WV Modular DRM Version 15 Delta

Changes from Version 14 to 15.2

Apr 29, 2018

Contents

References

DASH - 23001-7 ISO BMFF Common Encryption, 3rd edition.
DASH - 14496-12 ISO BMFF, 5th edition.
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)

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 v14 library, and want to upgrade to OEMCrypto v15, then this document is for you. If you are starting from scratch, you should read **Widevine Modular DRM Security Integration Guide for Common Encryption (CENC) Version 15**.

Overview

There are several new features required for OEMCrypto version 15. 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. In this document the phrase “OEMCrypto shall …” means that “an implementation of the OEMCrypto library shall …”.

This document describes

- 7 new functions.
- 5 modified functions.
- 4 modified enumeration or data structure.
- 2 new error codes.

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.
PST - Provider Session Token - the string used to track usage information and off line licenses.
SRM - System Renewability Message. Contains blacklist of revoked HDCP keys.

API Version Number

```
uint32_t OEMCrypto_APIVersion();
```

This function should now return 15. If it returns less than 15, then the calling code will assume that OEMCrypto does not support the new v15 features. Depending on the platform, the library may be run in backwards compatibility mode, or it may fail. See the supplemental document for your platform to see which version of OEMCrypto is required.

Resource Rating

The goal of having several levels of performance for OEMCrypto is to easily allow applications and content providers to decide which quality of content to serve to each device. OEMCrypto will report a performance level that it supports, and applications and content providers may use these ratings to make easier business decisions.

This is a new feature for OEMCrypto v15. There is a new function

```
uint32_t OEMCrypto_ResourceRatingTier();
```

This function returns a positive number indicating which resource rating it supports. This value will bubble up to the application level as a property in much the same way security level does. This will allow applications to estimate what resolution and bandwidth the device expects to support.

OEMCrypto unit tests and Android GTS tests will verify that devices do support the resource values specified in the table below at the tier claimed by the device. If a device claims to be a low end device, the OEMCrypto unit tests will only verify the low end performance values.

OEMCrypto implementers should consider the numbers below to be minimum values.

These performance parameters are for OEMCrypto only. In particular, bandwidth and codec resolution are determined by the platform.

Some parameters need more explanation. The Sample size is typically the size of one encoded frame. Converting this to resolution depends on the Codec, which is not specified by OEMCrypto. Some content has the sample broken into several subsamples. The “number of subsamples” restriction requires that any content can be broken into at least that many subsamples. However, this number may be larger if DecryptCENC returns OEMCrypto_ERROR_BUFFER_TOO_LARGE. In that case, the layer above OEMCrypto will break the sample into subsamples of size “Decrypt Buffer Size” as specified in the table below. The “Decrypt Buffer Size” means the size of one subsample that may be passed into DecryptCENC or CopyBuffer without returning error OEMCrypto_ERROR_BUFFER_TOO_LARGE.

The number of keys per session is an indication of how many different track types there can be for a piece of content. Typically, content will have several keys corresponding to audio and video at different

resolutions. If the content uses key rotation, there could be three keys -- previous interval, current interval, and next interval -- for each resolution.

Concurrent playback sessions versus concurrent sessions: some applications will preload multiple licenses before the user picks which content to play. Each of these licenses corresponds to an open session. Once playback starts, some platforms support picture-in-picture or multiple displays. Each of these pictures would correspond to a separate playback session with active decryption.

Decrypted frames per second -- strictly speaking, OEMCrypto only controls the decryption part of playback and cannot control the decoding and display part. However, devices that support the higher resource tiers should also support a higher frame rate. Platforms may enforce these values. For example Android will enforce a frame rate via a GTS test.

Resource Rating Tier	**1 - Low**	**2 - Medium**	**3 - High**
Example Device	A low cost phone that only plays SD would probably be in this tier.	A high cost phone that plays SD or HD would probably be in this tier.	A UHD television or home entertainment device would probably be in this tier.
Sample size (see note above)	1 MB	2 MB	4 MB
Number of Subsamples (see note above)	10	16	32
Decrypt buffer size	100 KB	500 KB	1 MB
Generic crypto buffer size	10 KB	100 KB	500 KB
Number of concurrent sessions	10	20	20
Number of keys per session	4	20	20
Decrypted Frames per Second	30 fps SD	30 fps HD	60 fps HD

HDCP Version Enumeration Includes 2.3

The enumeration OEMCrypto_HDCP_Capability shall have a new value:

```
HDCP_V2_3 = 5,          // HDCP version 2.3 Type 1.
```

This value is valid for the key control block, and for OEMCrypto_GetHDCPCapability.

Wireless Output Clarification

When a license requires HDCP, a device may use a wireless protocol to connect to a display only if that protocol supports the version of HDCP as required by the license. Both WirelessHD (formerly WiFi Display) and Miracast support HDCP.

