WIDEVINE

WV Modular DRM Version 16 Delta

Changes from Version 15 to 16.2

April 6th, 2019

DOCUMENT STATUS: This is the final draft of the delta document. Future changes to the API will have an update to the minor version number. Future changes to wording that do not change the API will have a date update.

Current schedule:

- **July 1, 2019** - Deadline for submitting requirements for OEMCrypto
- ~~**Aug 1**~~ **Aug 7, 2019** - Draft of the Version Delta document released for feedback
- ~~**Aug 15**~~**, Aug 30, 2019** - Deadline for feedback on the draft of the Delta document
- ~~**Sep 1**~~**, Sept 12** - Final Delta document released
- ~~**Oct 15**~~**,Dec 9.** Updated API documentation, Unit tests, headers, and reference code published

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 v15 library, and want to upgrade to OEMCrypto v16, 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) Version 16.

Overview

There are several new features required for OEMCrypto version 16. This update is strongly recommended for all partners to mitigate potential security vulnerabilities. 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, when we say “OEMCrypto shall …” we mean that “an implementation of the OEMCrypto library shall …”.

Most of the changes in this document are related to security or performance.

To improve security, OEMCrypto will parse entire messages and extract important fields. See the sections on Serialize Core Messages, Separate Signature by Message Type and LoadKeys is Replaced by LoadLicense below for details. This moves code from the CDM layer down to the trusted Execution Environment. To make this work easier for partners, Widevine will provide a library that parses messages so that partner’s existing code may be reused.

Other security enhancements are discussed in the sections Nonce Table and Separate Signing Keys for Renewal and License Request below.

For both security and performance reasons, certificates can now use Elliptic Curve based crypto (ECC). For OEMCrypto version 16, ECC is optional.

To improve performance, the DecryptCENC function will be modified to allow several samples to be processed at once. This adds some complexity to the function call, but many partners have expressed desire to do this.

Another improvement to performance is to allow OEMCrypto to delay some error reporting. This allows decryption to be optimized for the "happy path" when there are no errors. The biggest example is to delay verification of HDCP level until just before the content is sent to the HDMI subsystem.

The rest of this document discusses minor changes, new resource tiers, some requirement removal, and documentation clarifications.

Differences Between 16 and 16.1

The following changes have been made since the September 12th, 2019 version number 16.

1. The section Wrapped RSA Key Padding is new.
2. The section Create Secure Buffers for Testing is new.
3. The OEMCrypto_Sign*Request functions have been renamed to OEMCrypto_PrepAndSign*Request to make it clear that they prepare the core messages as well as sign the messages.
4. The function OEMCrypto_PrepAndSignLicenseRequest will not include the core message when computing the signature. This is for backwards compatibility, as discussed below.
5. The OEMCrypto_Load* functions will include a parameter indicating core_message_length. All substring offsets are relative to the message body.
6. Several new ODK helper functions are described in the section ODK Library.
7. The nonce flood requirements have been relaxed. See the section Nonce Table.
8. The number of sessions required for "living room devices" has increased. See the section New Resource Rating Tier.
9. The functions OEMCrypto_LoadKeys and OEMCrypto_RefreshKeys will still be used to load and renew legacy licenses. This includes offline licenses and licenses from servers that have not yet been updated to v16.

Differences Between 16.1 and 16.2

The following changes have been made since the December 9th, 2019 version number 16.1

1. The separate requirements for "living room devices" has been removed.

2. The nonce values, clock values and timer limits structures used by the ODK library have been modified. Logic within the ODK library to enforce license duration has changed. The structure of the license and renewal core messages has changed. The ODK functions have **not** changed.
3. ODK_RefreshV15Values has a new parameter new_key_duration which is the duration parameter from the key object passed into OEMCrypto_RefreshKeys.

Updates on April 6th, 2020

There were no function signature changes, so the API version number did not change from 16.2. There were several grammar and spelling errors. There were also the following corrections:

1. The description of OEMCrypto_LoadProvisioning now says that devices with a keybox use keys derived from the keybox device key, and devices using Provisioning 3.0 use keys derived from the session key. The description was previously reversed.
2. The function OEMCrypto_SupportedPatterns is no longer discussed. This function was never fully defined.
3. The function OEMCrypto_LoadRenewal no longer says that keys and key control blocks should be verified. This is because the function OEMCrypto_LoadRenewal processes a message with no key control block. It should update timers for the entire license.

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 offline licenses.

SRM - System Renewability Message - contains blacklist of revoked HDCP keys.

TEE - Trusted Execution Environment

REE - Rich Execution Environment

Function Changes

The following functions will change, with links to the sections describing the change.

1. OEMCrypto_APIVersion - see section API Version Number
2. OEMCrypto_GenerateNonce- see section Nonce Table and section Clarify OEMCrypto States and Remove Unused Functions

3. OEMCrypto_GenerateDerivedKeys - see section Function Prototype Changes
4. OEMCrypto_DeriveKeysFromSessionKey - see section Provisioning and Elliptic Curve Based Crypto (ECC)
5. OEMCrypto_GenerateSignature - see section Separate Signature by Message Type
6. OEMCrypto_GenerateRSASignature - see section Separate Signature by Message Type
7. OEMCrypto_SupportedCertificates - see section Provisioning and Elliptic Curve Based Crypto (ECC)
8. OEMCrypto_RewrapDeviceRSAKey30 - see section Provisioning Response, and section Provisioning and Elliptic Curve Based Crypto (ECC)
9. OEMCrypto_RewrapDeviceRSAKey - see section Provisioning Response, and section Provisioning and Elliptic Curve Based Crypto (ECC)
10. OEMCrypto_GetOEMPublicCertificate - see section Provisioning and Elliptic Curve Based Crypto (ECC)
11. OEMCrypto_LoadKeys see section LoadKeys is Replaced by LoadLicense
12. OEMCrypto_RefreshKeys - see section Renewal Response
13. OEMCrypto_Generic_Encrypt - see section First Playback -- Enabling Timer and During Playback -- Updating Timer
14. OEMCrypto_Generic_Decrypt - see section First Playback -- Enabling Timer and During Playback -- Updating Timer
15. OEMCrypto_Generic_Sign - see section First Playback -- Enabling Timer and During Playback -- Updating Timer
16. OEMCrypto_Generic_Verify - see section First Playback -- Enabling Timer and During Playback -- Updating Timer
17. OEMCrypto_LoadUsageEntry - see section Reloading Clock Values and section Clarify OEMCrypto States and Remove Unused Functions
18. OEMCrypto_CreateUsageEntry - see section Clarify OEMCrypto States and Remove Unused Functions
19. OEMCrypto_UpdateUsageEntry - see section Usage Entry Updates
20. OEMCrypto_DeactivateUsageEntry - see section Usage Entry Updates
21. OEMCrypto_DecryptCENC - see section Delayed Error Conditions and Combine Decrypt Calls and First Playback -- Enabling Timer and During Playback -- Updating Timer

The following functions will be replaced, with links to the sections describing the change.

1. OEMCrypto_GenerateSignature - see section Separate Signature by Message Type
2. OEMCrypto_GenerateRSASignature - see section Separate Signature by Message Type
3. OEMCrypto_RewrapDeviceRSAKey30 - see section Provisioning Response
4. OEMCrypto_RewrapDeviceRSAKey - see section Provisioning Response
5. OEMCrypto_LoadDeviceRSAKey - see section Provisioning and Elliptic Curve Based Crypto (ECC)
6. OEMCrypto_RefreshKeys - see section Renewal Response
7. OEMCrypto_GetRandom - deleted. Not used.

New Functions:

1. OEMCrypto_MinorAPIVersion - see section Minor Version Number
2. OEMCrypto_PrepAndSignLicenseRequest - see section Separate Signature by Message Type
3. OEMCrypto_PrepAndSignRenewalRequest - see section Separate Signature by Message Type
4. OEMCrypto_PrepAndSignProvisioningRequest - see section Separate Signature by Message

Type
5. OEMCrypto_LoadLicense - see section LoadKeys is Replaced by LoadLicense and Setting TImer Limits
6. OEMCrypto_LoadOEMPrivateKey - see section Provisioning and Elliptic Curve Based Crypto (ECC)
7. OEMCrypto_LoadDRMPrivateKey - see section Provisioning and Elliptic Curve Based Crypto (ECC)
8. OEMCrypto_LoadRenewal - see section Renewal Response
9. OEMCrypto_LoadProvisioning - see section Provisioning Response
10. OEMCrypto_MaximumUsageTableHeaderSize - see section Usage Table Size is Queryable

General Recommendations

Although OEMCrypto usually runs within a Trusted Execution Environment, we should still avoid giving unnecessary information about how the system works to an attacker.

- Widevine requires that production versions of the OEMCrypto library be stripped of debugging symbols and libraries should be linked with as few external symbols as possible -- i.e. with hidden visibility.
- Widevine requires that error logging messages be obfuscated. If possible, debugging logs should be removed from production systems.
- If available, the code should be built with Stack Smash Protectors (SSP) enabled.
- If possible, the Trusted Application (TA) should be encrypted as well as signed so that attackers cannot perform static analysis on OEMCrypto.
- The Root of Trust should be heavily protected. For example, it should not be left unencrypted in memory when not in use. This will make it harder for an attacker who has access to runtime memory through another attack.
- Code should be reviewed by an engineer who is not the author, and other “Industry best practices” shall be followed.
- The TEE should have hardware Anti-Rollback turned on for production devices.

