Google Confidential

Widevine Security Integration Guide for Android-based Devices

Version 2.2

Google Inc.
747 6th Street South
Kirkland, WA 98033 USA

425.739.5600 voice
425.968.9399 efax
www.widevine.com

WIDEVINE®

1 Revision History

11/02/2011	1.0	Initial version	Jeff Tinker, Edwin Wong
11/07/2011	1.1	Updated keybox request email address, fixed incorrect OEMCrypto_EnterSecurePlayback, OEMCrypto_ExitSecurePlayback listing in crypto device control API table, added OEMCrypto_Open, OEMCrypto_Close. Removed some methods that had been incorrectly marked as required for level 3.	Jeff Tinker
11/22/2011	1.2	Corrected API levels for OEMCrypto_Open, OEMCrypto_Close in tables. Corrected OEMCrypto_DecryptAudio and OEMCrypto_DecryptVideo that were incorrectly marked as used in level 2	Jeff Tinker
12/6/2011	1.3	Corrected inconsistent error codes for SHORT_BUFFER and NO_DEVICEID	Jeff Tinker
02/08/2012	1.4	Add “Device SOC” to 5.5.1.1 Request Body (for keybox)	Edwin Wong
02/21/2012	1.5	Update 6.4.3 OEMCrypto_EncryptAndStoreKeybox: this API is used by L1-L3, not just L1 and L3; this API should fail the call if the device is unlocked and running at L3	Edwin Wong
02/27/2012	1.6	Updated 6.4.3 as per Guru’s review: L1 does not require a signed system image, OEMCrypto_EncryptAndStoreKeybox must not fail the call if the device is unlocked	Edwin Wong
04/14/2012	1.7	Updated 6.5.1 to correct CRC algorithm used in keybox computation and 6.3.4 to use an all-zeros initialization vector in OEMCrypto_DecryptAudio Updated 5.1 white listing information, 5.2 to correct CRC used in keybox and 5.5.1 to update keybox request body	Rahul Frias Edwin Wong
7/12/2012	1.8	Added deliverable build instructions for Jellybean – note the new build step for libdrmdecrypt.so to support MediaCodec mode	Jeff Tinker
10/18/2012	1.9	Added ro.com.widevine.cachesize property to configure Widevine stream cache size for High Definition content. This read only property allows devices to tune memory used by Widevine plugin for streaming HD content. This property is added for Android 4.2 (Jelly Bean MR1) release.	Edwin Wong
08/28/2013	2.0	Clarify why Widevine does not provide L2 libraries.	Edwin Wong

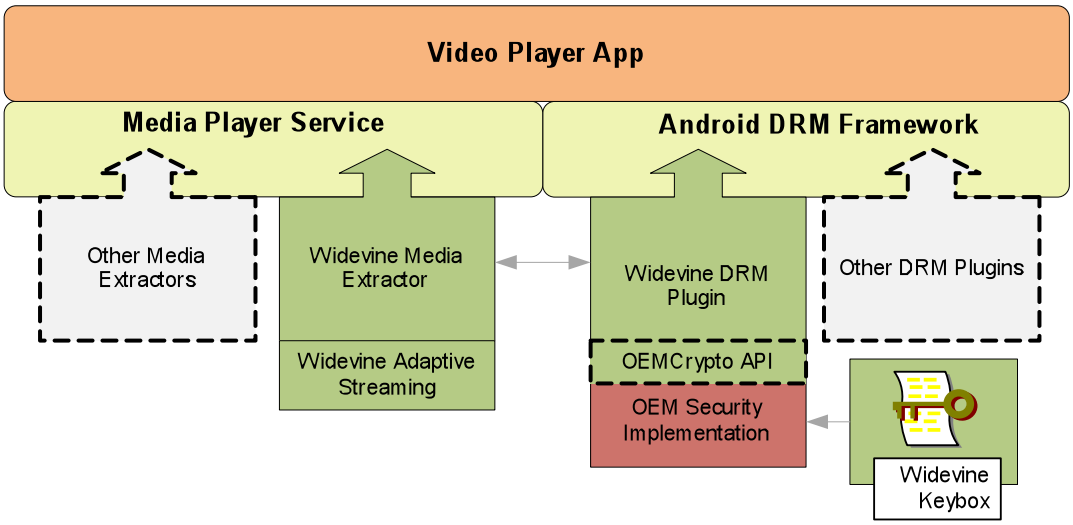
		In short, the difference between L1 and L2 is the protected video path, which is done in the SoC/vendor integration and does not affect the libraries. Integrators may implement L2 using the L1 libraries, and not providing a protected video path. Section 2 and section 7 are updated.	
04/23/2014	2.1	Update get keybox process in section 5.	Edwin Wong
09/10/2014	2.2	Update “Deliverables” section for KitKat and L	Edwin Wong

Table of Contents

2 Introduction

Widevine content protection is integrated with Android devices that use Android OS version 3.0 and later. These Android-based devices include tablets, phones, Smart TVs, Set Top Boxes and other devices capable of playing premium video content.

There are two main components involved in Widevine content protection in the Android media system. The *Widevine Media Extractor* includes adaptive streaming and container parsing for Widevine encrypted content. The *Widevine DRM Plug-in* implements secure key management and content decryption. It is a plug-in component in the Android DRM framework.



The security level of the content protection provided by the Widevine DRM plugin depends on the security capabilities of the underlying hardware platform. Ideally, a combination of hardware security functions including a trusted boot mechanism, an isolated secure OS for handling security functions and hardware protected video path can be provided. But not all devices have the underlying support for all of these features. Because of this, several levels of security are defined, depending on the hardware capabilities of the device and the Android platform integration:

Security Level	Secure Boot Loader	Widevine Key Provisioning	Security Hardware or ARM Trust Zone	Widevine Keybox and Video Key Processing	Hardware Video Path
Level 1	Yes	Factory	Yes	Keys never exposed in clear to host CPU	Hardware Protected Video Path
Level 2	Yes	Factory	Yes	Keys never exposed in clear to host CPU	Clear video streams delivered to decoder via an unprotected Video Path
Level 3	Yes	Field	No	Clear keys exposed	Clear video streams

				to host CPU	delivered to decoder via an unprotected Video Path

The OEMCrypto API defines a hardware abstraction layer to enable the Widevine DRM plugin functionality to be adapted to the underlying hardware feature set. There are different variants of the OEMCrypto API defined for each security level, since the types of interactions with the hardware vary by level.

It is recommended that new device designs implement Widevine Level 1 security. Level 3 security with field provisioning is only recommended for legacy devices that were not factory provisioned with a Widevine keybox at the time of manufacture.

2.1 Purpose

This document describes the processes device manufacturers use to provision Android-based devices for use with Widevine DRM prior to delivery to customers as well as defining the security APIs that are used to protect decrypted keys and decrypted video data.

3 References

Widevine Security Model for Internet Devices: Widevine security overview

Widevine Factory Provisioning Guide: Widevine general provisioning document

Movies on Android Devices: Device requirements for Widevine integration

4 Terms and Definitions

Device Id A null-terminated C-string uniquely identifying the device. Device Id is a maximum of 32 characters including NULL termination.

Keybox Widevine structure containing keys and other information used to establish a root of trust on a device. The keybox is either installed during manufacturer or in the field. Factory provisioned devices have a higher level of security and may be approved for access to higher quality content.

Device Key 128 bit AES key used to secure entitlements, provided by Widevine.

SystemID Unique ID assigned by Widevine to each customer, included in keybox

5 Device Provisioning

5.1 Overview

A Widevine keybox is installed on a device to establish a root of trust, which is used to secure content on the device. The device's security hardware, where applicable, is used to protect the contents of the keybox when it is stored. The device key in the keybox is used in the process of decrypting the media content played by the device.

Keyboxes may be installed on devices using Field provisioning or Factory provisioning. Field provisioning is only used for Level 3 implementations on legacy devices that were not provisioned with Widevine keys during manufacturing, or devices that do not have security hardware to protect their keys. Level 1 and 2 devices must be factory-provisioned and the keybox must be encrypted by an on-chip AES device unique secret key before being stored into non-erasable persistent memory.

Each Widevine keybox is associated with a device ID. Every device should have a unique ID. For factory-provisioned devices, the manufacturer will assign the ID when requesting keyboxes. For field-provisioned devices, the device ID must be provided by a function on the device.

In addition to the device ID, there is a Widevine-assigned system ID in the keybox that ensures keyboxes are unique across manufacturers. Two manufacturers may use the same device ID since they will have different system IDs. Widevine assigns system IDs based on the Manufacturer/Brand, device type and model year in the keybox request. The Manufacturer/Brand field in the keybox request is not case sensitive.

On the other hand, the manufacturer and model names are provided to Google for both staging and production server whitelisting. Both manufacturer and model names are case sensitive. These manufacturer and model fields submitted must match the corresponding ro.product.manufacturer and ro.product.model fields returned by "adb shell getprop". For field- provisioned devices, the provisioning client provides the system ID.

5.2 Keybox Definition

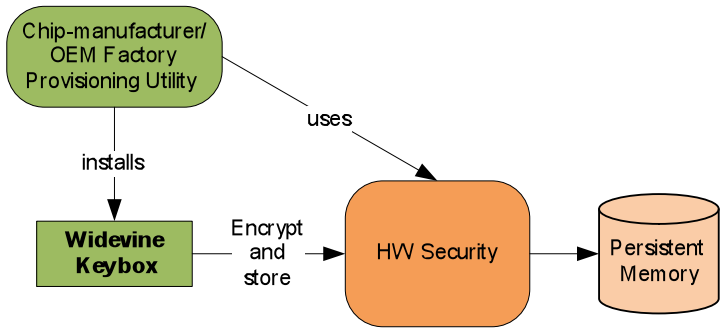
A Widevine keybox contains a device unique ID, device Key, encrypted key data and two fields for verifying the keybox validity: a constant code and a CRC.