When reporting current and maximum HDCP levels in OEMCrypto_GetHDCPCapability, the maximum HDCP level should be the maximum value that the device can enforce. The current HDCP level is the value currently negotiated for output through all active ports. For example, if the device has a port connected to a display supporting HDCP 1.0 and another port connected to a different display supporting HDCP 2.0, then the maximum is HDCP 2.0 and the current is HDCP 1.0.

As another example, If the device supports HDCP 2.3 and is connected to an HDCP 2.3 repeater that is able to negotiate a Type 1 connection with an HDCP 2.2 receiver, the current level is 2.3, and the device may display content decrypted with a key that requires HDCP 2.3.

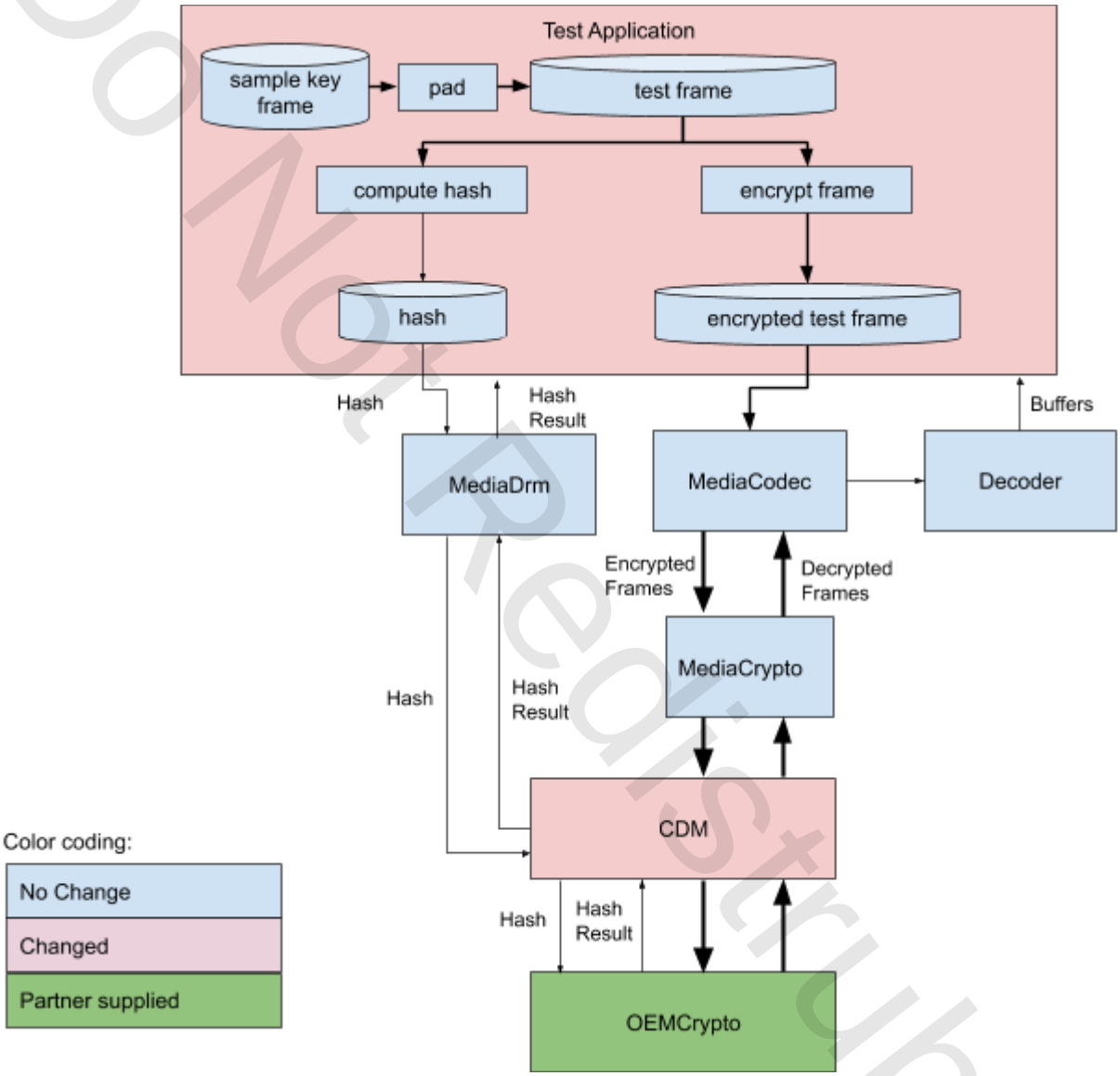
Full Decrypt Path Testing

In order to verify that the full decryption path works for secure buffers with the various pattern decryption standards, some new API functions will be added to verify that the frame to be displayed is correct. While testing the full decrypt path, the keys would be installed as usual. Then, before each frame is decrypted the function OEMCrypto_SetDecryptHash will be called. This sets a hash of next frame. Then the function OEMCrypto_GetHashErrorCode will be called. If the hash of any frame does not match the hash set by the test application, this will return an error code.

