WV Modular DRM Version 11 Delta

Changes from Version 10 to 11

Table of Contents

References

DASH - 23001-7 ISO BMFF Common Encryption

DASH - 14496-12 ISO BMFF Amendment

W3C Encrypted Media Extensions (EME)

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

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

Draft International Standard ISO/IEC DIS 23001-7

Audience

This document is intended for SOC and OEM device manufacturers to upgrade an integration with Widevine content protection using Common Encryption (CENC) on consumer devices. In particular, if you already have a working OEMCrypto v10 library, and want to upgrade to OEMCrypto v11, then this document is for you. If you are starting from scratch, you should read WV Modular DRM Security Integration Guide for Common Encryption (CENC).

Overview

There are several new features required for OEMCrypto version 11. The following sections discuss the main new features and give some idea why the new feature is being added. You should refer to WV Modular DRM Security Integration Guide for Common Encryption (CENC) for the full documentation of the API -- this document only discusses changes to the API.

Definitions

CENC - Common Encryption

DASH - Dynamic Adaptive Streaming over HTTP

CDM - Content Decryption Module -- this is the software that calls the OEMCrypto library and implements CENC.

API Version Number

```
uint32_t OEMCrypto_APIVersion();
```

This function should now return 11. If it returns 10, then the calling code will assume that OEMCrypto does not support the new v11 features. Depending on the platform, the library may be run in backwards compatibility mode, or it may fail. For example, OEMCrypto v11 is required on Android N devices.

CENC 3.0 Encryption Modes

Previously, the CDM would only decrypt content that was encrypted using AES CTR (counter) mode. Now, the CDM will decrypt content that has been encrypted with the other modes specified in CENC 3.0. These modes use various subsample patterns with either CTR (counter) mode or CBC (Cipher Block Chaining) mode. To support this, the function OEMCrypto_LoadKeys will be modified so that the CDM code can specify either CTR or CBC mode when the content keys are loaded. Also, the function OEMCrypto_DecryptCTR will be replaced with OEMCrypto_DecryptCENC. The new function has an additional argument that specifies the encryption pattern.

First, the function OEMCrypto_LoadKeys will change because the type of the parameter `key_array` is changing. The struct OEMCrypto_KeyObject will have a new field, cipher_mode. This will be used to tell OEMCrypto that the specified key will use either CTR or CBC mode for this key's decryptions.

```
typedef enum OEMCryptoCipherMode {
    OEMCrypto_CipherMode_CTR,
    OEMCrypto_CipherMode_CBC,
} OEMCryptoCipherMode;
typedef struct {
    ...
   OEMCryptoCipherMode cipher_mode;
} OEMCrypto_KeyObject;
```

LoadKeys may return OEMCrypto_ERROR_NOT_IMPLEMENTED if the mode is not supported. An implementation that does not support both modes is not considered complete.

Second, the decryption function is renamed to indicate that it supports both CTR and CBC modes. It has a new parameter, pattern, that indicates the encrypt/skip pattern for the decryption. Notice that the block_offset pattern is still passed in, but is only used for CTR mode.

```
OEMCryptoResult
OEMCrypto_DecryptCENC(OEMCrypto_SESSION session,
                      const uint8_t *data_addr,
                      size_t data_length,
                      bool is_protected,
                      const uint8_t *iv,
                      size_t block_offset, // used for CTR mode only.
                      const OEMCrypto_DestBufferDesc* out_buffer,
                      const OEMCrypto_CENCEncryptPatternDesc* pattern,
```

```
                        uint8_t subsample_flags);

typedef struct {
   size_t encrypt; // number of 16 byte blocks to decrypt.
   size_t skip;    // number of 16 byte blocks to leave in clear.
   size_t offset;  // offset into the pattern for this call, in blocks.
} OEMCrypto_CENCEncryptPatternDesc;
```

The secure path and HDCP control flags in the key control block should be honored for CBC mode as well as for CTR mode.