Field	Description	Size (bytes)
Device ID	C character string identifying the device, null terminated.	32
Device Key	128 bit AES key assigned to device, generated by Widevine.	16

Key Data	Encrypted data	72
Magic	Constant code used to recognize a valid keybox: “kbox” (0x6b626f78)	4
CRC	CRC-32 POSIX-1003.2 validates integrity of the key data field	4
	Total Size	***128***

5.3 Factory Provisioning

In 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.



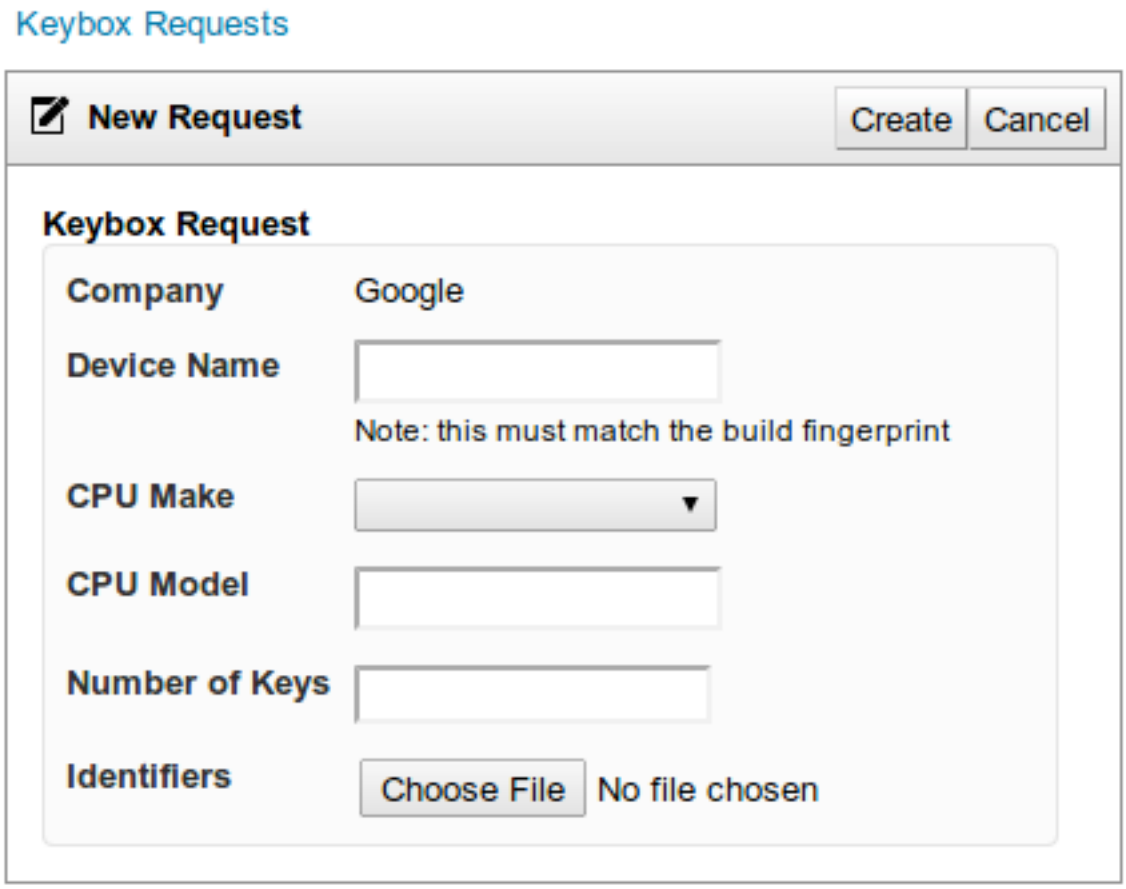
5.4 Keybox Requests and Installation Process

APFE now supports requests and downloads of keyboxes through https://partner.android.com. The process of obtaining keyboxes from Widevine for factory provisioning is outlined below.

5.4.1 Keybox Requests

1. First let your Android Technical Account Managers know you desire it.
2. They will enable requests and downloads for the users you specify.
3. Once done, your users can click **Make Request** under *Widevine Keyboxes* in the left navigation to start a request.

4. Have your users enter the keybox request details in the New Request screen.



5. Then click **Create**.
6. Download the file, which is generally a long list of IDs as per Widevine instructions.

Please contact your Technical Account Manager with any questions.