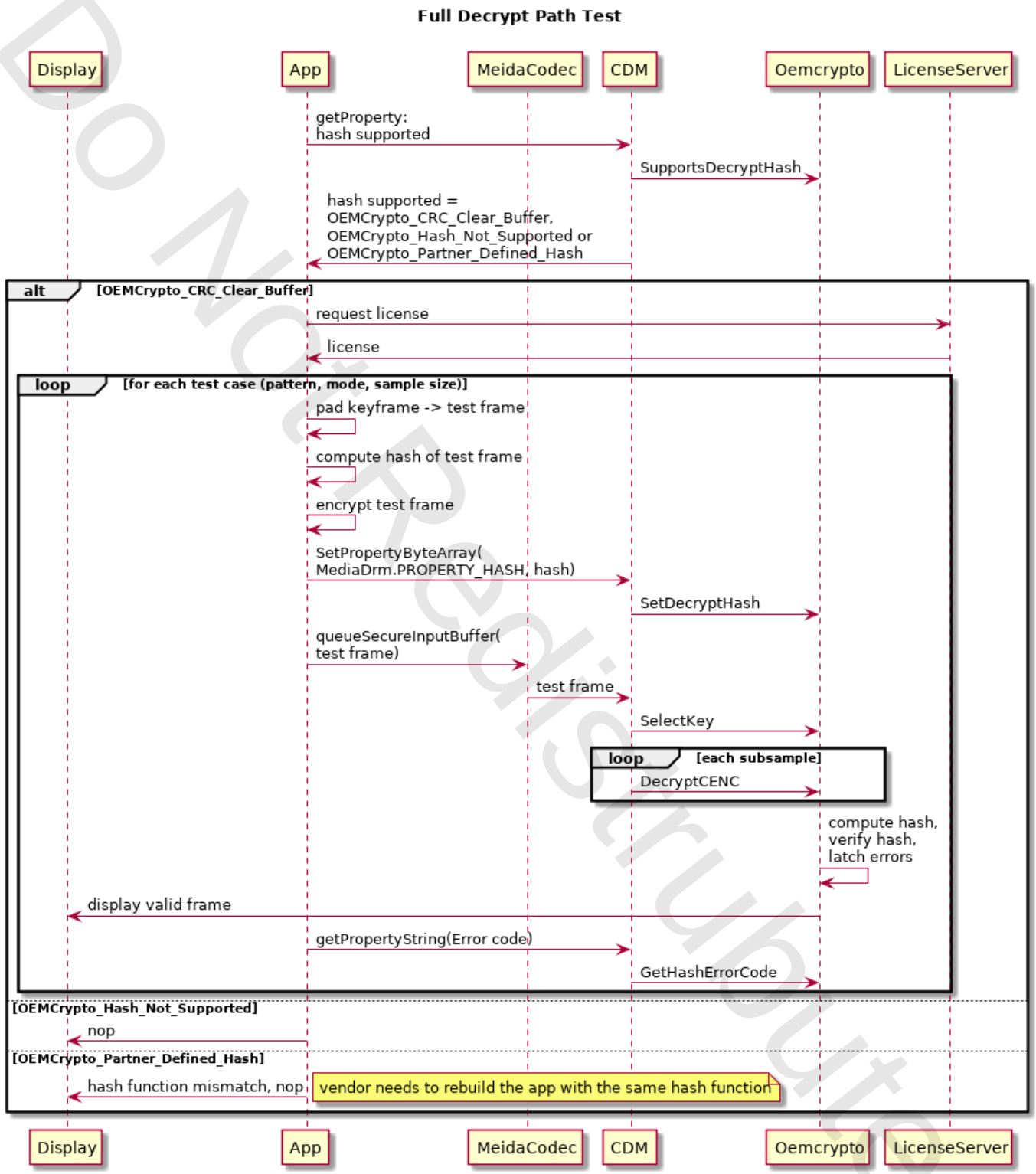
Hashes will only be verified if the key has the Allow_Hash_Verification bit set in the key control block. This bit will only be set on test content.

The main reason to do this is so that the contents of a secure buffer can be verified. Many chip makers have been using a different code path for DecryptCENC when the output buffer is secure than when the output buffer is not secure. This new feature will be used to verify that decryption is working correctly for secure buffers using real video content. This will make it much easier for you to verify that your version of OEMCrypto is working correctly in the future.

Below is a diagram illustrating the data flow for the Android platform. Partners using the source CE CDM release will need to ensure that each frame corresponds to one sample, and that both clear and encrypted subsamples are passed to the CDM layer. Also, partners will need to help write the test application, because each platform handles playback differently.



Below is a sequence diagram for Android. Again, a CE CDM platform would need to coordinate with a Widevine engineer to ensure that the test application sets the hash file correctly. Also, a CE CDM platform needs to ensure each sample is exactly one frame.