Signed OEMCrypto Patch Level Enforcement

This feature addresses the desire of a content provider to serve licenses to a device only if it has a specific security patch. This feature allows the device to indicate that it has received a security patch. Notice that this feature will not distinguish between a device whose root of trust has been compromised and one that has not --- it is assumed that the root of trust is still uncompromised.
This feature will be implemented by assigning a patch level to the OEM software -- either OEMCrypto or any underlying components. Initially the patch level will be 0. The patch level would only be rolled when a security problem has been discovered, and there is a need to distinguish between devices in the field that have the new security patch from those that do not. Since this is expected to happen very rarely, the patch level will be 0 for most devices. The patch level is only used to distinguish between devices with the same Widevine system ID. Devices with different system IDs will not have their patch levels compared.

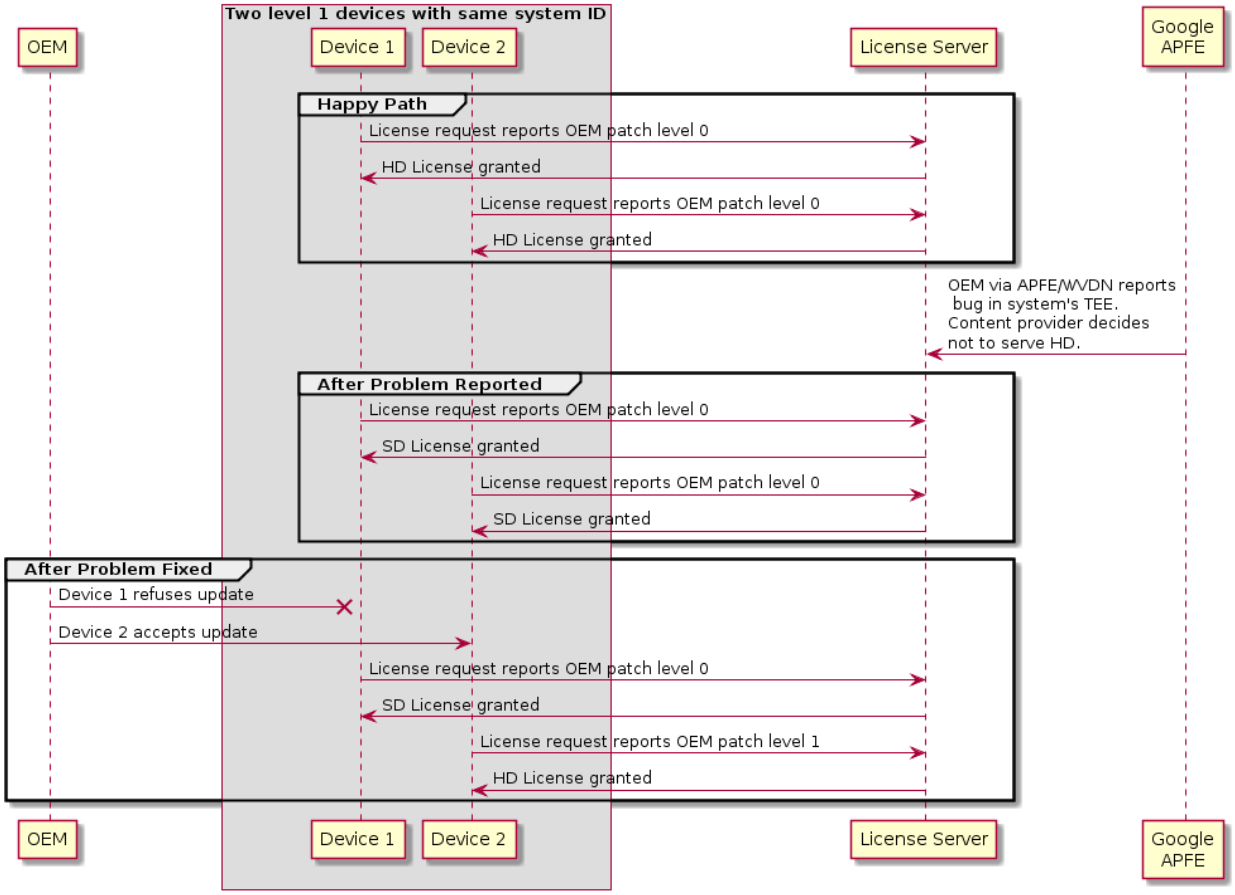
When the device sends a license request to the server, the current OEM patch level is included in the request. The server will decide which type of license to grant, and send the license response. When the function LoadKeys is called, the key control block will have the bits Minimum_Security_Patch_Level set to the patch level. If the minimum number is larger than the current patch level, the device should assume that there has been a man-in-the-middle attack, and reject the license.