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>9d56e4931762b52aa21e4e590df477b5c81c683e0579f041ffa21f875c4c5e4a1cd4c2331
e27e3f4a49352fb432557336f63b1cb62549fddc9224b84d0c0364c827365fc217d9cb0</ID>
<Magic>6b626f78</Magic>
<CRC>0b11b841</CRC>
</Keybox>
<Keybox
DeviceID="mfg_mod123_0000002"><Key>73e38eb4f313e4fce8a5ab547cc7e2c0</Key>
<ID>215a40a9d13da3a9648335081a182869cbe78f607ce3ceb7506f351a22f411ae3f324ab5f
5bfb7c542ffcd38ec09438e7f92855149b02921463153c441332d7a21f875c4c5e4a1cd </ID>
<Magic>6b626f78</Magic>
<CRC>2b4c5e9f</CRC>
</Keybox>
</Widevine>
```

5.4.2 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.

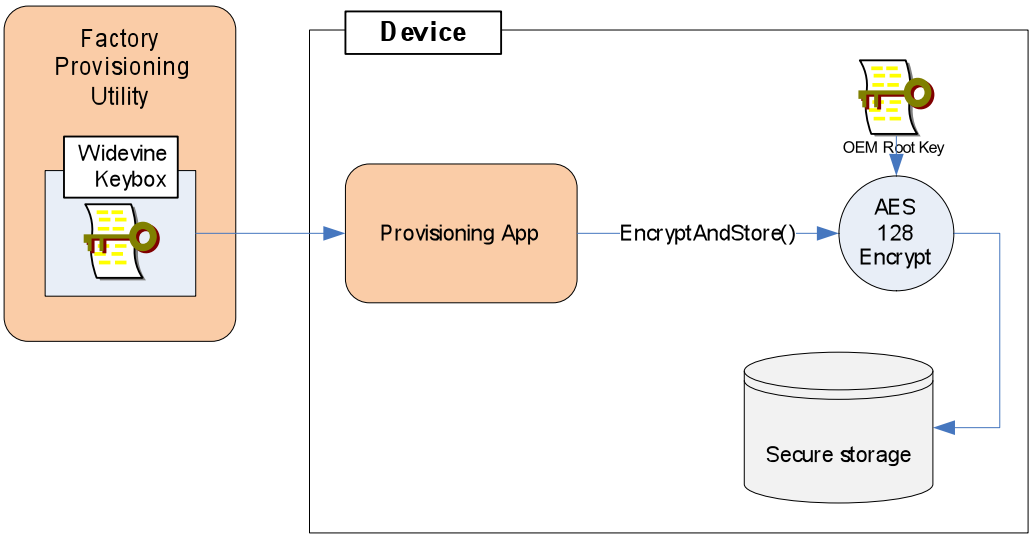


FIGURE 1. FACTORY PROVISIONING KEYBOX INSTALLATION

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 section 6.4.

5.4.3 Destroy keybox file after installation

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

5.5 Field Provisioning

In Field provisioning, the Widevine Field Provisioning Client either creates or receives a Widevine keybox, then encrypts and store the keybox in persistent memory using the OEMCrypto API, as shown in Figure 2.

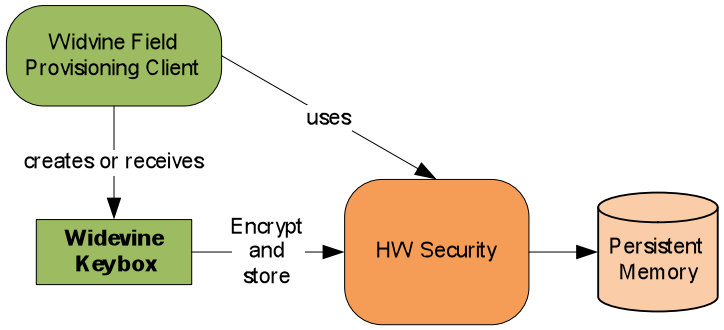


FIGURE 2 - FIELD PROVISIONING OVERVIEW

When the Widevine Field Provisioning Client is activated on the device, it first checks if a valid keybox has been loaded. During this check, the device ID, magic keybox identifier ("kbox") and CRC-32 fields of the keybox are validated. If any of these fields indicate that the keybox is invalid, the device will initiate a field provisioning operation. These flows are shown in the following sequence diagrams:

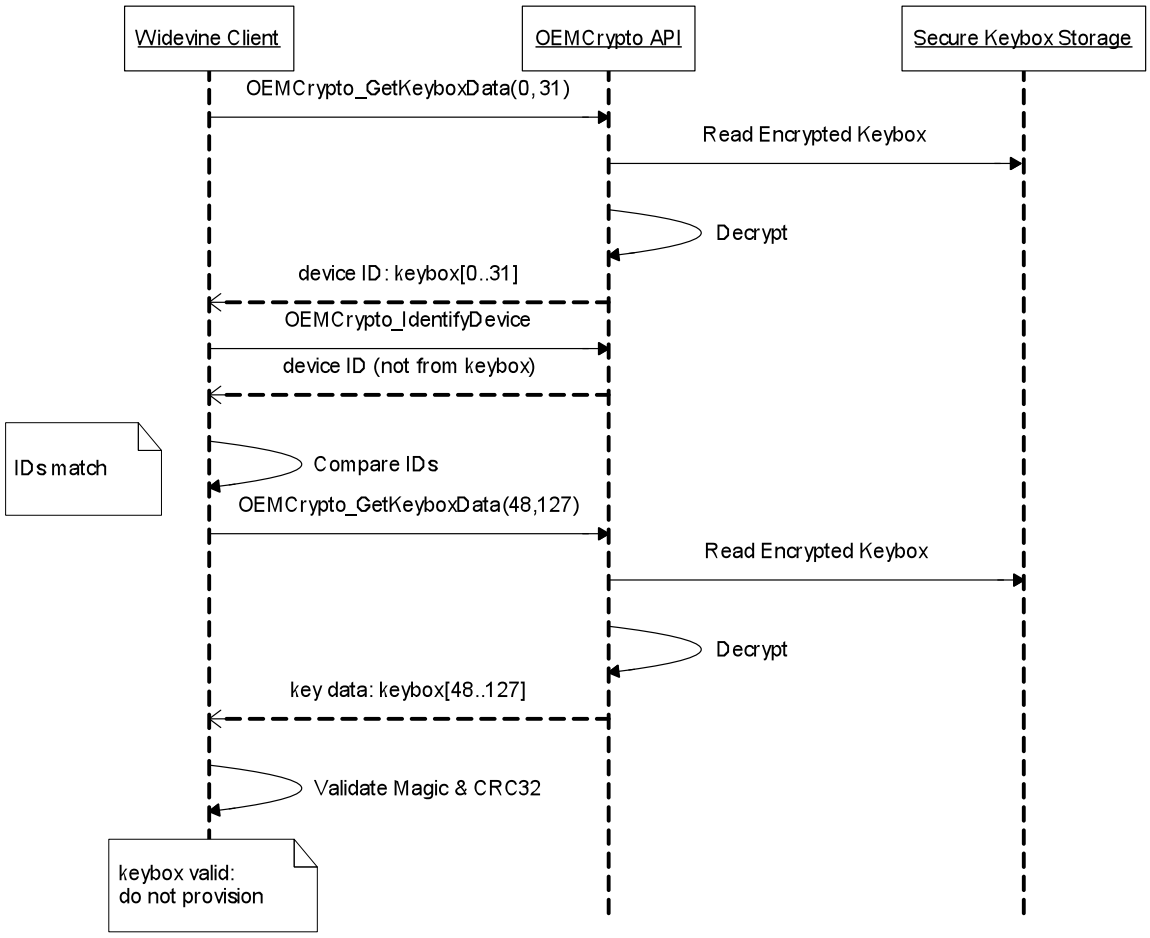


FIGURE 3 - FIELD PROVISION - KEYBOX INSTALLED

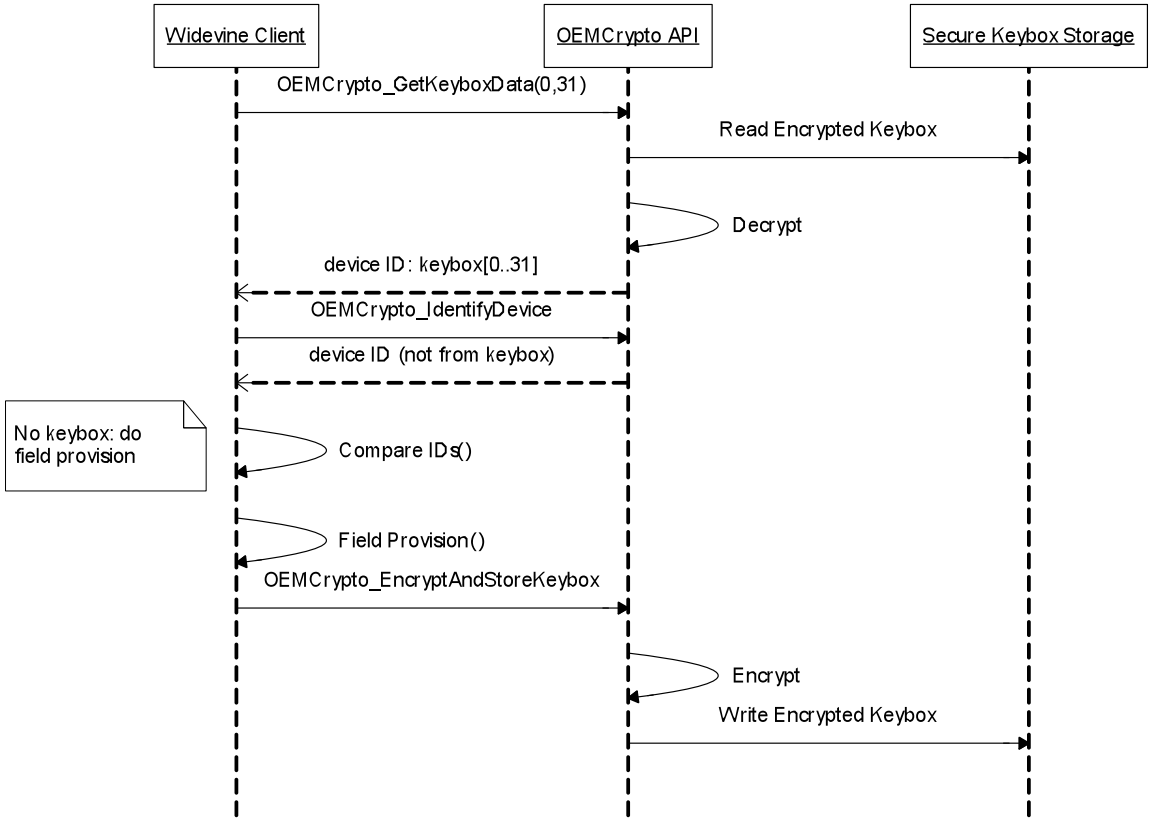


FIGURE 4 - KEYBOX INVALID – DO FIELD PROVISIONING

The keybox must be stored in a region of memory that becomes inaccessible when the device is unlocked.

6 OEMCrypto APIs

The OEMCrypto APIs are divided into 5 areas:

- Crypto Device Control
- Crypto Key Ladder
- Video Path
- Keybox Provisioning
- Keybox Access

Device manufacturers implement the API as a static library, which is linked into the Widevine DRM plugin.

6.1 Crypto Device Control API

The Crypto Device Control API involves initialization of and mode control of the security hardware.

The following table shows the APIs required by each security level:

Crypto Device Control API	Level 1	Level 2	Level 3
OEMCrypto_Initialize	✓	✓	
OEMCrypto_Terminate	✓	✓	
OEMCrypto_Open		✓	
OEMCrypto_Close		✓	

6.1.1 OEMCrypto_Initialize

API Levels	1	2	~~3~~

The API initializes the crypto hardware.

```
API: OEMCryptoResult OEMCrypto_Initialize(void);

Parameters:
   •  none


Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_INIT_FAILED failed to initialize crypto hardware
```

6.1.2 OEMCrypto_Terminate

API Levels	1	2	~~3~~

The API closes the crypto operation and releases all resources used.

```
API: OEMCryptoResult OEMCrypto_Terminate(void);

Parameters:
   •  none

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_TERMINATE_FAILED failed to de-initialize crypto
      hardware
```

6.1.3 OEMCrypto_Open

API Levels	~~1~~	2	~~3~~

Open the crypto security engine and provide a block of memory that can be used for crypto operations. If hardware decryption is used, this block of memory must be accessible by the decrypt hardware.

```
API: OEMCryptoResult OEMCrypto_Open(OEMCrypto_UINT8 **buffer,
                                    OEMCrypto_UINT32 *bufferSize);

Parameters:
   •     buffer: buffer to return the top address of I/O-buffer (Network-buffer)
   •     bufferSize: on return, set to the size of the allocated I/O-buffer (recommend 16 MB)


Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_OPEN_FAILURE failed to open
```

6.1.4 OEMCrypto_Close

API Levels	~~1~~	2	~~3~~

Close the crypto security engine and free the memory used for adaptive/decryption.

```
API: OEMCryptoResult OEMCrypto_Close(void);

Parameters:
   •  none

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_CLOSE_FAILURE failed to close
```

6.2 Crypto Key Ladder API

The crypto key ladder protects clear crypto keys from being exposed to non-secure system resources. The Crypto Key Ladder API requires the device to provide hardware support for AES-128 ECB and CBC modes and prevent clear keys from being exposed to the CPU.

The following table shows the APIs required by each security level:

Crypto Device Control API	Level 1	Level 2	Level 3
OEMCrypto_SetEntitlementKey	✓	✓	
OEMCrypto_DeriveControlWord	✓	✓	

6.2.1 OEMCrypto_SetEntitlementKey

API Levels	1	2	~~3~~

The API decrypts the entitlement (EMM) key, also known as the asset key, using the encrypted device key (Device Key field) in the Widevine Keybox.

As shown in Figure 1 on the next page, Step 1 uses an OEM root key to decrypt (AES-128-ECB) the Device Key in the Keybox; the result is “latched” in hardware key ladder.

Step 2 uses the “latched” clear device key to decrypt (AES-128-ECB) the entitlement key passed in as the *emmKey parameter and “latched” the clear entitlement key in hardware for the next operation.