New Functions

```
uint32_t OEMCrypto_SupportsDecryptHash();
```

Returns the hash type supported by this device.

```
const uint32_t OEMCrypto_Hash_Not_Supported = 0;
```

A hash type of 0 means this feature is not supported. OEMCrypto is not required by Google to support this feature, but support will greatly improve automated testing.

```
const uint32_t OEMCrypto_CRC_Clear_Buffer = 1;
```

A hash type of 1 means the device will be able to compute the CRC32 checksum of the decrypted content in the secure buffer after a call to OEMCrypto_DecryptCENC. Google intends to provide test applications on some platforms, such as Android, that will automate decryption testing using the CRC 32 checksum of all frames in some test content.

```
const uint32_t OEMCrypto_Partner_Defined_Hash = 2;
```

If an SOC vendor cannot support CRC 32 checksums of decrypted output, but can support some other hash or checksum, then the function should return OEMCrypto_Partner_Defined_Hash and those partners should provide files containing hashes of test content. An application that computes the CRC 32 hashes of test content and builds a hash file in the correct format will be provided by Widevine. The source of this application will be provided so that partners may modify it to compute their own hash format and generate their own hash files.

```
OEMCryptoResult OEMCrypto_SetDecryptHash(OEMCrypto_SESSION session,
                                         uint32_t frame_number,
                                         const uint8_t* hash,
                                         size_t hash_length);
```

Set the hash value for the next frame to be decrypted. This function is called before the first subsample is passed to OEMCrypto_DecryptCENC, when the subsample_flag has the bit OEMCrytpo_FirstSubsample set. The hash is over all of the frame or sample: encrypted and clear subsamples concatenated together, up to, and including the subsample with the subsample_flag having the bit OEMCrypto_LastSubsample set. If hashing the output is not supported, then this will return OEMCrypto_ERROR_NOT_IMPLEMENTED. If the hash is ill formed or there are other error conditions, this could return OEMCrypto_ERROR_UNKNOWN_FAILURE. The length of the hash will be at most 128 bytes.

This function returns OEMCrypto_ERROR_UNKNOWN_FAILURE if the current key does not have the bit Allow_Hash_Verification set in its key control block.

If a call to OEMCrypto_SelectKey is made after the hash is initialized, and the new key does not have the bit Allow_Hash_Verification, then the hash should be discarded and no new hash should be computed while the new key is selected. If an attempt is made to compute a hash when the selected

key does not have the bit Allow_Hash_Verification set, then a hash should not be computed, and DecryptCENC should return the error OEMCrypto_ERROR_UNKNOWN_FAILURE.

OEMCrypto should compute the hash of the frame and then compare it with the correct value. If the values differ, then OEMCrypto should latch in an error and save the frame number of the bad hash. It is allowed for OEMCrypto to postpone computation of the hash until the frame is displayed. This might happen if the actual decryption operation is carried out by a later step in the video pipeline, or if you are using a partner specified hash of the decoded frame. For this reason, an error state must be saved until the call to OEMCrypto_GetHashErrorCode is made.

OEMCrypto_SUCCESS - if the hash was set
OEMCrypto_ERROR_NOT_IMPLEMENTED - function not implemented
OEMCrypto_ERROR_INVALID_SESSION - session not open
OEMCrypto_ERROR_SHORT_BUFFER - hash_length too short for supported hash type
OEMCrypto_ERROR_BUFFER_TOO_LARGE - hash_length too long for supported hash type
OEMCrypto_ERROR_UNKNOWN_FAILURE - other error

```
OEMCryptoResult OEMCrypto_GetHashErrorCode(OEMCrypto_SESSION session,
                                           uint32_t* failed_frame_number);
```

If the hash set in OEMCrypto_SetDecryptHash did not match the computed hash, then an error code was saved internally. This function returns that error and the frame number of the bad hash. This will be called periodically, but not exactly in sync with the decrypt loop. OEMCrypto shall not reset the error state to “no error” once a frame has failed verification. It should be initialized to “no error” when the session is first opened. If there is more than one bad frame, it is the implementer’s choice if it is more useful to return the first bad frame, or the most recent bad frame.

Returns:
OEMCrypto_SUCCESS - if all frames have had a correct hash
OEMCrypto_ERROR_NOT_IMPLEMENTED
OEMCrypto_ERROR_BAD_HASH - if any frame had an incorrect hash
OEMCrypto_ERROR_UNKNOWN_FAILURE - if the hash could not be computed

Recoverable Error Reporting

Three new error codes will be added to indicate recoverable errors:

OEMCrypto_ERROR_OUTPUT_TOO_LARGE -- Returned from OEMCrypto_DecryptCENC or OEMCrypto_CopyBuffer to indicate that decrypt/copy output is too big.

This error code is different from OEMCrypto_ERROR_BUFFER_TOO_LARGE because that indicates breaking the input buffer into smaller subsamples is expected to correct the problem. If the output is too large, then breaking the sample into smaller subsamples will NOT correct the problem; but the application can recover by skipping the current frame, or by switching to a lower resolution.