New API:

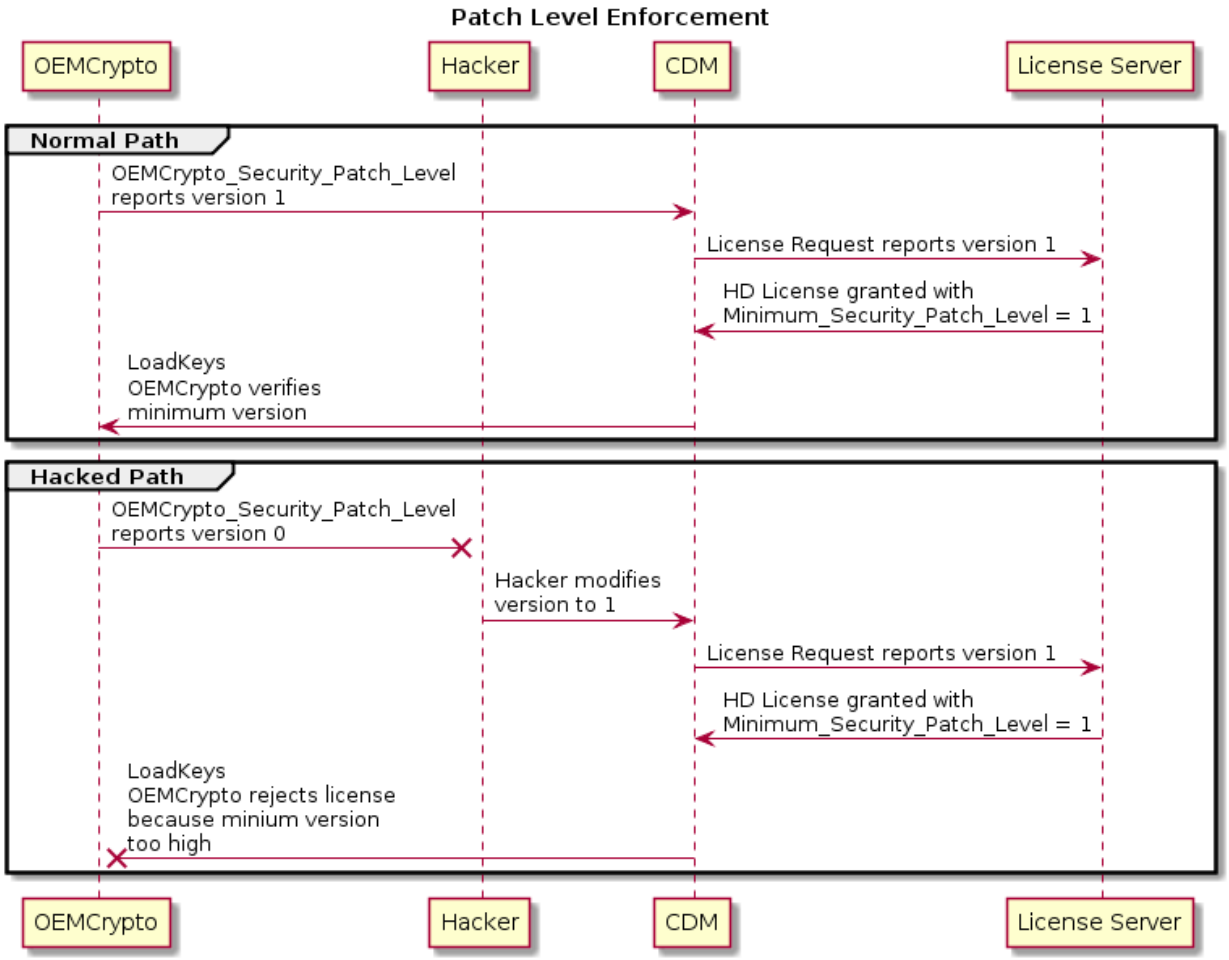
```
uint8_t OEMCrypto_Security_Patch_Level(void)
```

This function should return the current patch level of the OEM/TEE software. If no security updates have been made, this should return 0.