```
API: OEMCryptoResult OEMCrypto_SetEntitlementKey(
         const OEMCrypto_UINT8* emmKey, const OEMCrypto_UINT32 emmKeyLength);

Parameters:
   •  emmKey (in) - pointer to the encrypted entitlement key
   •  emmKeyLength (in) – length of entitlement key in bytes

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_NO_DEVICE_KEY failed to decrypt device key
      OEMCrypto_ERROR_NO_ASSET_KEY  failed to decrypt asset key
      OEMCrypto_ERROR_KEYBOX_INVALID cannot decrypt and read from Keybox
```

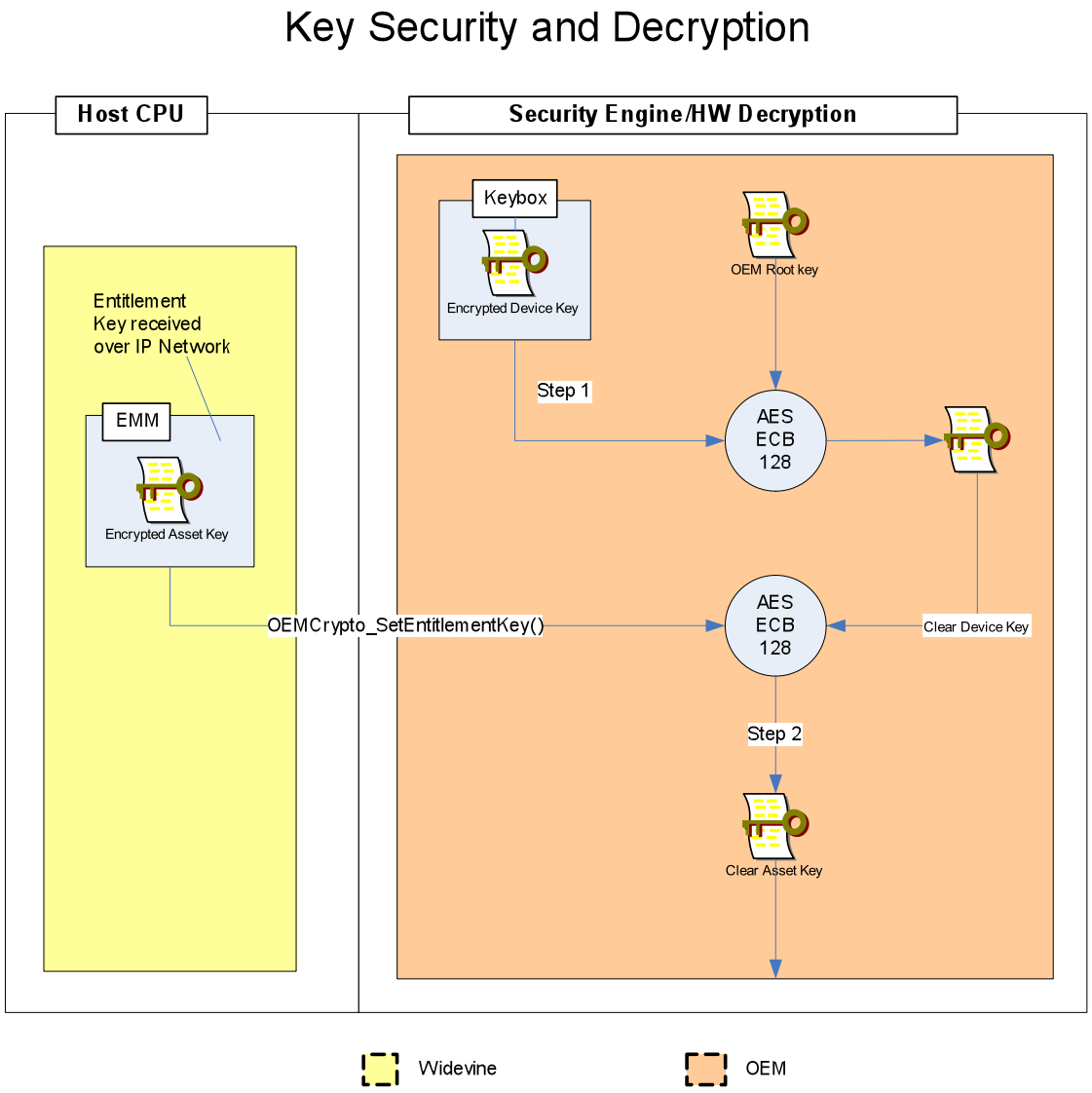


FIGURE 5 - SET ENTITLEMENT KEY

6.2.2 OEMCrypto_DeriveControlWord

API Levels	1	2	~~3~~

Using the active key ladder key from OEMCrypto_SetEntitlementKey(), decrypts (AES-128-CBC, iv=0) the 32-byte ECM referenced by the *ecm parameter; returns in *flags the first clear 4 bytes data. “Latched” the clear bytes [4..20] as the clear control word for subsequent payload decryption operation.

```
API: OEMCryptoResult OEMCrypto_DeriveControlWord(const OEMCrypto_UINT8* ecm,
        const OEMCrypto_UINT32 length, OEMCrypto_UINT32* flags);

Parameters:
   •  ecm (in) - points to encrypted ECM data
```

- length (in) – length of encrypted ECM data in bytes
- flags (out) - points to buffer to receive 4 byte clear flag value

```
Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_NO_CW cannot decrypt control word
```

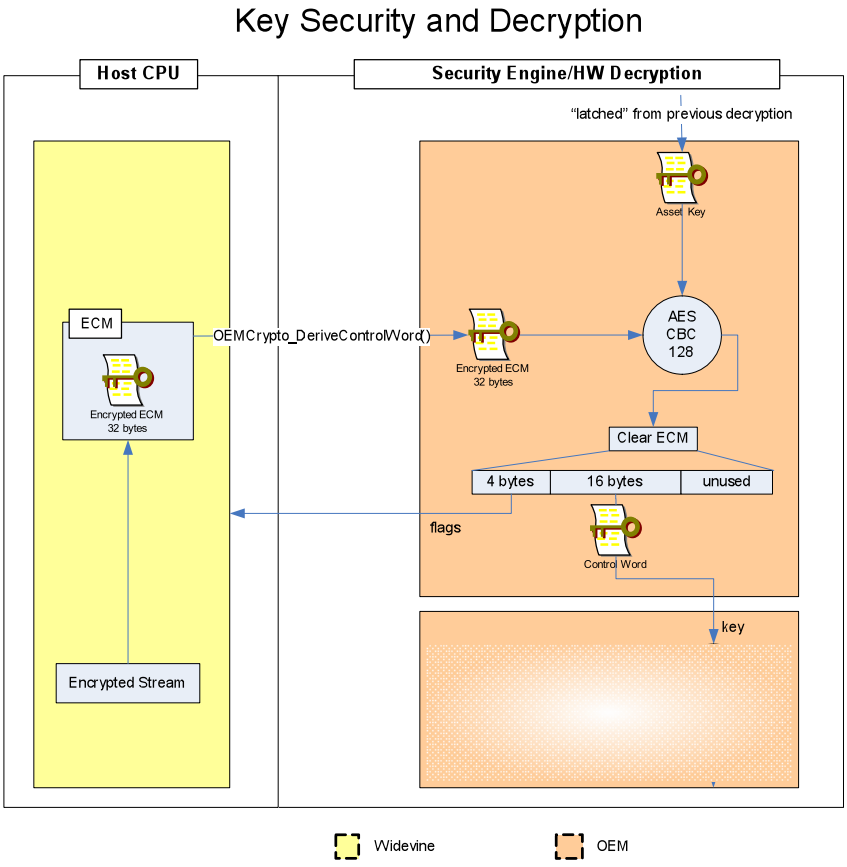


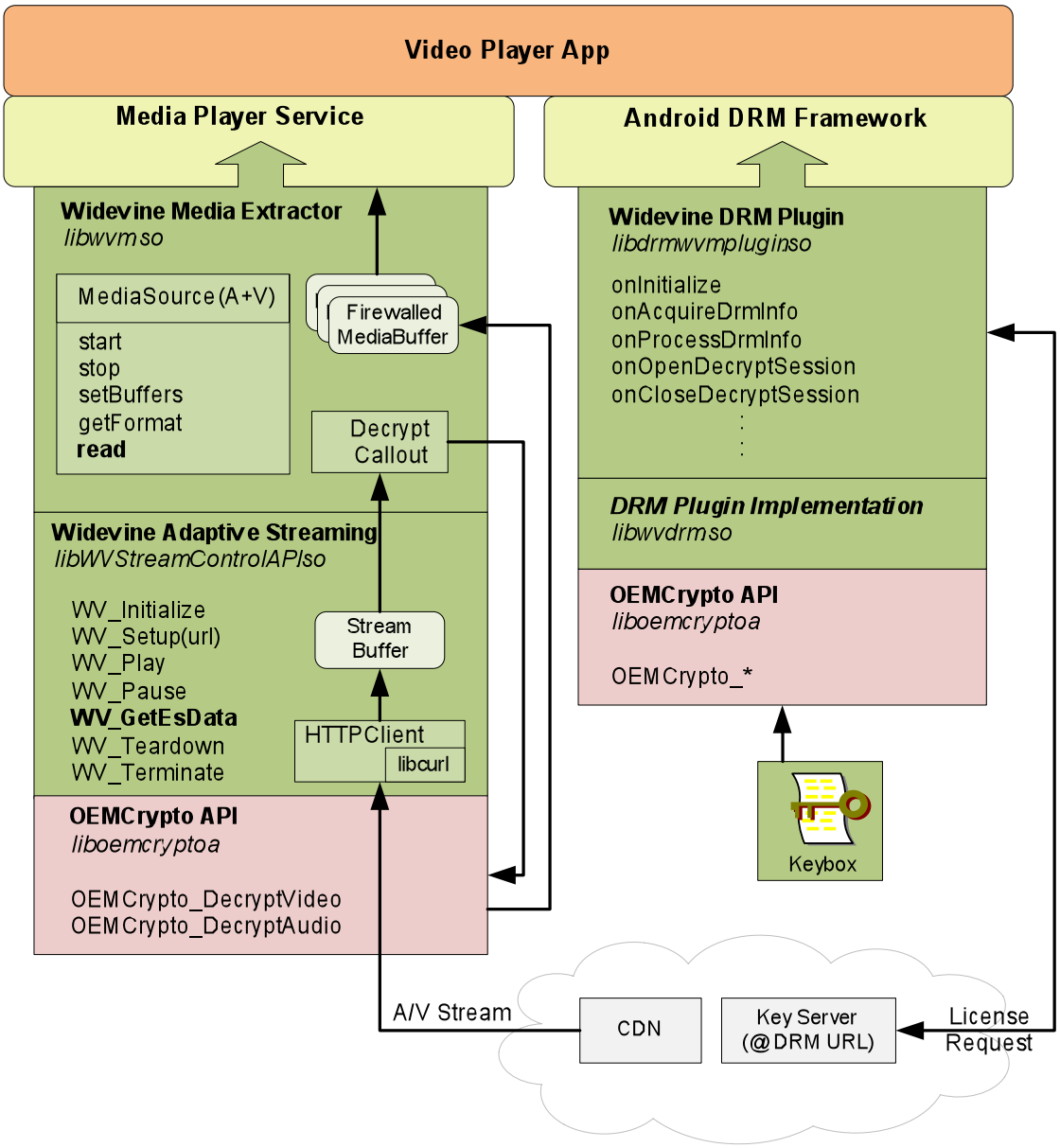
FIGURE 6 - DERIVE CONTROL WORD

6.3 Video Path API

Devices that implement the Key Ladder API must also implement the Video Path API. The video path may be secured either by hardware firewalling of the decrypted data buffers, or by implementing the decrypt operation in the decoder. Either approach requires the device to provide support for AES-128 CBC, ECB and AES-128 CBC-CTS modes.

6.3.1 Firewalled Buffers

A Level 1 security implementation may use buffer firewalling to secure the video path. The data flow through the Widevine Media Extractor and DRM plugin for a Firewalled buffer implementation is shown in the diagram below.



In a firewalled buffer configuration, the OEM implements the decryption in OEMCrypto_DecryptAudio and OEMCrypto_DecryptVideo. For compatibility with existing software codecs, Audio is decrypted into non-firewalled buffers. The video stream is decrypted into firewalled buffers. Decryption is performed using AES-128 bit CBC with CipherText Stealing (CTS) in hardware. If hardware does not support AES-128-CBC-CTS, the OEM must implement it in the trusted environment using either CBC+ECB primitives or software decrypt inside TrustZone.

6.3.2 Decrypt in Decoder

An alternative approach to protecting the video path is for the OEM to implement decryption in the decoders. In this implementation, the data is not decrypted using OEMCrypto_DecryptVideo or OEMCrypto_DecryptAudio. Instead, these functions may attach some metadata or headers to the buffers to indicate which buffers the decoder should decrypt, and the returned buffers are left encrypted. Since the buffers remain encrypted as they pass through the media player, firewalling is not required.

The following table shows the APIs required by each security level:

Crypto Device Control API	Level 1	Level 2	Level 3
OEMCrypto_DecryptVideo	✓		
OEMCrypto_DecryptAudio	✓		
OEMCrypto_Decrypt		✓	

In a Level 2 security implementation where the video path is not protected, the audio and video streams are decrypted using OEMCrypto_Decrypt and buffers are returned to the media player in the clear.

6.3.3 OEMCrypto_DecryptVideo

API Levels	1	~~2~~	~~3~~

The API decrypts (AES-128-CBC) the video payload in the buffer referenced by the *input parameter into the secure buffer referenced by the output parameter, using the control word “latched” in the active hardware key ladder. If inputLength is not a multiple of the crypto block size (16 bytes), the API handles the residual bytes using CipherText Stealing (CTS).