OEMCrypto_ERROR_SESSION_LOST_STATE -- this session has lost its license or other important state. This might happen if the key ladder for the session has been erased or the RSA key for the session has been erased. It is expected that closing the session, opening a new session, and loading a

new license should allow the application to recover and continue playback.

This error code can be returned from any session function: DecryptCENC, SelectKey, LoadKeys, Sign/Verify, Derive…, and the generic crypto functions.

OEMCrypto_ERROR_SYSTEM_INVALIDATED -- indicates an error condition on OEMCrypto that affects all sessions. It is expected that closing all sessions, calling OEMCrypto_Terminate and then calling OEMCrypto_Initialize again should put the system back in a healthy state.

OEMCrypto_ERROR_SYSTEM_INVALIDATED can be returned from almost any session function: DecryptCENC, SelectKey, LoadKeys, Sign/Verify, Derive…, and also from OpenSession or CloseSession.

Report OEMCrypto build information or number

```
const char* OEMCrypto_BuildInformation();
```

Report the build information of the OEMCrypto library as a short null terminated C string. The string should be at most 128 characters long. This string should be updated with each release or OEMCrypto build.

Some SOC vendors deliver a binary OEMCrypto library to a device manufacturer. This means the OEMCrypto version may not be exactly in sync with the system's versions. This string can be used to help track which version is installed on a device.

It may be used for logging or bug tracking and may be bubbled up to the app so that it may track metrics on errors.

Since the OEMCrypto API also changes its minor version number when there are minor corrections, it would be useful to include the API version number in this string, e.g. "15.1" or "15.2" if those minor versions are released.

VM and Sandbox Support

Although the CDM is usually running on a set top box or a mobile device where there is only one process that interacts with OEMCrypto, there are situations, like on an in-flight entertainment system or on a desktop computer supporting multiple browsers, where several processes or virtual machines (VM) will interact with OEMCrypto. On these devices, each CDM has its own file system where it stores DRM certificates, offline licenses, and usage table data. The term "sandbox" refers to the file system and to the process or VM in which the CDM instance is running. The CDM layer has no knowledge of other running sandboxes. If OEMCrypto is interacting with several sandboxes, then it must have a way to distinguish between different sandboxes. For example, OEMCrypto could look at the process id (PID) of the CDM or it could look at the VM id.

Device manufacturers who wish to use a sandbox model must coordinate with the provider of OEMCrypto to make sure that OEMCrypto can operate properly within a sandbox. **This feature is a special case, and is not commonly supported**.

The CDM can only guarantee the threading rules specified in this document for threads within the same sandbox. For example, there is no guarantee that the CDM layer from different sandboxes will not call OpenSession at the same time.

Each CDM instance will call OEMCrypto_SetSandbox and OEMCrypto_Initialize once before any other calls, and will call OEMCrypto_Terminate after use. Neither of these shall interfere with other sandboxes. In particular, OEMCrypto_Terminate shall not close any sessions which were opened in another sandbox. This expectation only holds for OEMCrypto implementations that are designed to work with a sandbox. As mentioned above: if you are a device maker, and wish to use sandbox support for your device, you must ensure that your OEMCrypto provider supports sandboxes. All other providers of OEMCrypto will assume that there is a single CDM instance.

Another issue that may arise when using multiple sandboxes is the uniqueness of the generation number in the usage table header. Because each CDM instance will have a separate file system, each CDM will have its own usage table header, and usage entries in the table. In order to distinguish among the different headers, the CDM will specify a sandbox ID. This sandbox ID is a string of bytes that uniquely identifies this CDM instance as belonging to a specific sandbox. This allows OEMCrypto to recreate the map from sandbox to the sandbox's persistent data when the sandbox's process or VM is shutdown and started in a new process or VM. The sandbox ID will be sent to OEMCrypto just before OEMCrypto_Initialize is called.

```
OEMCrypto_Result OEMCrypto_SetSandbox(const uint8_t* sandbox_id,
                                      size_t sandbox_id_length);
```

This tells OEMCrypto which sandbox the current process belongs to. Any persistent memory used to store the generation number should be associated with this sandbox id. OEMCrypto can assume that this sandbox will be tied to the current process or VM until OEMCrypto_Terminate is called.

If OEMCrypto does not support sandboxes, it will return OEMCrypto_ERROR_NOT_IMPLEMENTED. On most platforms, this function will just return OEMCrypto_ERROR_NOT_IMPLEMENTED. If OEMCrypto supports sandboxes, this function returns OEMCrypto_SUCCESS on success, and OEMCrypto_ERROR_UNKNOWN_FAILURE on failure.

The CDM layer will call OEMCrypto_SetSandbox once before OEMCrypto_Initialize. After this function is called and returns success, it will be OEMCrypto's responsibility to keep calls to usage table functions separate, and to accept a call to OEMCrypto_Terminate for each sandbox.