Unless otherwise stated, buffers that are passed into OEMCrypto should not be writeable by the REE while OEMCrypto is processing them. For example, it should be impossible for the REE to modify data between the time OEMCrypto verifies a signature and then processes the data. There are exceptions to this rule: for example, buffers that hold encrypted content may be kept in memory that is shared between the REE and the TEE. In the functions below, these buffers are marked as pointers to OEMCrypto_SharedMemory. A OEMCrypto_SharedMemory pointer is typedef to a uint8_t.

API Version Number

```
uint32_t OEMCrypto_APIVersion();
```

This function shall now return 16. If it returns less than 16, then the calling code will assume that OEMCrypto does not support the new v16 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.

```
uint32_t OEMCrypto_MinorAPIVersion(void);
```

This function shall return 2.

Minor Version Number

This feature is to prevent the scenario where a bug is found in the system that requires changes to the behavior of both OEMCrypto and the CDM, but does not require a change to any API function prototypes. Even if both the SOC and Widevine fix their respective code bases, it may not be clear to OEMs what is needed to fix the bug.

The minor version of this document does **not** have to match the minor version of the CDM code, either for Android or CE CDM devices. The minor version number for this document is not updated on a regular schedule. Instead, it is incremented whenever a change is made after the first release to indicate that there has been a change that modifies expected behavior.

In order to help SOCs coordinate with OEMs on these bug fixes, the CDM minor version will have better documentation. The CDM minor version and the OEMCrypto build information will be logged at initialization.

For Android devices, a GTS test will be added whenever the CDM minor version needs to be updated to fix a critical bug.

For CE CDM the version will be documented as part of the quarterly release.

To help devices know which version of OEMCrypto they are using, in addition to the existing OEMCrypto_APIVersion and OEMCrypto_BuildInformation functions, a new function is being added:

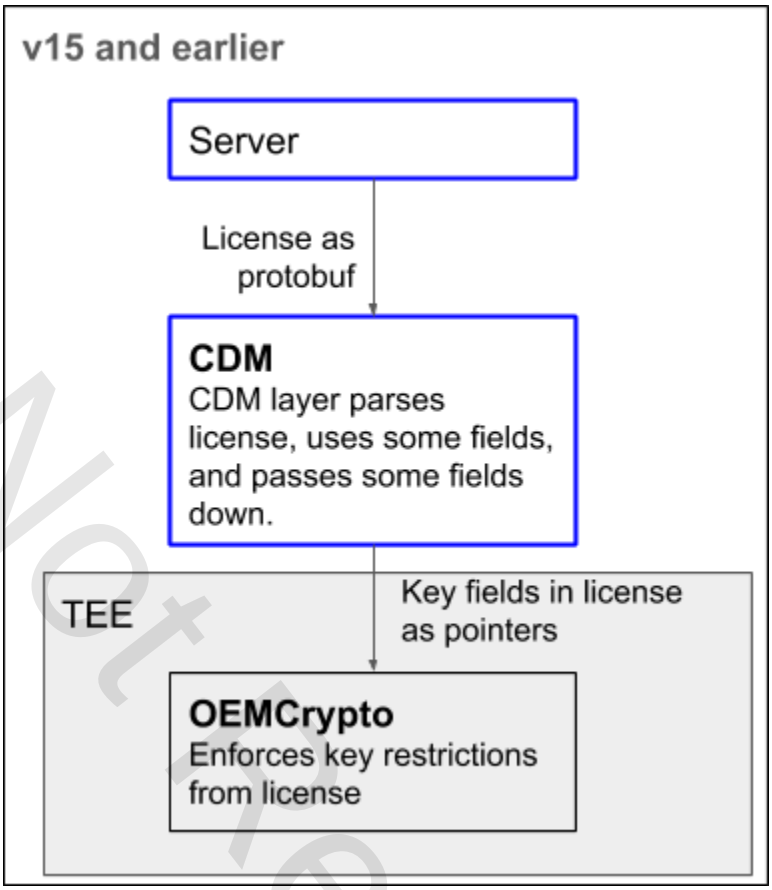
```
uint32_t OEMCrypto_MinorAPIVersion(void);
```

This shall return the minor version of this document. This number will be made available to the layer above as a property.

Serialize Core Messages

Provisioning, License, and License Renewal[1] messages have previously been parsed by the CDM and important information has been passed into OEMCrypto as substrings of the message. However, the pointers themselves were not signed by the server.

[1] A License Release message is treated as a special type of license renewal message.

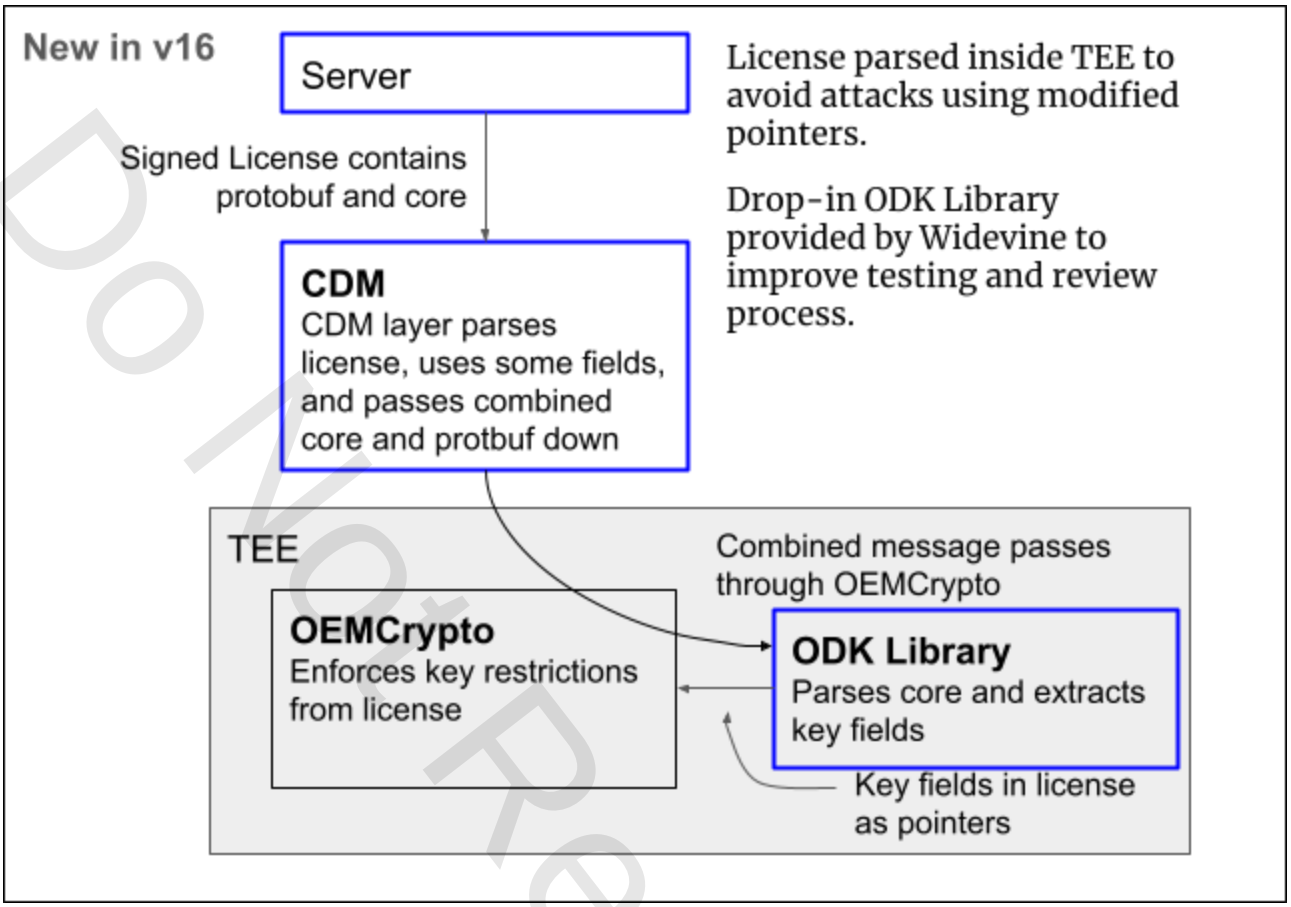


Similarly, the three request messages that are sent to the server and signed by OEMCrypto have data that was not verified by OEMCrypto before signing.

The three main design goals for v16 are to

1. protect the pointers to key fields from being modified.
2. minimize change to existing OEMCrypto design.
3. minimize risk from adding complicated parsing code to OEMCrypto.

For OEMCrypto v16, all fields used by OEMCrypto in each of these messages have been identified, as described later in this document. We will call these fields the core of the message, and will use a simplified method to serialize them. OEMCrypto will parse and verify the core of the message. Fields that are not used by OEMCrypto will still be in a protobuf and are parsed and used by the CDM layer.



