox Access APIs.

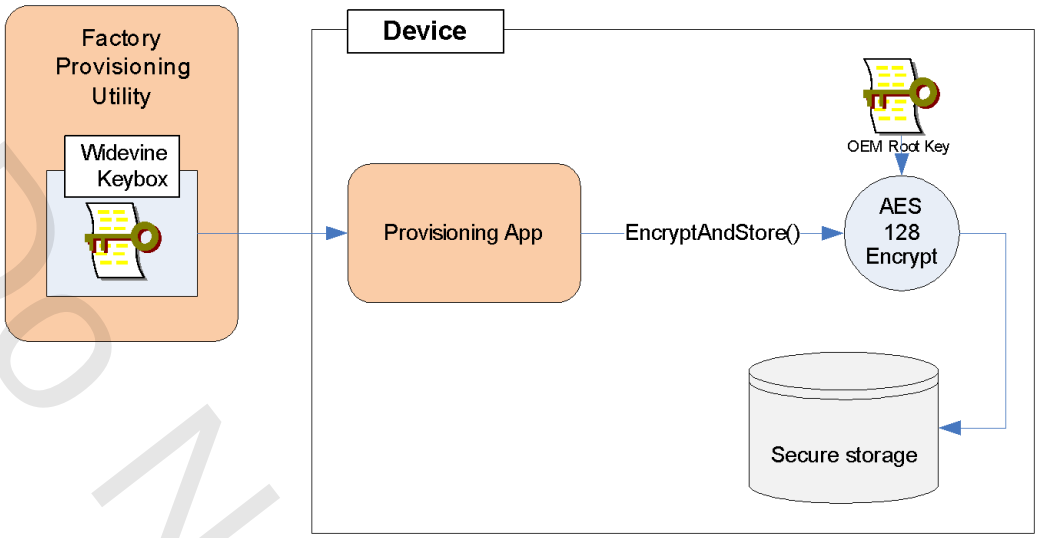


FIGURE 1. FACTORY PROVISIONING KEYBOX INSTALLATION

If the facilities of the secure environment on the device are not available at the time of factory provisioning, the manufacturer may implement the two-stage WrapKeybox and InstallKeybox method of provisioning described in more detail in the main document in the section titled "Provisioning API".

Destroy keybox file after installation

The clear keybox file must be destroyed after installation using PGP shredder.

Unit and Integration Testing

A unit test validates a single piece of functionality, in isolation from the rest of the system. The unit test class typically contains unit tests for all of the methods of a single C++ source file. An integration test combines various components and tests the system as a whole.

A number of unit and integration tests are provided for vendors to verify the basic functions of their implementation. The tests utilize Google C++ Testing Framework, which can be found in the Android tree under external/gtest. It can also be downloaded from Google C++ Testing Framework.

Unit tests for OEMCrypto are found in the android source tree, in the directory
`$ANDROID_BUILD_TOP/vendor/widevine/libwvdrmengine/`

Setting Up the Build Environment

Before building any tests, please setup the build environment in the local branch

```
. build/envsetup.sh
lunch <Your TARGET>
```

The unit tests depend on several libraries, it is best to build the entire tree at least once:

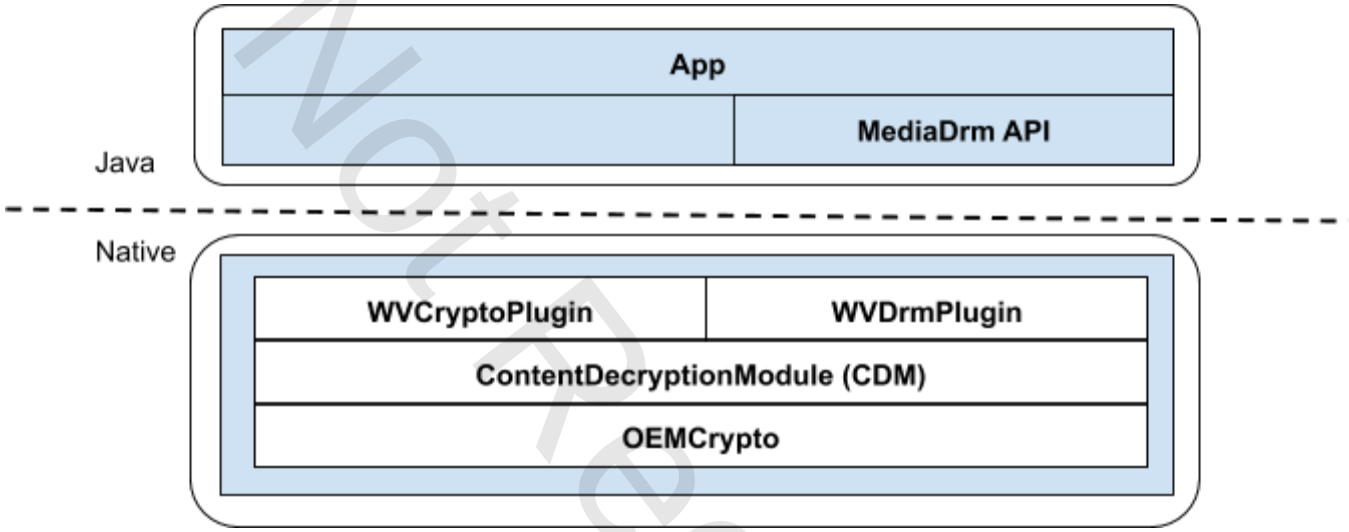
```
make -j 12
```

Reference Implementation

The reference implementation is sample code for all the features for OEMCrypto. It should **not** be included in a production device. The source code is intended for reference only.

Targeted Components

The tests provided verify the following emboldened components:



Testing OEMCrypto Library

Out of the box, you should be able to compile existing unit tests. Build the unit tests for oemcrypto.so:

```
cd $ANDROID_BUILD_TOP/vendor/widevine/libwvdrmengine/oemcrypto/test
mm
```

If the command “mm” fails because of missing dependencies, try the command “mma”. The command “mma” takes longer, but it scans all the makefiles in the tree and attempts to make all dependencies.

Run the existing unit tests:

```
cd $ANDROID_BUILD_TOP
adb root
adb remount
adb disable-verity
```

You may need to run `adb reboot` after disabling verity.

```
adb push $OUT/data/bin/oemcrypto_test /data/bin/
adb shell /data/bin/oemcrypto_test
```

You should see output that starts with:

```
      uses_keybox = true.
```

```
loads_certificate = true.
uses_certificate = true.
generic_crypto = true.
api_version = 15.
usage_table = true.
cast_receiver = false.
resource_rating = 3, security level L1.
LOAD_TEST_KEYBOX: Call LoadTestKeybox before deriving keys.
Note: Google Test filter = *-*ForceKeybox*:*CastReceiver*:*Performance*
[==========] Running (around 300) tests from 23 test cases.
[----------] Global test environment set-up.
[----------] 16 tests from OEMCryptoClientTest
[ RUN      ] OEMCryptoClientTest.VersionNumber
             OEMCrypto Security Level is L1
             OEMCrypto API version is 15
             OEMCrypto supports usage tables.
[       OK ] OEMCryptoClientTest.VersionNumber
```

… and ends with:

```
[----------] Global test environment tear-down
[==========] (around 300) tests from 23 test cases ran. (229937 ms total)
[  PASSED  ] (all of them) tests.
```

Near the top you should see the test NormalGetDeviceId or GetDeviceId, which should print out your device's device ID. Make sure your security level is correct. There is also a test that prints out the system ID:

```
[----------] 3 tests from OEMCryptoKeyboxTest
[ RUN      ] OEMCryptoKeyboxTest.NormalGetKeyData
             NormalGetKeyData: system_id = 4445 = 0x115D, version=2
```

If you see the system ID 8158, then your liboemcrypto.so library did not load correctly, and you are running with level 3 fallback.