Here is a top level sequence diagram showing two devices. One device is updated and the other is not.

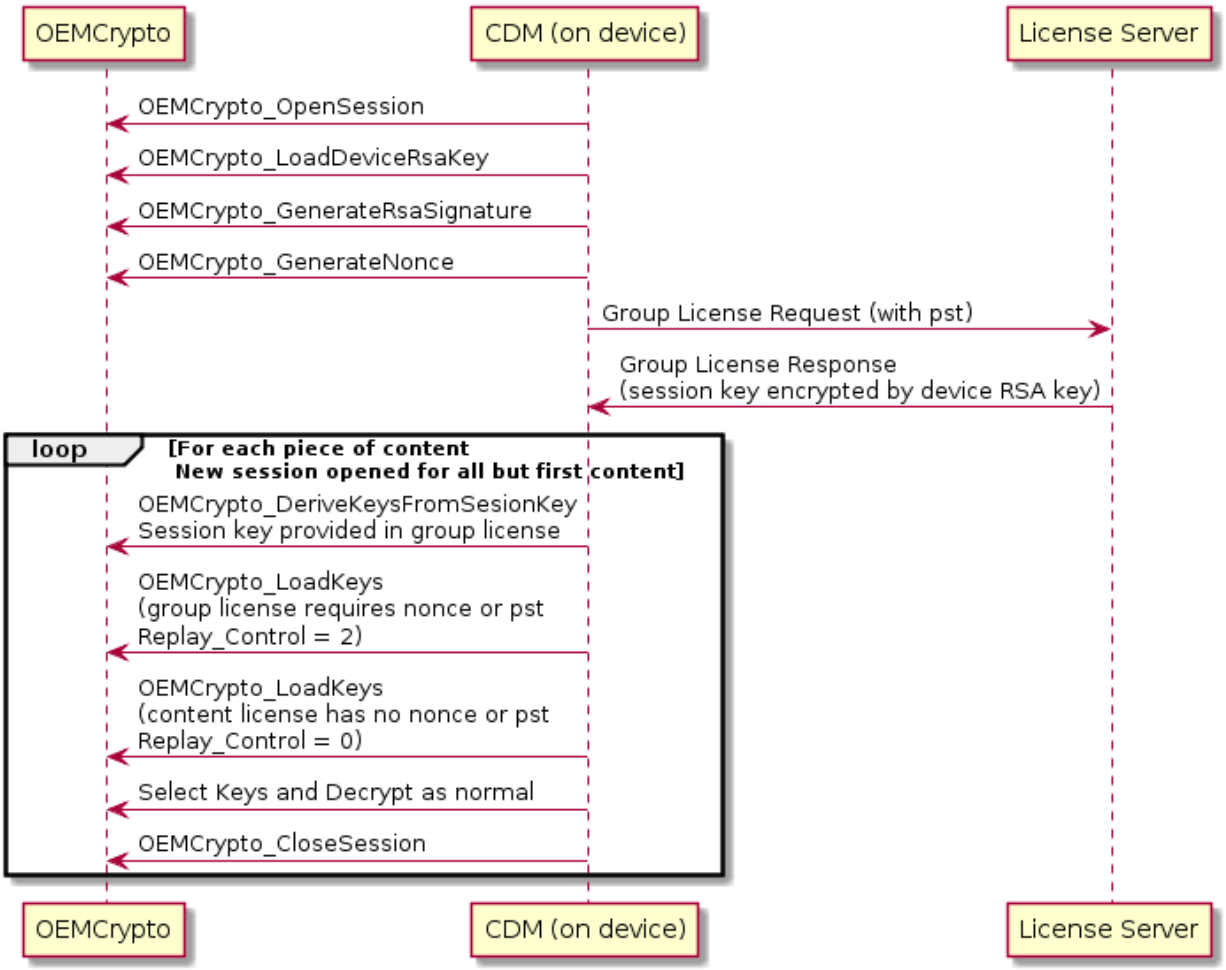

Two level 1 devices with same system ID
OEM
Device 1
Device 2
License Server
Google APFE
Happy Path
License request reports OEM patch level 0
HD License granted
License request reports OEM patch level 0
HD License granted
OEM via APFE/WVDN reports bug in system's TEE. Content provider decides not to serve HD.
After Problem Reported
License request reports OEM patch level 0
SD License granted
License request reports OEM patch level 0
SD License granted
After Problem Fixed
Device 1 refuses update
Device 2 accepts update
License request reports OEM patch level 0
SD License granted
License request reports OEM patch level 1
HD License granted
OEM
Device 1
Device 2
License Server
Google APFE

Here is a sequence diagram showing how OEMCrypto should behave in the normal case, and in the case where there is a man-in-the-middle.



Shared Group License

The term shared group license refers to a collection of content that shares the same session key. The goal is to require a single round trip between the device and the license server for the group license, but allow devices to share the content licenses for content in that group. Each piece of content will have its own content keys. From OEMCrypto's point of view, the license request begins the same as a standard license request for an offline license. When the group license is loaded, it will update the mac keys. A second call to LoadKeys is made with the content license that is signed by these mac keys. Below is a sequence diagram.



There is no API change for this feature. However, new unit tests will verify that LoadKeys may be called multiple times for a session. The second call will use the signing keys that were updated in the first call. The content keys will be loaded in the second call. The content keys from the first call may be discarded. In order to force the nonce check, it is an error for LoadKeys to be called with an empty array of key objects.

Unit Tests Modifications

Although CENC key ids are 16 bytes, generic crypto keys may have shorter key ids. There are new test that verify LoadKeys and SelectKey work with key ids that are less than 16 bytes.