```
API:

OEMCryptoResult OEMCrypto_DecryptVideo(OEMCrypto_UINT8* iv,
      const OEMCrypto_UINT8* input, const OEMCrypto_UINT32 inputLength,
      OEMCrypto_UINT32 outputHandle, OEMCrypto_UINT32 outputOffset,
      OEMCrypto_UINT32 *outputLength);

Parameters:
   •  iv (in/out) - If iv is NULL, then no decryption is required, i.e. the
      packets are already clear.  Otherwise, iv references the AES
      initialization vector. Note that the updated IV after processing the
      final crypto block must be passed back out in *iv.
   •  input (in) - buffer containing the encrypted video data
   •  inputLength (in) - number of bytes in the input payload, which may not
      be a multiple of 16 bytes
   •  outputHandle (in) – reference to the secure buffer which will receive
      the decrypted data
   •  outputOffset (in) – offset from the beginning of the secure buffer
      where the decrypted data will be written
   •  outputLength (out) - number of bytes written into the secure buffer
```

```
Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_DECRYPT_FAILED failed decryption
```

6.3.4 OEMCrypto_DecryptAudio

API Levels	1	~~2~~	~~3~~

The API decrypts (AES-128-CBC) the audio payload in the buffer referenced by the *input parameter into the non-secure buffer referenced by the output parameter, using the control word “latched” in the active hardware key ladder. If inputLength is not a multiple of the crypto block size (16 bytes), the API handles the residual bytes using CipherText Stealing (CTS).

OEMCrypto_DecryptAudio must make sure that it cannot be used to decrypt a video stream into non-firewalled buffers, by verifying that no video packets are processed. This can be done by excluding packets that start with the start code byte sequence {0x00 0x00 0x00 0x01} or {0x00 0x00 0x01}.

```
API:

OEMCryptoResult OEMCrypto_DecryptAudio(
      OEMCrypto_UINT8* iv,
      const OEMCrypto_UINT8* input, const OEMCrypto_UINT32 inputLength,
      OEMCrypto_UINT8 *output, OEMCrypto_UINT32 *outputLength);

Parameters:
   •  iv (in/out) - If iv is NULL, then no decryption is required, i.e. the
      packets are already clear. Otherwise, iv references the AES
      initialization vector. The value contained in a non-NULL iv should be
      ignored and an initialization vector of all zeros should be used in its
      place. This is to prevent video data from being decrypted through the
      audio interface in certain cases.  Note that the updated IV after
      processing the final crypto block must be passed back out in *iv.
   •  input (in) - buffer containing the encrypted audio data
   •  inputLength (in) - number of bytes in the input payload, which may not
      be a multiple of 16 bytes
   •  output (in) – reference to the non-secure buffer which will receive the
      decrypted data
   •  outputLength (out) - number of bytes written into the non-secure buffer

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_DECRYPT_FAILED failed decryption
```

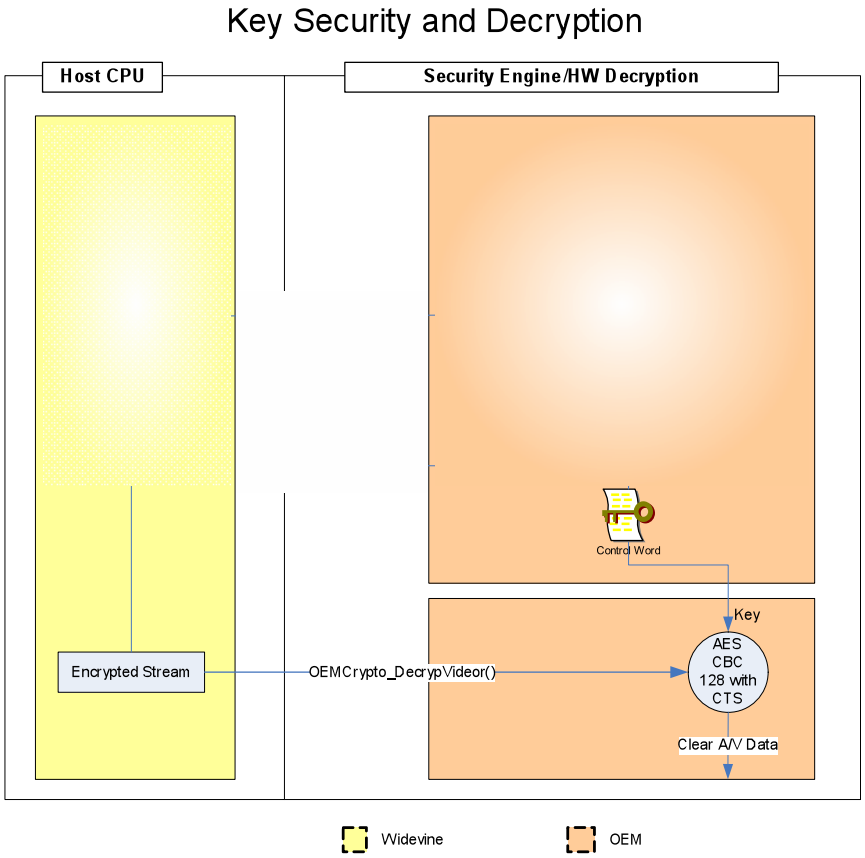


FIGURE 7 - DECRYPTS PAYLOAD USING AES-128 BIT CBC WITH CTS

6.3.5 OEMCrypto_Decrypt

API Levels	~~1~~	2	~~3~~

The API decrypts (AES-128-CBC-CTS) the buffer referenced by the *input parameter into the secure buffer referenced by the output parameter, using the control word “latched” in the active hardware key ladder. If inputLength is not a multiple of the crypto block size (16 bytes), the API handles the residual bytes using CipherText Stealing (CTS).

```
API:

OEMCryptoResult OEMCrypto_Decrypt(OEMCrypto_UINT8 *input,
                                  OEMCrypto_UINT8 *output,
                                  const OEMCrypto_UINT32 inputLength,
                                  const OEMCrypto_UINT8 initIvFlag);

Parameters:
   •  (in)  input: source (encrypted) payload
   •  (out) output: buffer to return clear payload. The size of the output
      buffer must be equal to or larger than length.
   •  (in) inputLength: length of the source (encrypted) payload
```

- (in) initIvFlag: flag which means initializing IV=0 is required or not (1=required, 0=continue)