For request messages, OEMCrypto will generate the core message. Then it will sign the combined message.

For messages from the server, OEMCrypto will verify the signature first, and then parse and verify the core message.

CDM and Server Changes

The CDM and server changes, as well as the format of core messages, is documented in the design document “Widevine Core Message Serialization”.

ODK Library

Widevine will provide a library of functions in C that can be used to generate core request messages and that parse response messages. The functions that parse code will fill out a struct that has a similar format to the function parameters in the v15 functions that are being replaced. These functions will be provided in source code and it is our intention that partners can build and link this library with their implementation of OEMCrypto with no changes, or very few changes to the source. These functions will have a name prefixed by ODK. The ODK library will be delivered along with the OEMCrypto unit tests and reference code.

OEMCrypto implementers shall build the ODK library as part of the TEE application. All memory and buffers used by the ODK library shall be sanitized by the OEMCrypto implementer to prevent memory attacks by a malicious REE application. OEMCrypto shall check for memory permissions, memory overlaps, integer overflows etc.

OEMCrypto is required to use a compatible version of the ODK library. For version 16, the compatible version of the ODK library is 16.2.

Each OEMCrypto session should maintain several structures of data that are modified by the ODK library. These structures are defined below in the section “Complete ODK API” of the document “Widevine Core Message Serialization”.

`ODK_TimerLimits timer_limits;`
Timer limits are specified in a license and are used to determine when playback is allowed. See the document “License Duration and Renewal” for a discussion on the time restrictions that may be placed on a license.

`ODK_ClockValues clock_values;`
Clock values are modified when decryption occurs or when a renewal is processed. They are used to track the current status of the license -- i.e. has playback started? When does the timer expire? See the document “License Duration and Renewal” for a discussion on the time restrictions that may be placed on a license. Most of these values shall be saved with the usage entry.

`ODK_NonceValues nonce_values;`
Nonce values are used to match a license or provisioning request to a license or provisioning response. For this reason, the api_version might be lower than that supported by OEMCrypto. The api_version matches the version of the license. Similarly the nonce and session_id match the session that generated the license request. For an offline license, these might not match the session that is loading the license. We use the nonce to prevent a license from being replayed. By also including a session_id in the license request and license response, we prevent an attack using the birthday paradox to generate nonce collisions on a single device.

The ODK API functions for message generation and parsing are described below with their corresponding OEMCrypto. The ODK API functions for processing timers are described in the section timers and renewals. Other ODK functions are as follows:

```
OEMCryptoResult ODK_InitializeSessionValues(
                            ODK_TimerLimits *timer_limits,
                            ODK_ClockValues *clock_values,
                            ODK_NonceValues *nonce_values,
                            uint32_t api_version,
                            uint32_t session_id);
```

This function initializes the session’s data structures. It shall be called from OEMCrypto_OpenSession.

The parameters are:

- [out] timer_limits: The session’s timer limits.
- [out] clock_values: The session’s clock values.
- [out] nonce_values: The session’s ODK nonce values.

- [in] api_version: The API Version of OEMCrypto.
- [in] session_id: The session id of the newly created session.

```
OEMCryptoResult ODK_SetNonceValues(ODK_NonceValues *nonce_values, uint32_t nonce);
```

This function updates the nonce value. It shall be called from OEMCrypto_GenerateNonce.

The parameters passed into this function are:

- [in/out] nonce_values: the session's nonce data.
- [in] nonce: the new nonce that was just generated.

Separate Signature by Message Type

Rather than sign blobs of data, OEMCrypto will be asked to verify and sign specific messages.

License Request

A new OEMCrypto API is:

```
OEMCryptoResult OEMCrypto_PrepAndSignLicenseRequest(
                            OEMCrypto_SESSION session,
                            uint8_t* message,
                            size_t message_length,
                            size_t* core_message_size,
                            uint8_t* signature,
                            size_t* signature_length);
```

OEMCrypto will use ODK_PrepareCoreLicenseRequest to prepare the core message. If it returns OEMCrypto_SUCCESS, then OEMCrypto shall sign the message body using the DRM certificate's private key. If it returns an error, the error should be returned by OEMCrypto to the CDM layer.

The message body is the buffer starting at `message + core_message_size`, and with length `message_length-core_message_size`. The reason OEMCrypto only signs the message body and not the entire message is to allow a v16 device to request a license from a v15 license server.

OEMCrypto shall compute a hash (SHA256) of the core license request. The core license request is the buffer starting at `message` and with length `core_message_size`. The hash will be saved with the session and verified that it matches a hash in the license response for v16 licenses.

OEMCrypto shall also call the function ODK_InitializeClockValues, described in the document "License Duration and Renewal", to initialize the sessions clock values.

The parameters passed from the CDM are:

- [in/out] message: Pointer to memory for the entire message. Modified by OEMCrypto via the ODK library.
- [in] message_length: length of the entire message buffer.
- [in/out] core_message_size: length of the core message at the beginning of the message. (in) size of buffer reserved for the core message, in bytes. (out) actual length of the core message,

in bytes.
- [out] signature: pointer to memory to receive the computed signature.
- [in/out] signature_length: (in) length of the signature buffer, in bytes. (out) actual length of the signature, in bytes.

```
OEMCryptoResult ODK_PrepareCoreLicenseRequest(
                            uint8_t* message,
                            size_t message_length,
                            size_t* core_message_size,
                            const ODK_NonceValues *nonce_values);
```

The parameters passed in by the ODK function:

- [in/out] message: Pointer to memory for the entire message. Modified by the ODK library.
- [in] message_length: length of the entire message buffer.
- [in/out] core_message_size: length of the core message at the beginning of the message. (in) size of buffer reserved for the core message, in bytes. (out) actual length of the core message, in bytes.
- [in] nonce_values: pointer to the session's nonce data.

Here, and in other functions where the buffer length is an in/out variable, the caller should set the value to 0 and make an initial call to the function. The function will modify the variable to be the correct buffer size. Then the calling function should call the function again after allocating a buffer with the correct size. If OEMCrypto_PrepAndSIgnLicenseRequest is called with core_message_length* == 0 or signature_length* == 0, then it should first call the ODK_PrepareCoreLicenseRequest, and then it should set the correct value of signature_length*, then it should return OEMCrypto_ERROR_SHORT_BUFFER.

Renewal Request

A new OEMCrypto API is:

```
OEMCryptoResult OEMCrypto_PrepAndSignRenewalRequest(
                            OEMCrypto_SESSION session,
                            uint8_t* message,
                            size_t message_length,
                            size_t* core_message_size,
                            uint8_t* signature,
                            size_t* signature_length);
```

OEMCrypto will use ODK_PrepareCoreRenewalRequest to prepare the core message.

If it returns an error, the error should be returned by OEMCrypto to the CDM layer. If it returns OEMCrypto_SUCCESS, then OEMCrypto computes the signature using the renewal mac key which was delivered in the license via LoadLicense.

If nonce_values.api_version is 16, then OEMCrypto shall compute the signature of the entire message using the session's client renewal mac key. The entire message is the buffer starting at `message` with length `message_length`.

If nonce_values.api_version is 15, then OEMCrypto shall compute the signature of the message body using the session's client renewal mac key. The message body is the buffer starting at `message+core_message_size` with length `message_length-core_message_size`. If the

session has not had a license loaded, it will use the usage entries client mac key to sign the message body.

The parameters passed from the CDM are:

- [in/out] message: Pointer to memory for the entire message. Modified by OEMCrypto via the ODK library.
- [in] message_length: length of the entire message buffer.
- [in/out] core_message_size: length of the core message at the beginning of the message. (in) size of buffer reserved for the core message, in bytes. (out) actual length of the core message, in bytes.
- [out] signature: pointer to memory to receive the computed signature.
- [in/out] signature_length: (in) length of the signature buffer, in bytes. (out) actual length of the signature, in bytes.

```
OEMCryptoResult ODK_PrepareCoreRenewalRequest(
                          uint8_t *message,
                          size_t message_length,
                          size_t *core_message_size,
                          const ODK_NonceValues *nonce_values,
                          ODK_ClockValues *clock_values,
                          uint64_t system_time_seconds);
```

The variables passed in are verified by the ODK function:

- [in/out] message: Pointer to memory for the entire message. Modified by the ODK library.
- [in] message_length: length of the entire message buffer.
- [in/out] core_message_size: length of the core message at the beginning of the message. (in) size of buffer reserved for the core message, in bytes. (out) actual length of the core message, in bytes.
- [in] nonce_values: pointer to the session's nonce data.
- [in] clock_values: the session's clock values.
- [in] system_time_seconds: the current time on OEMCrypto's clock, in seconds.

The discussion of timers and clocks are discussed in the document "TImer and License Renewal Updates". It is important to notice that the nonce passed into the renewal message is from the original message loaded via LoadLicense. A new nonce is not used for each renewal.