Device manufacturers and OEMCrypto providers who wish to support this new feature are strongly encouraged to coordinate integration testing with each other, and with a Widevine engineer.

As a matter of implementation, the function OEMCrypto_SetSandbox will always be called just before OEMCrypto_Initialize. Implementers may wish to put all of the initialization work in the OEMCrypto_SetSandbox function and make OEMCrypto_Initialize a no-op.

Replace Pointers with Offsets in LoadKeys

Replace pointers with unsigned ints as offsets into the message.

```
typedef struct {
  size_t offset;
  size_t length;
} OEMCrypto_Substring;

OEMCryptoResult OEMCrypto_LoadKeys(OEMCrypto_SESSION session,
```

```
                                    const uint8_t* message,
                                    size_t message_length,
                                    const uint8_t* signature,
                                    size_t signature_length,
                                    OEMCrypto_Substring enc_mac_keys_iv,
                                    OEMCrypto_Substring enc_mac_keys,
                                    size_t num_keys,
                                    const OEMCrypto_KeyObject* key_array,
                                    OEMCrypto_Substring pst,
                                    OEMCrypto_Substring srm_restriction_data,
                                    OEMCrypto_LicenseType license_type);
typedef struct {
    OEMCrypto_Substring key_id;
    OEMCrypto_Substring key_data_iv;
    OEMCrypto_Substring key_data;
    OEMCrypto_Substring key_control_iv;
    OEMCrypto_Substring key_control;
} OEMCrypto_KeyObject;
```

Similarly OEMCrypto_RefreshKeys and OEMCrypto_LoadEntitledContentKeys are changed.

```
OEMCryptoResult OEMCrypto_RefreshKeys(OEMCrypto_SESSION session,
                      const uint8_t* message,
                      size_t message_length,
                      const uint8_t* signature,
                      size_t signature_length,
                      size_t num_keys,
                      const OEMCrypto_KeyRefreshObject* key_array);

typedef struct {
    OEMCrypto_Substring key_id;
    OEMCrypto_Substring key_control_iv;
    OEMCrypto_Substring key_control;
} OEMCrypto_KeyRefreshObject;

OEMCryptoResult OEMCrypto_LoadEntitledContentKeys(
    OEMCrypto_SESSION session,
    const uint8_t* message,
    size_t message_length,
    size_t num_keys,
    const OEMCrypto_EntitledContentKeyObject* key_array);

typedef struct {
  OEMCrypto_Substring entitlement_key_id;
  OEMCrypto_Substring content_key_id;
  OEMCrypto_Substring content_key_data_iv;
  OEMCrypto_Substring content_key_data;
} OEMCrypto_EntitledContentKeyObject;
```

The goal is to make range checks slightly more obvious, although OEMCrypto still has to verify that there is no integer overflow when computing (offset + length). Also, if the message buffer is copied to secure memory, key pointers do not need to be updated. Instead of using a null pointer to signify that a field is not present, we will use a zero length substring with a zero offset.

GenerateSignature Clarification

GenerateSignature should use the mac keys in an associated usage entry instead of any derived keys associated with the session. This would happen if LoadKeys is not called, such as when a license release is being signed.

Keybox and Certificate Validation

The function OEMCrypto_IsKeyboxValid has been renamed to OEMCrypto_IsKeyboxOrOEMCertValid. This function is used to verify that the device has been provisioned with a valid keybox or OEM Certificate. OEMCrypto_IsKeyboxValid had always been required for devices with a keybox. In OEMCrypto v14.2, this function was optional for devices with an OEM Certificate -- i.e. using Provisioning 3.0. And now, for OEMCrypto v15, OEMCrypto_IsKeyboxOrOEMCertValid is required for all devices.

On some platforms, such as Android, the function OEMCrypto_InstallKeyboxOrOEMCert can be used to install the device keybox or OEM Certificate at initialization. Please see the Widevine Security Integration Guide Supplement for your platform for more information.

Clarify Padding

Padding is not included in any AES-wrapped keys -- i.e. content keys, entitlement keys, and mac keys. This clarification is needed because currently the license server includes padding. However, the padding is stripped off by the CDM layer before it reaches the OEMCrypto layer. There is no change to the OEMCrypto layer regarding AES padding.

Clarify Device ID Usage

For devices with a keybox, the function OEMCrypto_GetDeviceID is required to return the device ID embedded in the keybox. This requirement does not change in v15 for devices with a keybox.

For devices with an OEM Certificate -- i.e. using Provisioning 3.0 -- the requirement has changed in v15. The function OEMCrypto_GetDeviceID may either

1. Return OEMCrypto_ERROR_NOT_IMPLEMENTED. In this case, it is assumed that each device has a unique OEM Certificate and the CDM layer will use a SHA256 hash of the certificate as the unique identifier.
2. Set the parameter deviceID to be a short device unique identifier. This id must be stable across multiple calls and rebooting the device. It cannot be shared among several devices. It could, for example, be the device serial number.