```
Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_DECRYPT_FAILED failed decryption
```

6.4 Provisioning API

Widevine Keyboxes are installed on a device to establish a root of trust, which is used to secure content on a device. This section describes the APIs that install the Widevine Keybox.

Since the methods for provisioning a device are related to the specific methods of manufacturing, several options are available. This section describes some recommended methods.

API functions marked as optional may be implemented in the library, but are not called from the Widevine DRM Plugin during normal operation. They may be used by the OEM's factory provisioning procedure.

The following table shows the APIs for each security level:

Crypto Device Control API	Level 1	Level 2	Level 3
OEMCrypto_InstallKeybox	✓	✓	
OEMCrypto_WrapKeybox	✓ *optional*	✓ *optional*	
OEMCrypto_EncryptAndStoreKeybox	✓ *optional*	✓ *optional*	✓

6.4.1 OEMCrypto_WrapKeybox

API Levels	✓ *optional*	✓ *optional*	~~3~~

During manufacturing, the keybox should be encrypted with the OEM root key and stored on the file system in a region that will not be erased during factory reset. As described in section 5.4.2, the keybox may be directly encrypted and stored on the device in a single step, or it may use the two-step WrapKeybox/InstallKeybox approach. When the Widevine DRM plugin initializes, it will look for a wrapped keybox in the file /factory/wv.keys and install it into the security processor by calling OEMCrypto_InstallKeybox.

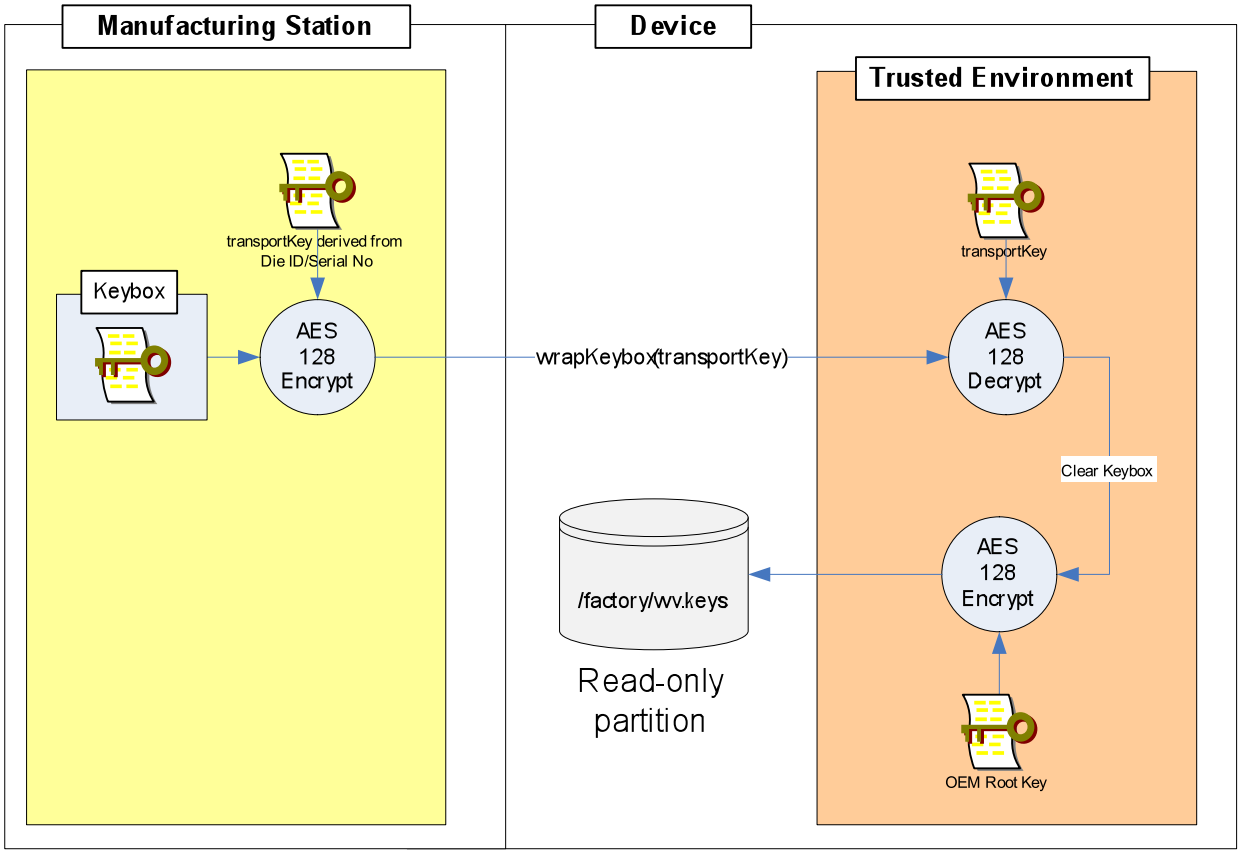


FIGURE 8. OEMCRYPTO_WRAPKEYBOX OPERATION

OEMCrypto_WrapKeybox is used to generate an OEM-encrypted keybox that may be passed to OEMCrypto_InstallKeybox for provisioning. The keybox may be either passed in the clear or previously encrypted with a transport key. If a transport key is supplied, the keybox is first decrypted with the transport key before being wrapped with the OEM root key.

```
API: OEMCryptoResult OEMCrypto_WrapKeybox(
       OEMCrypto_UINT8 *keybox,
       OEMCrypto_UINT32 keyboxLength,
       OEMCrypto_UINT8 *wrappedKeybox,
       OEMCrypto_UINT32 *wrappedKeyBoxLength,
       OEMCrypto_UINT8 *transportKey
       OEMCrypto_UINT32 transportKeyLength);

Parameters:
   • keybox (in) - pointer to Keybox data to encrypt.  May be NULL on the
     first call to test size of wrapped keybox. The keybox may either be
     clear or previously encrypted.
   • keyboxLength (in) - length the keybox data in bytes
   • wrappedKeybox (out) – Pointer to wrapped keybox
```

- wrappedKeyboxLength (out) – Pointer to the length of the wrapped keybox in bytes
- transportKey (in) – An optional AES transport key. If provided, the keybox parameter was previously encrypted with this key. The keybox will be decrypted with the transport key using AES-CBC and a null IV.
- transportKeyLength – number of bytes in the transportKey

```
Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_WRITE_KEYBOX failed to encrypt the keybox
      OEMCrypto_ERROR_SHORT_BUFFER if keybox is provided as NULL, to
      determine the size of the wrapped keybox
```

6.4.2 OEMCrypto_InstallKeybox

API Levels	1	2	~~3~~

Decrypt a wrapped keybox and install it in the security processor. The keybox is unwrapped then encrypted with the OEM root key. This function is called from the Widevine DRM plugin at initialization time if there is no valid keybox installed. It looks for a wrapped keybox in the file /factory/wv.keys and if it is present, will read the file and call OEMCrypto_InstallKeybox with the contents of the file.

```
API: OEMCryptoResult OEMCrypto_InstallKeybox(
         OEMCrypto_UINT8 *keybox, OEMCrypto_UINT32 keyboxLength);

Parameters:
   •  keybox (in) - pointer to encrypted Keybox data as input
   •  keyboxLength (in) - length of the keybox data in bytes

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_WRITE_KEYBOX failed to encrypt and store Keybox
```

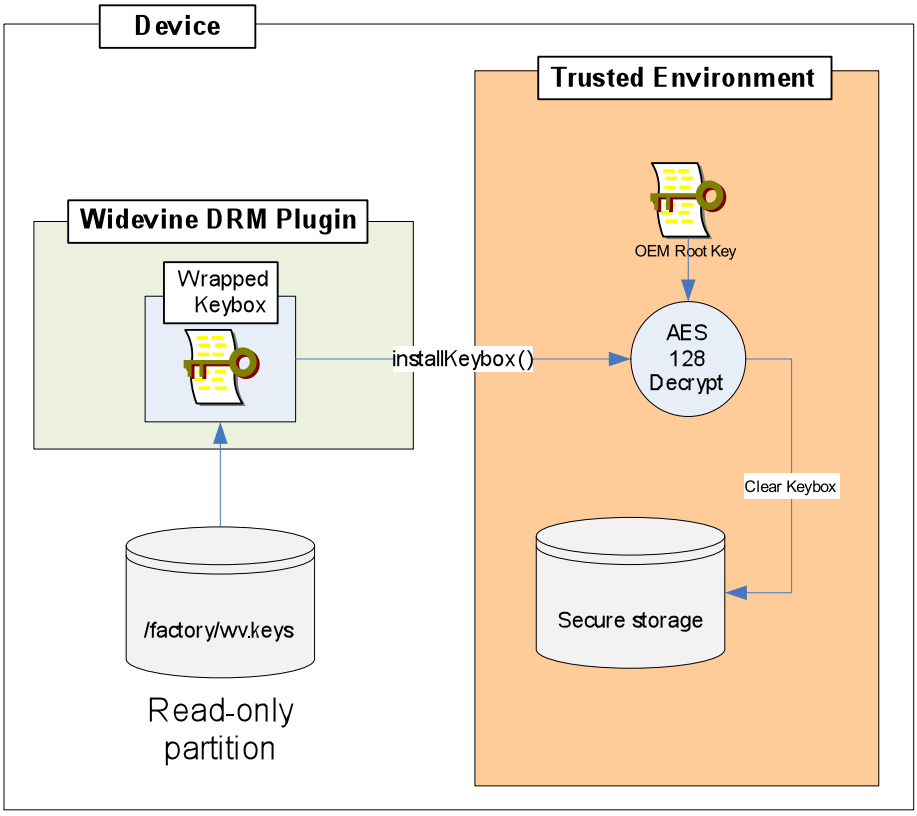


FIGURE 9 - INSTALL KEYBOX OPERATION

6.4.3 OEMCrypto_EncryptAndStoreKeyBox

API Levels	✓ *optional*	✓ *optional*	3

Encrypt and store the keybox to persistent memory. The device key or entire keybox must be stored securely, encrypted by an OEM root key.

This function is used once to save the keybox onto the device at provisioning time. It is an alternate provisioning method as compared to the WrapKeybox/InstallKeybox method.

When running at security level 3, the device should implement a secure boot mechanism that disables access to the provisioned keybox when the device is unlocked ("fastboot oem unlock"). The OEMCrypto_EncryptAndStoreKeyBox API should also fail the call and not store the field-provisioned keybox in the unlock state. These measures prevent an unsigned system image from accessing the keybox.

Widevine security level 1 devices do not require a signed system image to be loaded, they do require a signed secure OS image to be loaded by the sec.boot ldr. The

OEMCrypto_Encrypt&StoreKeyBox must not fail the call if the bootloader is in an unlocked state.