Provisioning Request

A new OEMCrypto API is:

```
OEMCryptoResult OEMCrypto_PrepAndSignProvisioningRequest(
                            OEMCrypto_SESSION session,
                            uint8_t* message,
                            size_t message_length,
                            size_t* core_message_size,
                            uint8_t* signature,
                            size_t* signature_length);
```

OEMCrypto will use ODK_PrepareCoreProvisioningRequest to prepare the core message. If it returns

an error, the error should be returned by OEMCrypto to the CDM layer. If it returns OEMCrypto_SUCCESS, then OEMCrypto shall compute the signature of the entire message. The entire message is the buffer starting at `message` with length `message_length`.

For a device that has a keybox, i.e. Provisioning 2.0, OEMCrypto will sign the response with the session's derived client mac key.

For a device that has an OEM Certificate, i.e. Provisioning 3.0, OEMCrypto will sign the response with the private key associated with the OEM Certificate.

The parameters passed from the CDM are:

- [in/out] message: Pointer to memory for the entire message. Modified by OEMCrypto via the ODK library.
- [in] message_length: length of the entire message buffer.
- [in/out] core_message_size: length of the core message at the beginning of the message. (in) size of buffer reserved for the core message, in bytes. (out) actual length of the core message, in bytes.
- [out] signature: pointer to memory to receive the computed signature.
- [in/out] signature_length: (in) length of the signature buffer, in bytes. (out) actual length of the signature, in bytes.

```
OEMCryptoResult ODK_PrepareCoreProvisioningRequest(
                            uint8_t *message,
                            size_t message_length,
                            size_t *core_message_size,
                            const ODK_NonceValues *nonce_values,
                            const uint8_t *device_id,
                            size_t device_id_length);
```

The parameters passed in by the ODK function:

- [in/out] message: Pointer to memory for the entire message. Modified by the ODK library.
- [in] message_length: length of the entire message buffer.
- [in/out] core_message_size: length of the core message at the beginning of the message. (in) size of buffer reserved for the core message, in bytes. (out) actual length of the core message, in bytes.
- [in] nonce_values: pointer to the session's nonce data.
- [in] device_id: For devices with a keybox, this is the device id from the keybox. For devices with an OEM Certificate, this is a device unique id string.
- [in] device_id_length: length of device_id. The device id can be at most 64 bytes.

LoadKeys will be Replaced by LoadLicense

The function OEMCrypto_LoadKeys is being deprecated and replaced by OEMCrypto_LoadLicense. Legacy licenses will still be loaded with OEMCrypto_LoadKeys and new licenses will be loaded with OEMCrypto_LoadLicense. The CDM layer above OEMCrypto will decide if a license is legacy or new depending on the presence of a core license response.

Legacy License Response

The function OEMCrypto_LoadKeys will be used to load keys for legacy licenses only. This includes offline licenses and licenses that are from servers that have not updated their SDK until the summer of 2020. The functionality of OEMCrypto_LoadKeys does not change except that it shall call ODK_InitializeV15Values after decrypting the keys and key control blocks. In particular, it shall still verify each key control block and verify replay control and nonces as needed.

```
OEMCryptoResult ODK_InitializeV15Values(
                          ODK_TimerLimits *timer_limits,
                          ODK_ClockValues *clock_values,
                          ODK_NonceValues *nonce_values,
                          uint32_t key_duration,
                          uint64_t system_time_seconds);
```

This function sets all limits in the timer_limits struct to the key_duration and initializes the other values. The field nonce_values.api_version will be set to 15. The parameters are:

- [out] timer_limits: The session's timer limits.
- [in/out] clock_values: The session's clock values.
- [in/out] nonce_values: The session's ODK nonce values.
- [in] key_duration: The duration from the first key's key control block. In practice, the key duration is the same for all keys and is the same as the license duration.
- [in] system_time_seconds: The current time on the system clock, as described in the document "License Duration and Renewal".

License Response

For v16 licenses the keys will be loaded using OEMCrypto_LoadLicense. The existing function LoadEntitledContentKeys will continue to be used for entitled content keys.

```
OEMCryptoResult OEMCrypto_LoadLicense(OEMCrypto_SESSION session,
                                      const uint8_t* message,
                                      size_t message_length,
                                      size_t core_message_length,
                                      const uint8_t* signature,
                                      size_t signature_length);
```

OEMCrypto will verify the license and load the keys from the license into the current session. It does this in the following way:

1. The signature of the message shall be computed using the session's derived server mac key, and the API shall verify the computed signature matches the signature passed in. If not, return OEMCrypto_ERROR_SIGNATURE_FAILURE. The signature verification shall use a

constant-time algorithm (a signature mismatch will always take the same time as a successful comparison). The signature is over the entire message buffer starting at `message` with length `message_length`.

2. The function ODK_ParseLicense is called to parse the message. If it returns an error, OEMCrypto shall return that error to the CDM layer.
3. Since the ODK library is also running in a Trusted Execution Environment, OEMCrypto does not need to verify each substring is in range.
4. If the enc_mac_keys substring is nonzero, it will be decrypted using the sessions's encryption key and stored in the session's renewal mac keys.
5. If the session is associated with a usage table entry, and the PST has zero length, then an error OEMCrypto_ERROR_WRONG_PST is returned. Otherwise, OEMCrypto shall perform the verification in the section "Usage Entry Verification" below.
6. OEMCrypto shall loop through each key object and decrypt the key data using the session's encryption key. It shall use the content key to decrypt the key control block.
7. OEMCrypto shall perform the same verification of the key control block as it did in the v15 spec.
8. OEMCrypto will check each key control block for the bit SRMVersionRequired and perform the same checks as were done for OEMCrypto v15.

Usage Entry Verification

The parameter usage_entry_present shall be set to true if a usage entry was created or loaded for this session. This parameter is passed into ODK_ParseLicense and used for usage entry verification.

The parameter initial_license_load shall be false if the usage entry was loaded. If there is no usage entry or if the usage entry was created with OEMCrypto_CreateNewUsageEntry, then initial_license_load shall be true.

If a usage entry is present, then it shall be verified after the call to ODK_ParseLicense.

If initial_license_load is true:

1. OEMCrypto shall copy the PST from the parsed license to the usage entry.
2. OEMCrypto shall verify that the server and client mac keys were updated by the license. The server and client mac keys shall be copied to the usage entry.

If initial_license_load is false:

1. OEMCrypto shall verify the PST from the parsed license matches that in the usage entry. If not, then an error OEMCrypto_ERROR_WRONG_PST is returned.
2. OEMCrypto shall verify that the server and client mac keys were updated by the license. OEMCrypto shall verify that the server and client mac keys match those in the usage entry. If not the error OEMCrypto_ERROR_WRONG_KEYS is returned.

After Verification

After verifying all the data, OEMCrypto_LoadLicense shall continue as the v15 function OEMCrypto_LoadKeys and it shall update the timer and clock values. See the design document "TImer and License Renewal Updates" for a design of timers and clocks.

The function that OEMCrypto shall call to verify and parse the license is:

```
OEMCryptoResult ODK_ParseLicense(const uint8_t* message,
                          size_t message_length,
                          size_t core_message_length,
```

```
                            bool initial_license_load,
                            bool usage_entry_present,
                            const uint8_t request_hash[ODK_SHA256_HASH_SIZE],
                            ODK_TimerLimits *timer_limits,
                            ODK_ClockValues *clock_values,
                            ODK_NonceValues *nonce_values,
                            ODK_ParsedLicense *parsed_license);

typedef struct {
  OEMCrypto_Substring enc_mac_keys_iv;
        OEMCrypto_Substring enc_mac_keys;
  OEMCrypto_Substring pst;
  OEMCrypto_Substring srm_restriction_data;
  OEMCrypto_LicenseType license_type;
  ODK_TimerLimits timer_limits;
  size_t key_array_length;
  OEMCrypto_KeyObject key_array[ODK_MAX_NUM_KEYS];
} ODK_ParsedLicense;

typedef struct {
  bool soft_enforce_rental_duration;
  bool soft_enforce_playback_duration;
  uint64_t earliest_playback_start_seconds;
  uint64_t rental_duration_seconds;
  uint64_t total_playback_duration_seconds;
  uint64_t initial_renewal_duration_seconds;
} ODK_TimerLimits;

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;
```

The variables passed in are verified by the ODK function:

- [in] message - pointer to the message buffer.
- [in] message_length - length of the entire message buffer.
- [in] core_message_size - length of the core message, at the beginning of the message buffer.
- [in] initial_license_load - true when called for OEMCrypto_LoadLicense and false when called for OEMCrypto_ReloadLicense.
- [in] usage_entry_present - true if the session has a new usage entry associated with it created via OEMCrypto_CreateNewUsageEntry.
- [in] request_hash: the hash of the license request core message. This was computed by OEMCrypto when the license request was signed.
- [in/out] timer_limits: The session's timer limits. These will be updated.
- [in/out] clock_values: The session's clock values. These will be updated.
- [in/out] nonce_values: The session's ODK nonce values. These will be updated.
- [out] parsed_license - the destination for the data.

The function ODK_ParseLicense will parse the message and verify

1. Either the nonce matches the one passed in or the license does not require a nonce.

2. The API version of the message matches.
3. The session id matches.

The function ODK_ParseLicense will parse the message and set each substring pointer to point to a location in the original message. If the message does not parse correctly, ODK_VerifyAndParseLicense will return an error that OEMCrypto should return to the CDM layer above.
The number ODK_MAX_NUM_KEYS is a compile time constant defined by the platform. See the section on resource ratings for minimum values for this constant.