Level 3 devices that share a common system ID are allowed to have the same OEM Certificate, i.e. a single leaf certificate for all of the devices. These devices use OEMCrypto_GetDeviceID to provide a

way to uniquely identify the device. Even for systems that allow a shared OEM Certificate, each device must still have a unique secret, as well as a unique id. This unique secret is needed so that the device can encrypt the usage table header and entries in such a way that they cannot be copied from one device to another. The unique id is used by the CDM layer to generate a Stable Per Origin Identification (SPOID), which must be unique to each device.

Threading Model Clarification

This section is a clarification and rewording. Also, to make it clear which session CopyBuffer is associated with, its signature is changing.

Applications using the CDM and OEMCrypto may be multithreaded. On some OSes, there may be several applications that are active at the same time. The OEMCrypto threading model is a list of assumptions that OEMCrypto may make about which functions may be called simultaneously with which other functions.

To specify these rules, this document uses the terms “read lock” and “write lock”. When a thread holds a “write lock” on a mutex, no other thread may hold a lock for reading or writing on the same mutex. When a thread holds a “read lock” on a mutex, no other thread may hold a lock for writing on that mutex, but other threads may simultaneously hold a lock for reading. The rules below are for the CDM layer. OEMCrypto may assume that the CDM layer holds the appropriate read or write lock before calling the OEMCrypto function.

We will use a model that assumes there is one mutex for the entire OEMCrypto system. The system mutex will have both read and write locks on it. All functions will hold at least a read lock on the system. Also, each session will have its own mutex. Each of the session mutexes will have write locks on it.

There are four classes of functions for threading, which are described below.

Initialization and Termination Functions

Initialization and termination functions are called sequentially, as if they hold a write lock on the OEMCrypto system. These functions include

- OEMCrypto_SetSandbox
- OEMCrypto_Initialize
- OEMCrypto_InstallKeyboxOrOEMCert
- OEMCrypto_LoadTestKeybox
- OEMCrypto_Terminate

All other functions will be called after OEMCrypto_Initialize and before OEMCrypto_Terminate.

Property Functions

Property functions and functions that do not modify the system are used to gather information about the system. OEMCrypto may assume these are called between the initialization and termination functions,

as if the CDM holds a read lock on the OEMCrypto system.

These functions include OEMCrypto_GetKeyData, OEMCrypto_GetRandom, OEMCrypto_APIVersion, and all functions that are not initialization or termination functions or do not need a session or the usage table.

Session Initialization and Usage Table Functions

Session initialization functions are OEMCrypto_OpenSession and OEMCrypto_CloseSession. Usage table functions are functions that modify the usage table header or update the master generation number. This category includes the functions to create or load the usage table header, create or load a usage table entry, update or deactivate a usage entry, generate a usage report, move usage entries, or shrink the usage table header. These functions will not be called at the same time as any other functions, as if the CDM holds a write lock on the OEMCrypto system.

This category does **not** include functions that only modify data within a usage entry, such as the decryption functions, like OEMCrypt_DecryptCENC, because that does not update the generation number of the entry or the usage header.

Session Functions

Session functions are all of the functions that take a session as a parameter. This category includes OEMCrypto_GenerateNonce, the derive key functions, the key-loading functions, and the decryption functions.

These functions can be called simultaneously with any property function, or simultaneously with any session function from another session. They will not be called simultaneously with any session function for the same session.

These functions may behave as if the CDM layer has at least a read lock on the OEMCrypto system, and a write lock on the individual session they target.

In order to make it clear which session OEMCrypto_CopyBuffer is associated with, the signature of this function changes to:

```
OEMCrypto_CopyBuffer(OEMCrypto_SESSION session,
                     const uint8_t *data_addr,
                     size_t data_length,
                     OEMCrypto_DestBufferDesc* out_buffer,
                     uint8_t subsample_flags);
```

Deprecate error OEMCrypto_KEY_NOT_LOADED

The error code OEMCrypto_KEY_NOT_LOADED is redundant with

OEMCrypto_ERROR_NO_CONTENT_KEY and OEMCrypto_KEY_NOT_ENTITLED. The function LoadEnttiledContentKey should return KEY_NOT_ENTITLED if it does not find the corresponding entitlement key in its key table. All other functions that do not find a key id in the key table should return OEMCrypto_ERROR_NO_CONTENT_KEY. This includes QueryKeyControl, SelectKey, and RefreshKeys.

Schedule for Deprecating Keyboxes

The deprecation schedule has been pushed back.

Key Control Block Changes

The four verification bytes in the key control block are valid if they are “kctl”, “kc09”, “kc10”, “kc11”, ... or “kc15”. An OEMCrypto that implements the new v15 functions described in this document should accept any of these strings. It should not accept any other string -- in particular, it should NOT accept “kc16”.