To prevent accidentally ignoring future changes to the key control block, new tests will verify that LoadKeys fails if any reserved fields in the key control are nonzero.

Documentation Clarification: HDCP 2.0

If HDCP is version 2.2, then it must be type 1. This means that the device will verify that downstream devices must also use HDCP version 2.2. This is not an API change, but a clarification of the spec from previous versions. This information has already been included in versions 10.3 and 9.1 of the document, but was not mentioned in the previous delta.

LoadKeys Verification Tightening

Previously, in LoadKeys and RefreshKeys, we required that OEMCrypto verify that each pointer be contained in the message. Now, we also require that thate OEMCrypto verify that the entire string pointed to by a pointer is contained in the message. i.e we are changing

```
(message <= p && p < message+message_length)
```

to

```
(message <= p && p+p_length <= message+message_length).
```

Key Control Block Changes

The functionality described above requires the following changes to the key control block. The key control block should still be verified in the LoadKeys function. To allow for backwards compatibility, OEMCrypto should accept a key control block for previous versions of the API.

The four verification bytes in the key control block are valid if they are “kctl”, “kc09” or “kc10” or “kc11”. An OEMCrypto that implements the new v11 functions described in this document should accept any of these four strings. It should not accept any other string -- in particular, it should NOT accept “kc12”.

As described above in the section about OEM/TEE Version Enforcement, the new field Minimum_Security_Patch_Level has been added to the key control block. This is a six bit number in network byte order. The patch level defaults to 0.

bit 31..28	***bit 27-21***	***bit 20..15***	***bit 14..0***
Previously defined	Reserved *set to 0*	Minimum_Security_Patch_Level	Previously defined

If the current software version, as specified by the OEM, running in the trusted environment is less than the Minimum_Security_Patch_Level, then LoadKeys should reject the license and return OEMCrypto_ERROR_UNKNOWN_FAILURE.