Legacy Renewal Response

The function OEMCrypto_RefreshKeys will be used to renew the license for legacy licenses only. This means license with an api_version < 16. This includes offline licenses and licenses that are from servers that have not updated their SDK until the summer of 2020. The functionality of OEMCrypto_RefreshKeys may be reduced as follows:

1. OEMCrypto shall verify the signature of the message, as before.
2. Extract the duration field from the key control block of the first element in the array OEMCrypto_KeyRefreshObject.
3. OEMCrypto shall call ODK_RefreshV15Values to update the clock_values, and the timer limit.

```
OEMCryptoResult ODK_RefreshV15Values(const ODK_TimerLimits* timer_limits,
                                     ODK_ClockValues* clock_values,
                                     const ODK_NonceValues* nonce_values,
                                     uint64_t system_time_seconds,
                                     uint32_t new_key_duration,
                                     uint64_t* timer_value);
```

This function updates the clock_values as needed if the renewal is accepted. The field nonce_values.api_version is verified to be 15. The parameters are:

- [in] timer_limits: The session's timer limits.
- [in/out] clock_values: The session's clock values.
- [in] nonce_values: The session's ODK nonce values.
- [in] system_time_seconds: The current time on the system clock, as described in the document "License Duration and Renewal".
- [in] new_key_duration: The duration from the first OEMCrypto_KeyRefreshObject in key_array.
- [out] timer_value: set to the new timer value. Only used if the return value is ODK_SET_TIMER.

Renewal Response

```
OEMCryptoResult OEMCrypto_LoadRenewal(OEMCrypto_SESSION session,
                                      const uint8_t* message,
                                      size_t message_length,
                                      size_t core_message_length,
                                      const uint8_t* signature,
                                      size_t signature_length);
```

OEMCrypto shall verify the signature of the message using the session's renewal mac key for the server. If the signature does not match, OEMCrypto returns OEMCrypto_ERROR_SIGNATURE_FAILURE.

If nonce_values.api_version is 16, then OEMCrypto shall verify the signature of the entire message using the session's server renewal mac key. The entire message is the buffer starting at `message` with length `message_length`.

If nonce_values.api_version is 15, then OEMCrypto shall compute the signature of the message body using the session's server renewal mac key. The entire message is the buffer starting at `message+core_message_size` with length `message_length-core_message_size`.

If the signature passes, OEMCrypto shall use the function ODK_ParseRenewal to parse and verify the message. If ODK_ParseRenewal returns an error OEMCrypto returns the error to the CDM layer.

If ODK_ParseRenewal returns success, then the session's timers and clocks will be updated as described in the document "TImer and License Renewal Updates".

```
OEMCryptoResult ODK_ParseRenewal(const uint8_t* message,
                                 size_t message_length,
                                 size_t core_message_length,
                                 const ODK_NonceValues *nonce_values,
                                 uint64_t system_time_seconds,
                                 const TimerLimits* timer_limits,
                                 ODK_ClockValues* clock_values,
                                 uint64_t *timer_value)
```

The variables passed in are verified by the ODK function:

- [in] message - pointer to the message buffer.
- [in] message_length - length of the entire message buffer.
- [in] core_message_size - length of the core message, at the beginning of the message buffer.
- [in] nonce_values: pointer to the session's nonce data.
- [in] system_time_seconds: the current time on OEMCrypto's clock, in seconds.
- [in] timer_limits - timer limits specified in the license.
- [in/out] clock_values: the sessions clock values.
- [out] timer_value: set to the new timer value. Only used if the return value is ODK_SET_TIMER.

Timers and clocks are described in the document "TImer and License Renewal Updates" and in "Widevine Modular DRM Version 16 Delta". ODK_ParseRenewal returns:

- ODK_ERROR_CORE_MESSAGE if the message did not parse correctly, or there were other incorrect values. An error should be returned to the CDM layer.
- ODK_SET_TIMER: Success. The timer should be reset to the specified timer value.
- ODK_DISABLE_TIMER: Success, but disable timer. Unlimited playback is allowed.
- ODK_TIMER_EXPIRED -- Set timer as diabled. Playback is **not** allowed.
- ODK_STALE_RENEWAL: This renewal is not the most recently signed. It is rejected.

Provisioning Response

The functions OEMCrypto_RewrapDeviceRSAKey and OEMCrypto_RewrapDeviceRSAKey30 will be replaced by the provisioning response function:

```
OEMCryptoResult OEMCrypto_LoadProvisioning(OEMCrypto_SESSION session,
                                           const uint8_t* message,
```

```
                                    size_t message_length,
                                    size_t core_message_length,
                                    const uint8_t* signature,
                                    size_t signature_length,
                                    const uint8_t* wrapped_private_key,
                                    size_t *wrapped_private_key_length);
```

OEMCrypto shall verify the signature using the session's derived server mac key and use the function ODK_ParseProvisioning to parse the provisioning message. After the message has been parsed, OEMCrypto does the same verification and data flow as the v15 function OEMCrypto_RewrapDeviceRSAKey. This is the function that was used for a device that has a keybox (Provisioning 2.0). If the device has an OEM Certificate (Provisioning 3.0) the difference is that the DRM Certificate's private key will be decrypted using the attached message key instead of using the derived encryption key.

```
OEMCryptoResult ODK_ParseProvisioning(const uint8_t* message,
                        size_t message_length,
                        size_t core_message_length,
                        const ODK_NonceValues *nonce_values,
                        const uint8_t *device_id,
                        size_t device_id_length,
                        ODK_ParsedProvisioning* parsed_response);
```

The variables passed in are verified by the ODK function:

- [in] message - pointer to the message buffer.
- [in] message_length - length of the entire message buffer.
- [in] core_message_size - length of the core message, at the beginning of the message buffer.
- [in] nonce_values: pointer to the session's nonce data.
- [in] device_id: a pointer to a buffer containing the device id of the device. The ODK function will verify it matches that in the message.
- [in] device_id_length: the length of the device id.
- [out] parsed_response - destination for the parse data.

```
typedef enum OEMCrypto_PrivateKeyType {
  OEMCrypto_RSA_Private_Key,
  OEMCrypto_ECC_Private_Key,
} OEMCrypto_PrivateKeyType;

typedef struct {
  OEMCrypto_PrivateKeyType key_type;
  OEMCrypto_Substring enc_private_key;
  OEMCrypto_Substring enc_private_key_iv;
} ODK_ParsedProvisioning;
```

Timer and License Renewal Updates

The way OEMCrypto maintains a timer for the license is changing. First, the timer is tied to a license,

or OEMCrypto session -- it is no longer tied to individual keys. For a more detailed description of how the design change is related to the server and CDM changes, please see the document “License Duration and Renewal”.

Most of the new functionality is encapsulated into several new ODK functions and structures. The OEMCrypto implementer is responsible for providing a clock, called the system clock, described below. OEMCrypto shall use the ODK library functions to maintain timer and clock values. There are two new data structures related to duration and time that shall be part of an OEMCrypto Session.

```
ODK_TimerLimits timer_limits;
```

Timer limits are specified in a license and are used to determine when playback is allowed. See the section “Complete ODK API” of the document “Widevine Core Message Serialization” for a complete list of all fields in this structure. The fields are set when OEMCrypto calls the function ODK_ParseLicense or ODK_InitializeV15Values.

```
ODK_ClockValues clock_values;
```

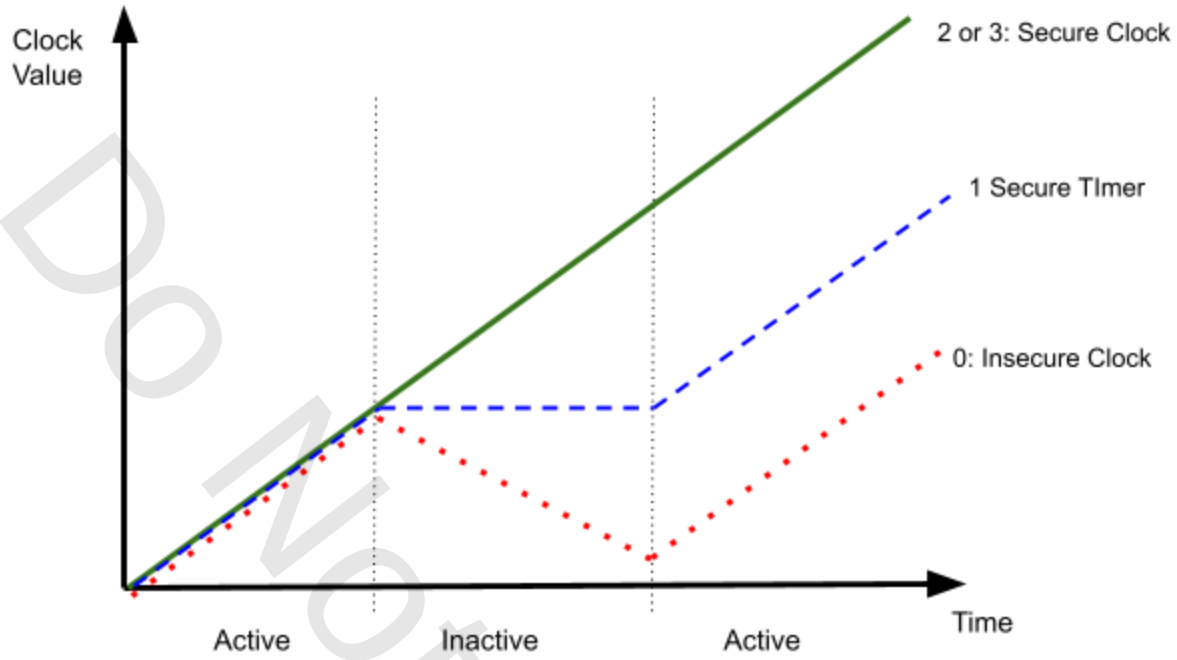
Clock values are modified when decryption occurs or when a renewal is processed. They are used to track the current status of the license -- i.e. has playback started? When does the timer expire? See the section “Complete ODK API” of the document “Widevine Core Message Serialization” for a complete list of all fields in this structure. Most of these values shall be saved with the usage entry.