The following new flag is added to the key control block:

bit 24: Allow_Hash_Verification -- if this bit is set, then the function OEMCrypto_SetDecryptHash may be used to compute a hash of the content that was decrypted with this key. If this bit is not set, and a hash was set for the current session, OEMCrypto_GetHashErrorCode should return OEMCrypto_ERROR_UNKNOWN_FAILURE.

Difference between 15 and 15.1

The difference between versions 15 and 15.1 of this API is as follows.

The design for the Full Decrypt Path Testing application has changed. Instead of reading hashes from an external file, it will use a single key frame and modify it to match the desired size. The test application will then compute the hash and encrypt the frame. For OEMCrypto, this means that there will not be a call to OEMCrypto_InitializeDecryptHash before the frame and OEMCrypto_SetDecryptHash after the frame. Instead, there will be a single call to OEMCrypto_SetDecryptHash before the frame. The function OEMCrypto_InitializeDecryptHash will not be used.

The “Shared License” feature is not used by any production server. This functionality is no longer required and OEMCrypto may reject licenses with a nonzero bit 23 in the key control block.

Difference between 15.1 and 15.2

The difference between versions 15.1 and 15.2 of this API is as follows.

Resource Ratings. The resource rating values for “simultaneous secure playback” has been removed. These resource requirements are more stringent on the secure decoder than on OEMCrypto. We have decided that these requirements should be levied by the system or platform, rather than from OEMCrypto. For example, Android TV strongly recommends that devices should be able to play back two streams simultaneously, but it is not verified by any test. Widevine recommends that device makers that support simultaneously playback of two streams should also support simultaneous playback of two **secure** streams.

Keybox Deprecation. Provisioning 3.0 is supported by Widevine, but will not be required for OEMCrypto v16.

HDCP 2.3 Requirements. Per the HDCP spec, there is no mechanism for an HDCP transmitter that supports HDCP 2.3 to distinguish between 2.2 and 2.3 receivers that are downstream from a repeater that supports 2.3. As such, the wording related to HDCP requirements in the key control block needs to be clarified.

In the description of the function `OEMCrypto_GetHDCPCapability`, the definition of the current and maximum levels are as follows.

The maximum HDCP level should be the maximum value that the device can enforce. For example, if the device has an HDCP 1.0 port and an HDCP 2.0 port, and the first port can be disabled, then the maximum is HDCP 2.0. If the first port cannot be disabled, then the maximum is HDCP 1.0. The maximum value can be used by the application or server to decide if a license may be used in the future. For example, a device may be connected to an external display while an offline license is downloaded, but the user intends to view the content on a local display. The user will want to download the higher quality content.

The current HDCP level should be the level of HDCP currently negotiated with any connected receivers or repeaters either through HDMI or a supported wireless format. If multiple ports are connected, the current level should be the minimum HDCP level of all ports. If the key control block requires an HDCP level equal to or lower than the current HDCP level, the key is expected to be usable. If the key control block requires a higher HDCP level, the key is expected to be forbidden.

When a key has version HDCP_V2_3 required in the key control block, the transmitter must have HDCP version 2.3 and have negotiated a connection with a version 2.3 receiver or repeater. The transmitter must configure the content stream to be Type 1. Since the transmitter cannot distinguish between 2.2 and 2.3 downstream receivers when connected to a repeater, it may transmit to both 2.2 and 2.3 receivers, but not 2.1 receivers.

For example, if the transmitter is 2.3, and is connected to a receiver that supports 2.3 then the current level is `HDCP_V2_3`. If the transmitter is 2.3 and is connected to a 2.3 repeater, the current level is `HDCP_V2_3` even though the repeater can negotiate a connection with a 2.2 downstream receiver for a Type 1 Content Stream.

As another example, if the transmitter can support 2.3, but a receiver supports 2.0, then the current level is `HDCP_V2`.

When a license requires HDCP, a device may use a wireless protocol to connect to a display only if that

protocol supports the version of HDCP as required by the license. Both WirelessHD (formerly WiFi Display) and Miracast support HDCP.

Nonce and MAC Key Collision. Recent security reviews have pointed out that the probability of generating a nonce collision is much higher if an attack is generating nonces simultaneously. The probability is related to the famous "Birthday Paradox". To prevent attacks based on this collision we ask that implementers:

1. Restrict the nonce table to at most 4 nonces per session. There is currently a requirement for at least 4 nonce values, but there is never a need for more than that.
2. OEMCrypto_GenerateNonce should verify that any nonce it generates is not already in the session's nonce table. It should also verify that the nonce is not in any other session's nonce table. This changes the threading model for OEMCrypto_GenerateNonce -- it is now a "session initialization function", instead of just a "session function".
3. To prevent an attack based on modification of the offsets passed into OEMCrypto_LoadKeys, we now require that OEMCrypto_LoadKeys should verify that the substring enc_mac_keys_iv is not the same as the 16 bytes before enc_mac_keys.
4. To prevent an attempt to call OEMCrypto_LoadKeys several times, the session's encrypt_key should be cleared at the end of OEMCrypto_LoadKeys.