Starting with version 10 of the API, the unit test program will filter out tests that are not expected to run. For example, if you have not implemented usage tables yet, it will not run most of those tests. Instead, you will see

```
[   FAILED ] OEMCryptoAndroidLMPTest.SupportsUsageTable
```

Once you have usage tables supported, you will see the rest of those tests running. Near the top of the output is the test filter. Most devices should have a filter of

```
GTest Filter: *-*ForceKeybox*:*CastReceiver*:*Performance*
```

If your device has a keybox, then the Prov30 tests for provisioning 3.0 will be filtered out. Otherwise, the keybox tests will be filtered out.

Cast Receiver or Android TV

Android TV devices should implement functionality to sign with an alternate RSA certificate using the alternate RSA padding schemes. To test this functionality, you should pass the argument "--cast" to the oemcrypto_test program:

```
adb shell /data/bin/oemcrypto_test --cast
```

This tells the unit tests not to filter out the CastReceiver tests.

Testing ContentDecryptionModule

The full suite of tests can be built and run using the provided script.

```
cd $ANDROID_BUILD_TOP/vendor/widevine/libwvdrmengine
./build_and_run_all_unit_tests.sh
```

Two of the tests are used to verify the CDM functions, they are request_license_test and cdm_engine_test. If the script fails because of missing dependencies, try the command “mma” in the offending directory. The command “mma” takes longer than “mm”, but it scans all the makefiles in the tree and attempts to make all dependencies.

The request_license_test uses wvcd::WvContentDecryptionModule interface. The tests include generating and sending a license request to the license server and verifying the response coming back from the server. It also performs query key status and query status tests.

The cdm_engine_test is a unit test that calls the cdm_engine directly to generate a license request and sends it to the license server, then verifies the response coming back from the server.

There are other tests that verify various other cdm code. Please review the script to see a full list of tests.

Testing Java Drm API and Plugins

The plugin tests include tests for the Java Drm API for DASH, WVCryptoPlugin, WVDrmPlugin and WVDrmPluginFactory.

These are isolated unit tests for the top level components of the DRM engine, they do not exercise the OEMCrypto API.

Build libwvdrmdrmplugin_test from vendor/widevine/libwvdrmengine/mediadrm/test.

```
cd $ANDROID_BUILD_TOP/vendor/widevine/libwvdrmengine/mediadrm/test
mma
adb push $OUT/data/bin/libwvdrmdrmplugin_test /data/bin
adb shell /data/bin/libwvdrmdrmplugin_test
```

Build libwvdrmmediacrypto_test from vendor/widevine/libwvdrmengine/mediacrypto/test.

```
cd $ANDROID_BUILD_TOP/vendor/widevine/libwvdrmengine/mediacrypto/test
mma
adb push $OUT/data/bin/libwvdrmmediacrypto_test /data/bin
adb shell /data/bin/libwvdrmmediacrypto_test
```

Build libwvdrmengine_test from vendor/widevine/libwvdrmengine/test/unit.

```
cd $ANDROID_BUILD_TOP/vendor/widevine/libwvdrmengine/test/unit
mma
```

```
adb push $OUT/data/bin/libwvdrmengine_test /data/bin
adb shell LD_LIBRARY_PATH=/system/vendor/lib/mediadrm/ /data/bin/libwvdrmengine_test
```

Play Back Testing -- ExoPlayer

ExoPlayer is an open source demo application that plays a variety of video, including Widevine DRM protected content. Source code can be found on GitHub: https://github.com/google/ExoPlayer. A prebuilt copy of the ExoPlayer application can be installed from the android source tree:

```
adb install vendor/widevine/libwvdrmengine/test/demo/ExoPlayerDemo.apk
```