All times are in seconds. Most of the fields in this structure are saved in the usage entry. This structure should be initialized when a usage entry is created or loaded, and should be used to save a usage entry. It is updated using the ODK functions listed below. The time values are based on OEMCrypto’s system clock, as described in the next section.

Timer Implementation

All vendors are expected to provide a system clock that measures time in seconds. OEMCrypto and the CDM layer do not use this clock for frame display times, so it does not need to be accurate to more than one second. This clock does not have to have a predefined meaning for 0 -- it could be January 1st, 1970 or it could be started on first boot for the device or another start time that is convenient for device maker. However, we expect that the OEMCrypto’s clock never goes backwards in time -- even after a device reboot. In other words, using the definitions below, an insecure clock is not allowed.

- 0 = Insecure Clock - clock just uses system time.
- 1 = Secure Timer - clock runs from a secure timer which is initialized from system time when OEMCrypto becomes active and cannot be modified by user software or the user while OEMCrypto is active. A secure timer cannot run backwards, even while OEMCrypto is not active.
- 2 = Secure Clock - Real-time clock set from a secure source that cannot be modified by user software regardless of whether OEMCrypto is active or inactive. The clock time can only be modified by tampering with the security software or hardware.
- 3 = Hardware Secure Clock - Real-time clock set from a secure source that cannot be modified by user software and there are security features that prevent the user from modifying the clock in hardware, such as a tamper proof battery.



A common way to implement a secure timer is to periodically save the current time to the secure persistent storage. When the system initializes, it will start the secure timer at the last saved clock value. The current time should be saved at least once every five minutes, approximately.

Vendor Provided Timer

Some OEMCrypto implementations have a hardware timer which has better performance than polling a system clock. These vendors may use their own timer to cut off playback. In the ODK functions listed below, if the function returns ODK_SET_TIMER, then the vendor supplied timer should be set to start counting down so that it expires at time clock_values->time_when_timer_expires. If the implementer wishes, they may use parameter timer_value, which will be set to the number of seconds left on the timer. If the function returns ODK_DISABLE_TIMER then the timer should be disabled for this session -- the timer should never expire. If the function returns ODK_TIMER_EXPIRED, then the timer should expire immediately and playback should not be allowed.

Using ODK Timer

If the OEMCrypto implementer does not have a hardware timer, the vendor may rely on the ODK functions to track timer cutoff. These implementers need to call ODK_UpdateLastPlayckTime in order to update the timer value. If the function returns ODK_SET_TIMER or ODK_DISABLE_TIMER, playback may continue. If the function returns ODK_TIMER_EXPIRED, then the timer should expire immediately and playback should not be allowed.

ODK Time Functions

Setting TImer Limits

The following OEMCrypto functions shall set the session's timer limits.

```
OEMCryptoResult OEMCrypto_LoadLicense(...)
OEMCryptoResult OEMCrypto_LoadKeys(...)
```

These functions are described in the document “Widevine Core Message Serialization”. They load or reload a license. OEMCrypto_LoadLIcense shall update the timers by calling the ODK function ODK_ParseLicense. OEMCrypto_LoadKeys shall update the timers by calling ODK_InitializeV15Values. Both of these functions are described in “Widevine Core Message Serialization”.

OEMCrypto shall save the field timer_limits from the parsed_license structure to the session’s data.

Setting and Updating Clock Values

The following functions shall update the session’s clock values.

Initializing Clock Values

The clock values are initialized in the OEMCrypto function

```
OEMCryptoResult OEMCrypto_PrepAndSignLicenseRequest(...)
```

It does this by calling the ODK function

```
OEMCryptoResult ODK_InitializeClockValues(ODK_ClockValues* clock_values,
                                          uint64_t system_time_seconds);
```

This function initializes the license status to unused and starts the rental clock.

Reloading Clock Values

For an offline license, the clock values are reloaded from the usage entry. When the existing OEMCrypto function OEMCrypto_LoadUsageEntry is called, it shall initialize the session’s clock values using the ODK function:

```
void ODK_ReloadClockValues(ODK_ClockValues* clock_values,
                           uint64_t time_of_license_signed,
                           uint64_t time_of_first_decrypt,
                           uint64_t time_of_last_decrypt,
                           enum OEMCrypto_Usage_Entry_Status status,
                           uint64_t system_time_seconds);
```

The time values are taken from the usage entry. The times are based on the system clock.

First Playback -- Enabling Timer

The first time a call is made to OEMCrypto_DecryptCENC, OEMCrypto_Generic_Encrypt, OEMCrypto_Generic_Decrypt, OEMCrypto_Generic_Sign, or OEMCrypto_Generic_Verify for a session, the clock values are updated and the timer may be started. OEMCrypto does this by calling ODK_AttemptFirstPlayback:

```
uint32_t ODK_AttemptFirstPlayback(uint64_t system_time_seconds,
                                  const ODK_TimerLimits* timer_limits,
                                  ODK_ClockValues* clock_values,
                                  uint64_t* timer_value);
```

This updates the clock values, and determines if playback may start based on the given system time. The variables passed in are verified by the ODK function:

- [in] system_time_seconds: the current time on OEMCrypto’s clock, in seconds.

- [in] timer_limits - timer limits specified in the license.
- [in/out] clock_values: the sessions clock values.
- [out] timer_value: set to the new timer value. Only used if the return value is ODK_SET_TIMER.

ODK_AttemptFirstPlayback returns:

- ODK_SET_TIMER: Success. The timer should be reset to the specified value and playback is allowed.
- ODK_DISABLE_TIMER: Success, but disable timer. Unlimited playback is allowed.
- ODK_TIMER_EXPIRED -- Set timer as disabled. Playback is **not** allowed.

During Playback -- Updating Timer

Vendors that do not implement their own timer should call ODK_UpdateLastPlaybackTime regularly during playback. All vendors should call this function from the existing OEMCrypto function OEMCrypto_UpdateUsageEntry.

```
OEMCryptoResult ODK_UpdateLastPlaybackTime(
       const ODK_TimerLimits* timer_limits,
       uint64_t system_time_seconds,
       ODK_ClockValues* clock_values);
```

This function updates the session's clock values and determines if the playback timer has expired. The return values are:

- OEMCrypto_SUCCESS: Success. Playback is allowed.
- ODK_TIMER_EXPIRED: Playback is **not** allowed. The decrypt operation should fail.

Resetting Timer

The OEMCrypto function OEMCrypto_LoadRenewal calls the function OEMCryptoResult ODK_ParseRenewal(...). The OEMCrypto function OEMCrypto_RefreshKeys calls the function ODK_RefreshV15Values. These functions are described in the document "Widevine Core Message Serialization".
ODK_ParseRenewal and ODK_RefreshV15Values return:

- ODK_ERROR_CORE_MESSAGE if the message did not parse correctly, or there were other incorrect values. An error should be returned to the CDM layer.
- ODK_SET_TIMER: Success. The timer should be reset to the specified timer value.
- ODK_DISABLE_TIMER: Success, but disable timer. Unlimited playback is allowed.
- ODK_TIMER_EXPIRED: Set timer as diabled. Playback is **not** allowed.
- ODK_STALE_RENEWAL: This renewal is not the most recently signed. It is rejected.

Usage Entry Updates

When the usage entry is updated, the clock_values should also be updated. The function OEMCrypto_UpdateUsageEntry shall call the function ODK_UpdateLastPlaybackTime described above, and then copy the values from the clock_values to the usage entry.
The OEMCrypto function OEMCrypto_DeactivateUsageEntry shall call the new ODK function

```
OEMCryptoResult ODK_DeactivateUsageEntry(ODK_ClockValues* clock_values);
```

This function updates the status in clock_values.

Provisioning and Elliptic Curve Based Crypto (ECC)

In addition to the core message change discussed above, OEMCrypto is allowed to use (Elliptic Curve Cryptography) ECC instead of RSA for the DRM certificate. License servers will accept either, and the provisioning server will serve either, based on the supported type of certificate.