```
OEMCryptoResult
OEMCrypto_EncryptAndStoreKeyBox(OEMCrypto_UINT8 *keybox,
                                OEMCrypto_UINT32 keyBoxLength)
```

Parameters:

- keybox (in) - Pointer to clear keybox data. Must be encrypted with an OEM root key.
- keyboxLength (in) - Length of the keybox data in bytes

Returns:

OEMCryptoResult indicating success or failure

6.5 Keybox Access API

Widevine Keyboxes are installed on a device to establish a root of trust, which is used to secure content on a device. This section describes the APIs that allow the security processor or general CPU to access the Widevine Keybox, depending on the security level. In a Level 1 or Level 2 implementation, only the security processor may access the keys in the keybox. In a Level 3 implementation, functions are provided to allow the CPU to access the keys to perform decryption.

The following table shows the APIs required by each security level:

Crypto Device Control API	Level 1	Level 2	Level 3
OEMCrypto_IsKeyboxValid	✓	✓	
OEMCrypto_GetDeviceId	✓	✓	
OEMCrypto_IdentifyDevice			✓
OEMCrypto_GetKeyData	✓	✓	
OEMCrypto_GetKeyboxData			✓
OEMCrypto_GetRandom	✓	✓	✓

6.5.1 OEMCrypto_IsKeyboxValid

API Levels	1	2	~~3~~

The API validates the Widevine Keybox loaded into the security processor device.

The API performs two verification steps on the Keybox. It first verifies the MAGIC field contains a valid signature (i.e. ‘k’’b’’o’’x’). The API then computes the CRC using CRC-32-IEEE 802.3 standard and compares the checksum to the CRC stored in the Keybox.

The CRC is computed over the entire Keybox excluding the 4 bytes CRC (i.e. Keybox[0..123].

Field	Description	Size (bytes)
Device ID	C character string identifying the device, null terminated.	32
Device Key	128 bit AES key assigned to device, generated by Widevine.	16
Key Data	Encrypted data	72
Magic	Constant code used to recognize a valid keybox: “kbox” (0x6b626f78) API verifies Keybox contains a valid signature	4
CRC	CRC-32 Posix-1003.2 validates integrity of the Keybox data; the CRC is computed over Device ID field through Magic field (i.e. Keybox[0..123]) API computes CRC on all Keybox data excluding this 4 bytes CRC; the result is compared with this 4 bytes	4
	Total Size	***128***

```
API: OEMCryptoResult OEMCrypto_IsKeyboxValid();

Parameters:
   •  none

Returns:
      OEMCrypto_SUCCESS
      OEMCrypto_ERROR_BAD_MAGIC
      OEMCrypto_ERROR_BAD_CRC
```

6.5.2 OEMCrypto_GetDeviceID

API Levels	1	2	~~3~~

Retrieve the device's unique identifier from the Keybox.

Field	Description	Size (bytes)
Device ID	C character string identifying the device, null terminated. API returns this null terminated string	32
Device Key	128 bit AES key assigned to device, generated by Widevine.	16
Key Data	Encrypted data	72
Magic	Constant code used to recognize a valid keybox: “kbox” (0x6b626f78)	4
CRC	CRC-32-IEEE 802.3 validates integrity of the Keybox data; the CRC is computed over Device ID field through Magic field (i.e. Keybox[0..123])	4
	Total Size	***128***

```
API: OEMCryptoResult OEMCrypto_GetDeviceID(
         OEMCrypto_UINT8* deviceID, OEMCrypto_UINT32 *idLength);

Parameters:
   •  deviceId (out) - pointer to the buffer that receives the Device ID
   •  idLength (in/out) – on input, size of the caller’s device ID buffer. On
      output, the number of bytes written into the buffer.

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_SHORT_BUFFER if the buffer is too small to return
      device ID
      OEMCrypto_ERROR_NO_DEVICEID failed to return Device Id
```

6.5.3 OEMCrypto_IdentifyDevice

API Levels	~~1~~	~~2~~	3

Return the device's unique identifier. The device identifier shall not come from the Widevine keybox.

```
OEMCryptoResult OEMCrypto_IdentifyDevice(OEMCrypto_UINT8* deviceID,
                                         OEMCrypto_UINT32 idLength)
```

Parameters:

- deviceID (out) - Points to the buffer that should recieve the key data.
- idLength (in) - Length of the device ID buffer. Maximum of 32 bytes allowed.

Returns:

OEMCryptoResult indicating success or failure

6.5.4 OEMCrypto_GetKeyData

API Levels	1	2	~~3~~

The API returns the Key Data field from the Keybox.

This function should return the clear Key Data field from the Widevine keybox

Field	Description	Size (bytes)
Device ID	C character string identifying the device, null terminated.	32
Device Key	**128 bit AES key assigned to device, generated by Widevine (API encrypts Device Key using an OEM root key)**	**16**
Key Data	API decrypts returns Key Data in *keyData	72
Magic	Constant code used to recognize a valid keybox: “kbox” (0x6b626f78)	4
CRC	CRC-32-IEEE 802.3 validates integrity of the Keybox data; the CRC is computed over Device ID field through Magic field (i.e. Keybox[0..123])	4
	Total Size	***128***

```
API: OEMCryptoResult OEMCrypto_GetKeyData(
         OEMCrypto_UINT8* keyData, OEMCrypto_UINT32 *keyDataLength);

Parameters:
   • keyData (out) - pointer to the buffer to hold the Key Data field from
     the Keybox
   • keyDataLength (in/out) – on input, the allocated buffer size.  On
     output, the number of bytes in Key Data

Returns:
      OEMCrypto_SUCCESS success
```

```
OEMCrypto_ERROR_SHORT_BUFFER if the buffer is too small to return
KeyData
OEMCrypto_ERROR_NO_KEYDATA failed to return KeyData
```

6.5.5 OEMCrypto_GetKeyboxData

API Levels	~~1~~	~~2~~	3

Retrieve a range of bytes from the Widevine keybox. This function should decrypt the keybox and return the specified bytes.

```
OEMCryptoResult OEMCrypto_GetKeyboxData(OEMCrypto_UINT8* buffer,
          OEMCrypto_UINT32 offset, OEMCrypto_UINT32 length)
```

Parameters:

- buffer (out) - Points to the buffer that should recieve the keybox data.
- offset (in) – Byte offset from the beginning of the keybox of the first byte to return
- length (in) – Number of bytes of data to return.

Returns:

OEMCryptoResult indicating success or failure

6.5.6 OEMCrypto_GetRandom

API Levels	1	2	3

Returns a buffer filled with hardware-generated random bytes, if supported by the hardware.

```
API: OEMCryptoResult OEMCrypto_GetRandom(
         OEMCrypto_UINT8* randomData, OEMCrypto_UINT32 dataLength);

Parameters:
   •  randomData (out) - pointer to the buffer that receives random data
   •  dataLength (in) - length of the random data buffer in bytes

Returns:
      OEMCrypto_SUCCESS success
      OEMCrypto_ERROR_RNG_FAILED failed to generate random number
      OEMCrypto_ERROR_RNG_NOT_SUPPORTED function not supported
```

7 Deliverables

The difference between Widevine Security Level 1 (L1) and Level 2 (L2) is the protected video path, which is done in the SoC/vendor integration and does not affect the libraries provided by Widevine. Therefore, only L1 and L3 libraries are provided for libWVStreamControlAPI*.so and libwvdrm*.so. Integrators may implement L2 using the L1 libraries, and not providing a protected video path. However, it is recommended that new device designs implement Widevine Level 1 security.

7.1 For Honeycomb (Android 3.x Releases)

The OEM should implement the functions defined in the APIs section in a static library called liboemcrypto.a. The static library will be linked into the Widevine DRM plugin for the device during the android build. The binary library should be included in the OEM's vendor branch of the android tree. The vendor's source code to implement OEMCrypto functions and the OEMCrypto.h header should not be checked into the android source tree.

To avoid library name conflicts in the build system, the vendor's Android.mk should conditionally add liboemcrypto.a to the LOCAL_PREBUILT_LIBS based on the target device:

```
ifeq ($(TARGET_DEVICE),xxx)

# oem security lib for Widevine drm provided by oem
LOCAL_STATIC_LIBRARIES += \
   liboemcrypto

include $(BUILD_MULTI_PREBUILT)

LOCAL_SHARED_LIBRARIES += \
   lib1             \
   lib2
else
# for devices that don't support WV drm liboemcrypto.a is not provided by the vendor
# Use liboemstub instead
LOCAL_STATIC_LIBRARIES += \
   liboemstub
endif

ifeq($(TARGET_DEVICE),xxx)
where xxx is the target device.
```

In addition, add the following lines to <vendor-root>/product.mk file:

```
libdrmwvmplugin \
com.google.widevine.software.drm.xml \
libwvm \
libWVStreamControlAPI.so \
libwvdrm.so \
```

7.2 For Ice Cream Sandwich (Android 4.0.x Releases)

The OEM should implement the functions defined in the APIs section in a static library called liboemcrypto.a. The static library will be linked into the Widevine DRM plugin for the device during the android build. The binary library should be included in the OEM's vendor branch of the android tree. The vendor's source code to implement OEMCrypto functions and the OEMCrypto.h header should not be checked into the android source tree.

To avoid library name conflicts in the build system, the vendor's Android.mk should conditionally add liboemcrypto.a to the BUILD_MULTI_PREBUILT based on the target device.

Step 1: modify vendor/manufacturer/device/liboemcrypto/Android.mk file as indicated in red; replace manufacturer and device marked in red with the actual names:

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

ifeq ($(TARGET_ARCH),arm)
ifneq (,$(filter device, $(TARGET_DEVICE)))

##########################################################################
# liboemcrypto.a, lib1

LOCAL_PREBUILT_LIBS := liboemcrypto.a
LOCAL_MODULE_TAGS := optional
include $(BUILD_MULTI_PREBUILT)

##########################################################################
# libdrmwvmplugin.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/drmwvmplugin/plugin-core.mk

LOCAL_MODULE := libdrmwvmplugin
LOCAL_MODULE_PATH := $(TARGET_OUT_VENDOR_SHARED_LIBRARIES)/drm
```

```
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto lib1
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

##########################################################################
# libwvm.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/wvm/wvm-core.mk

LOCAL_MODULE := libwvm
LOCAL_PROPRIETARY_MODULE := true
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto lib1
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

endif
endif
```

Note: lib1 is an example of a vendor supplied static library to support memory access control (hardware firewall).