If a device supports both RSA and ECC, the provisioning server might serve either type of certificate. The type of certificate will be configured at the server.

The function OEMCrypto_SupportedCertificates returns a bit mask of supported certificates. The bit mask includes the following new values:

```
#define OEMCrypto_Supports_ECC_secp256r1   0x100
#define OEMCrypto_Supports_ECC_secp384r1   0x200
#define OEMCrypto_Supports_ECC_secp521r1   0x400
```

There is no change to the function OEMCrypto_GetProvisioningMethod. The server will examine the certificate to see which key algorithm is used.

The factory provisioning functions OEMCrypto_WrapKeyboxOrOEMCert, OEMCrypto_InstallKeyboxOrOEMCert, and OEMCrypto_IsKeyboxOrOEMCertValid do not change their signature. It is the OEM's responsibility to verify that the correct type of OEM Certificate or Keybox is installed.

The function OEMCrypto_GetOEMPublicCertificate is replaced by two functions:

```
OEMCryptoResult OEMCrypto_LoadOEMPrivateKey(OEMCrypto_SESSION session);
```

OEMCrypto shall load the private key associated with the OEM Certificate into the session. For v16, this certificate can only be an RSA key. Support for ECC OEM Certificates is planned for v17.

```
OEMCryptoResult OEMCrypto_GetOEMPublicCertificate(uint8_t* public_cert,
                                                  size_t* public_cert_length);
```

OEMCrypto shall return the public oem certificate. It does **not** load the private key.

The functions OEMCrypto_RewrapDeviceRSAKey and OEMCrypto_RewrapDeviceRSAKey30 will be replaced by OEMCrypto_LoadProvisioning described above in the discussion of core messages. The core message shall contain the private key type of the DRM certificate.

The function OEMCrypto_LoadDeviceRSAKey will be replaced by OEMCrypto_LoadDRMPrivateKey to indicate it can be either an RSA key or an ECC key. It will always be used with a buffer that had been previously wrapped with OEMCrypto_LoadProvisioning.

```
OEMCryptoResult OEMCrypto_LoadDRMPrivateKey(
                OEMCrypto_SESSION session,
                PrivateKeyType key_type,
                const uint8_t* wrapped_private_key,
                size_t wrapped_private_key_length);
```

The new parameter key_type indicates whether this is an RSA or ECC key.

OEMCrypto_PrepAndSignLicenseRequest will use the RSA DRM private key if the DRM certificate is RSA and the ECC DRM private key if it is ECC. It will use ECDSA to sign the license request. Specifics on the signature algorithm will be included in the document “OEMCrypto Elliptic Curve Support”.

The signature of the function OEMCrypto_DeriveKeysFromSessionKey does not change, but the names of the parameters change to indicate they are slightly different when using ECC.

```
OEMCryptoResult OEMCrypto_DeriveKeysFromSessionKey(
     OEMCrypto_SESSION session,
     const uint8_t* derivation_key,
     size_t derivation_key_length,
     const uint8_t* mac_key_context,
     size_t mac_key_context_length,
     const uint8_t* enc_key_context,
     size_t enc_key_context_length);
```

The current implementation is the same for an RSA certificate. The deriviation_key is the session key encrypted by the RSA public key. If the DRM certificate uses an ECC key, then the derivation key is an ephemeral public ECC key. The session key will be the SHA256 of the shared secret key calculated by ECDH between the device’s ECC private key and the derivation_key. See the document “OEMCrypto Elliptic Curve Support”, which will be released this winter, for details.

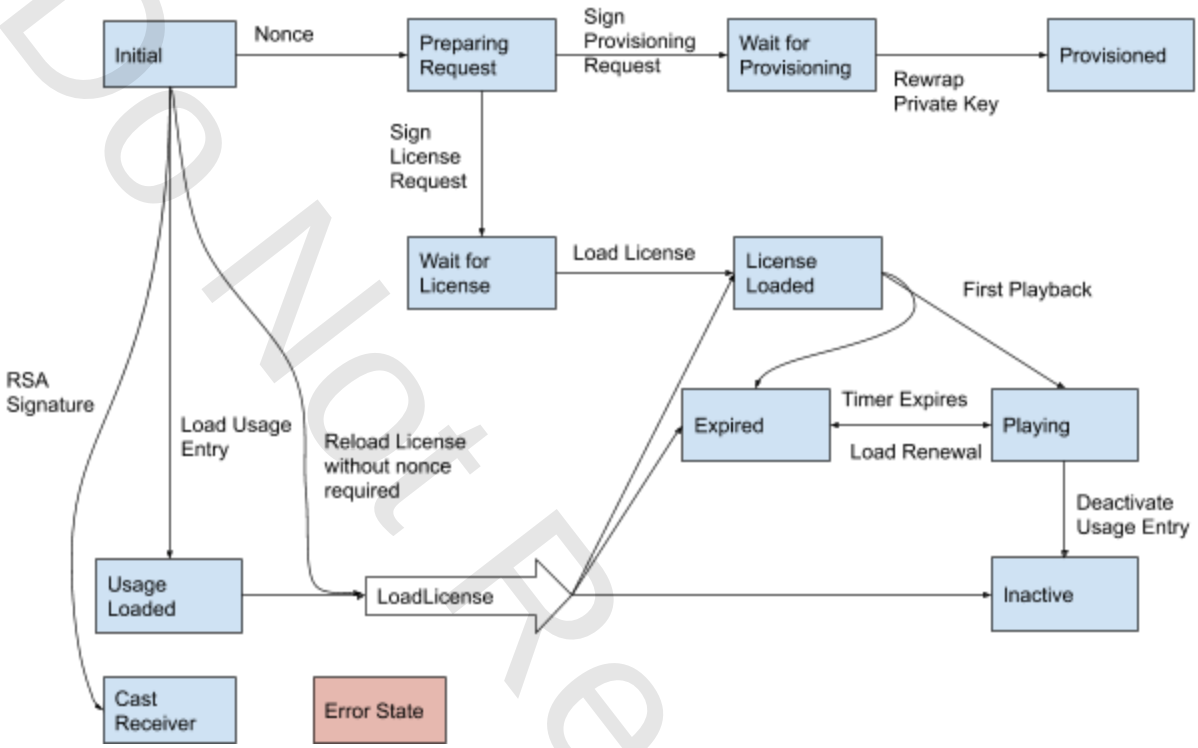
Clarify OEMCrypto States and Remove Unused Functions

To understand proper usage of OEMCrypto functions, it is helpful to think of an OEMCrypto session as a state machine. OEMCrypto implementers are not required to use a state machine. For v16, Widevine does have the following requirements.

1. Only one of OEMCrypto_LoadLicense or OEMCrypto_ReloadLicense can be called in each session, and that function shall only be called once in each session.
2. Only one of OEMCrypto_LoadUsageEntry and OEMCrypto_CreateUsageEntry can be called in each session, and that function shall only be called once in each session.

3. OEMCrypto_GenerateNonce can only be called once in each session, and it must be called before signing either a provisioning request or a license request.

The diagram below shows the possible transitions among each state:



The states for an OEMCrypto Session are:

- **Initial** - when a new session is opened, it starts in the initial state.
- **Preparing_Request** - After a successful call to OEMCrypto_GenerateNonce, the session is being used to prepare a message to a server.
- **Wait_For_Provisioning** - After a successful call to SignProvisioningRequest, the session is waiting for a provisioning response.
- **Provisioned -** After a successful call to LoadProvisioning, the session is provisioned.
- **Wait_For_License** - After a successful call to SignLicenseRequest, the session is waiting for a license response.
- **License_Loaded -** After a successful call to LoadLicense, the session is waiting for playback to begin.
- **Playing -** The session moves to the playing state when the first playback is attempted. If license durations have not expired, the session moves to the playing state.
- **Expired** - After the playback timer expires, the session is in the expired.
- **Inactive** - After a call to DeactivateUsageEntry, the session is holding a license that cannot be used for playback. It may still be used to report usage.
- **Usage_Loaded** - After a successful call to LoadUsageEntry, the session is ready to reload a license or clean up the usage table.

- **Cast_Receiver -** After a call to GenerateRSASignature, the session may only be used to sign cast receiver messages.
- **Error_State** - If any Sign*Request, LoadLicense, ReloadLicense, or RewrapDevice* function fails, the session is in an error state. SImilarly, if an attempt is made to call a function in the wrong state, the session transitions to the error state. The session may only be closed when in this state. This does not include “buffer too small” errors, where a recall to the function is expected.