Step 2: add the following lines to vendor/google/products/device_common.mk file; replace <device>_common.mk marked in red with your device name:

For Widevine Streaming Security Model 1 and 2:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
    com.google.widevine.software.drm \
    libdrmwvmplugin libwvm libWVStreamControlAPI_L1 libwvdrm_L1 \
    Additional drivers to support hardware firewall …
```

For Widevine Streaming Security Model 3:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
    com.google.widevine.software.drm \
    libdrmwvmplugin libwvm libWVStreamControlAPI_L3 libwvdrm_L3
```

7.3 For JellyBean (Android 4.1 - 4.3)

The OEM should implement the functions defined in the APIs section in a static library called liboemcrypto.a. The static library will be linked into the Widevine DRM plugin for the device during the android build. The binary library should be included in the OEM's vendor branch of the android tree. The vendor's source code to implement OEMCrypto functions and the OEMCrypto.h header should not be checked into the android source tree.

To avoid library name conflicts in the build system, the vendor's Android.mk should conditionally add liboemcrypto.a to the BUILD_MULTI_PREBUILT based on the target device.

Builds for JellyBean add a new library called libdrmdecrypt.so that is used to support Media Codec mode.

Step 1: modify vendor/manufacturer/device/liboemcrypto/Android.mk file as indicated in red; replace manufacturer and device marked in red with the actual names:

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

ifeq ($(TARGET_ARCH),arm)
ifneq (,$(filter device, $(TARGET_DEVICE)))

##########################################################################
# liboemcrypto.a, lib1

LOCAL_PREBUILT_LIBS := liboemcrypto.a
LOCAL_MODULE_TAGS := optional
include $(BUILD_MULTI_PREBUILT)

##########################################################################
# libdrmwvmplugin.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/drmwvmplugin/plugin-core.mk

LOCAL_MODULE := libdrmwvmplugin
LOCAL_MODULE_PATH := $(TARGET_OUT_VENDOR_SHARED_LIBRARIES)/drm
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto lib1
LOCAL_PRELINK_MODULE := false
```

```
include $(BUILD_SHARED_LIBRARY)

##########################################################################
# libwvm.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/wvm/wvm-core.mk

LOCAL_MODULE := libwvm
LOCAL_PROPRIETARY_MODULE := true
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto lib1
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

endif
endif
```

Note: lib1 is an example of a vendor supplied static library to support memory access control (hardware firewall).

Step 2: modify vendor/manufacturer/device/libdrmdecrypt/Android.mk file as indicated; replace manufacturer and device marked in red with the actual names:

```
# libdrmdecrypt.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/cryptoPlugin/decrypt-core.mk

LOCAL_C_INCLUDES := \
   $(TOP)/frameworks/native/include/media/hardware \
   $(TOP)/vendor/widevine/proprietary/cryptoPlugin

LOCAL_SHARED_LIBRARIES := \
   libstagefright_foundation \
   liblog

LOCAL_MODULE := libdrmdecrypt
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)
```

Step 3: add the following lines to vendor/google/products/device_common.mk file; replace <device>_common.mk marked in red with your device name:

For Widevine Streaming Security Model 1 and 2:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
      com.google.widevine.software.drm \
      libdrmwvmplugin libwvm libWVStreamControlAPI_L1 libwvdrm_L1 \
      Additional drivers to support hardware firewall …
```

For Widevine Streaming Security Model 3:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
      com.google.widevine.software.drm \
      libdrmwvmplugin libwvm libWVStreamControlAPI_L3 libwvdrm_L3
```

7.4 For KitKat (Android 4.4.x)

The OEM should implement the functions defined in the APIs section in a static library called liboemcrypto_L1.a (or liboemcrypto_L3.a). The static library will be linked into the Widevine DRM plugin for the device during the android build. The binary library should be included in the OEM's vendor branch of the android tree. The vendor's source code to implement OEMCrypto functions and the OEMCrypto.h header should not be checked into the android source tree.

To avoid library name conflicts in the build system, the vendor's Android.mk should conditionally add liboemcrypto_L1.a to the BUILD_MULTI_PREBUILT based on the target device.

Step 1: modify vendor/manufacturer/device/liboemcrypto/Android.mk file as indicated in red; replace manufacturer, device and chipset marked in red with the actual names:

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

ifeq ($(TARGET_ARCH),arm)
ifneq (,$(filter device, $(TARGET_DEVICE)))

##########################################################################
# libdrmwvmplugin.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/drmwvmplugin/plugin-core.mk
```

```
LOCAL_MODULE := libdrmwvmplugin
LOCAL_MODULE_PATH := $(TARGET_OUT_VENDOR_SHARED_LIBRARIES)/drm
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto_L1 lib1
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

##########################################################################
# libwvm.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/wvm/wvm-core.mk

LOCAL_MODULE := libwvm
LOCAL_PROPRIETARY_MODULE := true
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto_L1 lib1
LOCAL_PRELINK_MODULE := false
include $(BUILD_SHARED_LIBRARY)

endif
endif
```

Note1: liboemcrypto_L1.a is provided by vendor and resides in vendor/manufacturer/device/proprietary/prebuilt/target/chipset/obj/STATIC_LIBRARIES/liboemcrypto_L1_intermediates/

Note2: lib1 is an example of a vendor supplied static library to support memory access control (hardware firewall).

Step 2: modify vendor/manufacturer/device/libdrmdecrypt/Android.mk file as indicated; replace manufacturer and device marked in red with the actual names:

```
# libdrmdecrypt.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/cryptoPlugin/decrypt-core.mk

LOCAL_C_INCLUDES := \
    $(TOP)/frameworks/native/include/media/hardware \
    $(TOP)/vendor/widevine/proprietary/cryptoPlugin

LOCAL_SHARED_LIBRARIES := \
    libstagefright_foundation \
```

liblog

```
LOCAL_MODULE := libdrmdecrypt
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)
```

Step 3: add the following lines to vendor/google/products/device_common.mk file; replace <device>_common.mk marked in red with your device name:

For Widevine Streaming Security Model 1 and 2:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
    com.google.widevine.software.drm \
    libdrmwvmplugin libwvm libWVStreamControlAPI_L1 libwvdrm_L1 \
    Additional drivers to support hardware firewall …
```

For Widevine Streaming Security Model 3:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
    com.google.widevine.software.drm \
    libdrmwvmplugin libwvm libWVStreamControlAPI_L3 libwvdrm_L3
```

7.5 For L (Android 5.x)

The OEM should implement the functions defined in the APIs section in a static library called liboemcrypto_L1.a (or liboemcrypto_L3.a). The static library will be linked into the Widevine DRM plugin for the device during the android build. The binary library should be included in the OEM's vendor branch of the android tree. The vendor's source code to implement OEMCrypto functions and the OEMCrypto.h header should not be checked into the android source tree.

To avoid library name conflicts in the build system, the vendor's Android.mk should conditionally add liboemcrypto_L1.a to the BUILD_MULTI_PREBUILT based on the target device.

Step 1: modify vendor/manufacturer/device/liboemcrypto/Android.mk file as indicated in red; replace manufacturer, device and chipset marked in red with the actual names:

```
LOCAL_PATH:= $(call my-dir)
```

```
include $(CLEAR_VARS)

ifeq ($(TARGET_ARCH),arm)
ifneq (,$(filter device, $(TARGET_DEVICE)))

##########################################################################
# libdrmwvmplugin.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/drmwvmplugin/plugin-core.mk

LOCAL_MODULE := libdrmwvmplugin
LOCAL_MODULE_PATH := $(TARGET_OUT_VENDOR_SHARED_LIBRARIES)/drm
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto_L1 lib1
LOCAL_PRELINK_MODULE := false

# Android L release supports both 32 bit and 64 bit architecture. Since mediaserver and
# drmserver both run as 32 bit, set this build variable to build 32 bit Widevine libraries.
LOCAL_MULTILIB := 32

include $(BUILD_SHARED_LIBRARY)

##########################################################################
# libwvm.so

include $(CLEAR_VARS)
-include $(TOP)/vendor/widevine/proprietary/wvm/wvm-core.mk

LOCAL_MODULE := libwvm
LOCAL_PROPRIETARY_MODULE := true
LOCAL_MODULE_TAGS := optional
LOCAL_STATIC_LIBRARIES += liboemcrypto_L1 lib1
LOCAL_PRELINK_MODULE := false
LOCAL_MULTILIB := 32

include $(BUILD_SHARED_LIBRARY)

endif
endif
```

Note1: liboemcrypto_L1.a is provided by vendor and resides in vendor/manufacturer/device/proprietary/prebuilt/target/chipset/obj/STATIC_LIBRARIES/liboemcrypto_L1_intermediates/

Note2: lib1 is an example of a vendor supplied static library to support memory access control (hardware firewall).

Step 2: add the following lines to vendor/google/products/device_common.mk file; replace <device>_common.mk marked in red with your device name:

For Widevine Streaming Security Model 1 and 2:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
    com.google.widevine.software.drm \
    libdrmwvmplugin libwvm libWVStreamControlAPI_L1 libwvdrm_L1 \
    Additional drivers to support hardware firewall …
```

For Widevine Streaming Security Model 3:

```
#enable Widevine drm
PRODUCT_PROPERTY_OVERRIDES += drm.service.enabled=true
PRODUCT_PACKAGES += com.google.widevine.software.drm.xml \
    com.google.widevine.software.drm \
    libdrmwvmplugin libwvm libWVStreamControlAPI_L3 libwvdrm_L3
```

7.6 Configure Widevine Stream Cache Size for HD content in Android 4.2 (Jelly Bean MR1 release) and newer Android versions.

The recommended minimum cache size for devices that support High Definition (1080p) content and have sufficient memory is 16MB. Vendors can configure the cache size using the read only property: ro.com.widevine.cachesize.

In the vendor's Android makefile, edit the following line:

```
PRODUCT_PROPERTY_OVERRIDES += ro.com.widevine.cachesize=16777216
```

If ro.com.widevine.cachesize is unspecified, a 10MB default size is used.