The following functions may only be called in some states. Functions that cause a transition are in bold.
OEMCrypto_OpenSession - puts session in Initial_State.
OEMCrypto_GenerateNonce - May only be called in Initial state. The session transitions to Preparing_Request state.
OEMCrypto_GenerateDerivedKeys - May only be called in Wait_For_Provisioning state.
OEMCrypto_DeriveKeysFromSessionKey - May only be called in Wait_For_Provisioning or Wait_For_License state.
~~OEMCrypto_GenerateSignature~~ (obsolete)
OEMCrypto_PrepAndSignLicenseRequest - May only be called in Preparing_Request state. The session transitions to Wait_For_License state.
OEMCrypto_PrepAndSignProvisionRequestt - May only be called in Preparing_Request state. The session transitions to Wait_For_Provisioning state.
OEMCrypto_PrepAndSignRenewalRequest - May only be called in License_Loaded, Expired or Playing states.
OEMCrypto_LoadKeys - May only be called once. The session transitions to LIcense_Loaded state.
OEMCrypto_LoadLicense - May only be called once. The session transitions to LIcense_Loaded state.
OEMCrypto_LoadProvisioning - May only be called in Wait_For_Provisioning. The session transitions to Provisioned.
~~OEMCrypto_RewrapDeviceRSAKey30, OEMCrypto_RewrapDeviceRSAKey~~- obsolete.
OEMCrypto_LoadDRMPrivateKey - May only be called in Preparing_Request state.
OEMCrypto_GenerateRSASignature - May only be called in Initial state. The session transitions to Cast_Receiver state.
OEMCrypto_CreateNewUsageEntry - May only be called in Wait_For_License state. It may only be called once.
OEMCrypto_LoadUsageEntry - May only be called in Initial state. The session transitions to Usage_Loaded state.

The following functions may be called in License_Loaded, Expired, Playing or Inactive states:
OEMCrypto_LoadEntitledContentKeys
OEMCrypto_SelectKey
OEMCrypto_QueryKeyControl
OEMCrypto_DeactivateUsageEntry - The session and usage entry transitions to Inactive state.

OEMCrypto_RefreshKeys - May only be called in License_Loaded, Expired, or Playing state.

OEMCrypto_LoadRenewal - May only be called in License_Loaded, Expired, or Playing state.

The following functions may be called in License_Loaded or Playing state:
OEMCrypto_DecryptCENC, OEMCrypto_Generic_Encrypt, OEMCrypto_Generic_Decrypt, OEMCrypto_Generic_Sign, OEMCrypto_Generic_Verify. When called from License_Loaded state, the call is treated as a first decrypt for the session. The session transitions to either Playing or Expired depending on the state of the session's timers.

OEMCrypto_MoveEntry - May only be called in Usage_Loaded state.

The following functions may be called in any state in which a usage entry has been loaded or create:
OEMCrypto_ReportUsage
OEMCrypto_UpdateUsageEntry

The following functions may be called in any state:
OEMCrypto_CopyBuffer
OEMCrypto_LoadSRM
OEMCrypto_GetKeyData
OEMCrypto_GetDeviceID
OEMCrypto_GetProvisioningMethod
OEMCrypto_APIVersion
OEMCrypto_BuildInformation
OEMCrypto_Security_Patch_Level
OEMCrypto_SecurityLevel
OEMCrypto_GetHDCPCapability
OEMCrypto_SupportsUsageTable
OEMCrypto_IsAntiRollbackHwPresent
OEMCrypto_GetNumberOfOpenSessions
OEMCrypto_GetMaxNumberOfSessions
OEMCrypto_SupportedCertificates
OEMCrypto_IsSRMUpdateSupported
OEMCrypto_GetCurrentSRMVersion
OEMCrypto_GetAnalogOutputFlags
OEMCrypto_ResourceRatingTier
OEMCrypto_CloseSession

The following functions are special:
OEMCrypto_SetSandbox -- only called on system initialization.
OEMCrypto_Initialize -- only called on system initialization.
OEMCrypto_Terminate -- only called on system termination.
OEMCrypto_WrapKeyboxOrOEMCert - only called in the factory.
OEMCrypto_InstallKeyboxOrOEMCert -- only called on system initialization.
OEMCrypto_LoadTestKeybox -- only called on system initialization.
OEMCrypto_IsKeyboxOrOEMCertValid -- only called on system initialization.

OEMCrypto_LoadTestRSAKey -- only called on system initialization.
OEMCrypto_CreateUsageTableHeader -- not a session function.
OEMCrypto_LoadUsageTableHeader -- not a session function.
OEMCrypto_DeleteOldUsageTable -- not a session function.
OEMCrypto_CopyOldUsageEntry -- only called on system initialization
OEMCrypto_ShrinkUsageTableHeader -- not a session function.
OEMCrypto_GetOEMPublicCertificate - modified version is not a session function.

The following functions are obsolete:
OEMCrypto_GenerateSignature - replaced.
OEMCrypto_RewrapDeviceRSAKey30 - replaced,
OEMCrypto_RewrapDeviceRSAKey - replaced.
OEMCrypto_LoadDeviceRSAKey - replaced.

Nonce Table

Each OEMCrypto session shall have only one nonce. The nonce table can be replaced with a single nonce. The function OEMCrypto_GenerateNonce shall only be called only once per session. Nonce Flood protection from v15 is still required. The nonce will be used in either a provisioning request/response pair or a license request/response pair. A renewal request/response pair will use the same nonce as that in the license so that the renewal is tied to that instance of the license. There is no longer a need to send multiple license requests or provisioning requests for a single session. To help prevent attacks based on the birthday paradox for simultaneous sessions, provisioning and license messages will also include the current session number. See the section on serializing the core message for more details.

The function OEMCrypto_GenerateNonce shall call the following:

```
OEMCryptoResult ODK_SetNonceValue(ODK_NonceValues *nonce_values, uint32_t nonce);
```

The parameters passed into this function are:

- [in/out] nonce_values: the session's nonce data.
- [in] nonce: the new nonce that was just generated.

We still require nonce flood prevention specified in the existing OEMCrypto API, but the number of nonces allowed per second is **increasing from 20 to 200**. This is to account for the increased number of sessions required for some resource rating tiers. Since some devices support 100 sessions open simultaneously, an application can expect to generate at least 100 nonces for 100 license requests. The number 200 is to allow some wiggle room.

Separate Signing Keys for Renewal and License Request

A license response includes a pair of mac keys that are used to sign and verify license renewals. They are also used in the usage table. These should never be used to sign a license request or verify a license response. The pair of mac keys used for signing the license request/response or provisioning request/response are generated by either OEMCrypto_GenerateDerivedKeys or OEMCrypto_DeriveKeysFromSessionKey. These keys should be deleted in the function OEMCrypto_LoadLicense or OEMCrypto_LoadProvisioning. A new pair of mac keys are loaded in OEMCrypto_LoadLicense for use in signing the renewal message.

Usage Table Size is Queryable

There will be a new function that estimates the maximum size of the usage table.

```
size_t OEMCrypto_MaximumUsageTableHeaderSize();
```

If the device does not have a fixed size, this returns an estimate. A maximum size of 0 means the header is constrained only by dynamic memory allocation.

Widevine is also increasing the required size to at least 300 entries.

Combine Decrypt Calls -- Performance Improvement

The function OEMCrypto_DecryptCENC will be modified to allow the application to send several entire samples at once. Instead of a single subsample, an array of input and output buffers will be sent, and each pair will have an array of subsample maps.

Currently, the Android CDM API accepts multiple subsamples and a subsample map, but then breaks it up into one decrypt call per-subsample. The CE CDM API only accepts one subsample at a time and requires the application to separate the subsamples and make multiple calls. This change would require changes to the CE CDM API.

The DecryptCENC function will now have the following prototype.

```
OEMCryptoResult
OEMCrypto_DecryptCENC(OEMCrypto_SESSION session,
                      const OEMCrypto_SampleDescription* samples, // an array of
                      samples.
                      size_t samples_length, // the number of samples.
                      const OEMCrypto_CENCEncryptPatternDesc* pattern
                      );

typedef struct {
  const uint8_t* input_data; // source for protected data.
  size_t input_data_length;  // length of protected data.
  OEMCrypto_DestBufferDesc output_descriptor; // destination for clear data.
} OEMCrypto_InputOutputPair;
```

```
typedef struct {
  size_t num_bytes_clear;
  size_t num_bytes_encrypted; // number of protected bytes.
  uint8_t subsample_flags;  // is this the first/last subsample in a sample?
  size_t block_offset;  // used for CTR “cenc” mode only.
} OEMCrypto_SubSampleDescription;

typedef struct {
  OEMCrypto_InputOutputPair buffers;  // The source and destination for this sample.
  uint8_t iv[16];  // The IV for the initial subsample.
  const OEMCrypto_SubSampleDescription* subsamples; // an array of subsamples.
  size_t subsamples_length;  // the number of subsamples in the sample.
} OEMCrypto_SampleDescription;
```

The IV is specified for the initial subsample in a sample. The IV should be incremented according to the ISO-CENC 3.0 specification, ISO/IEC 23001-7:2016. Section 9.5.2.3 covers 'cenc' and 'cens'. Section 9.5.2.4 covers 'cbc1'. And section 9.5.2.5 covers 'cbcs'.

Here is the old signature for comparison:

```
OEMCryptoResult
OEMCrypto_DecryptCENC(OEMCrypto_SESSION session,
                      const uint8_t *data,
                      size_t data_length,
                      bool is_encrypted,
                      const uint8_t *iv,
                      size_t block_offset, // used for CTR “cenc” mode only.
                      OEMCrypto_DestBufferDesc* out_buffer_descriptor,
                      const OEMCrypto_CENCEncryptPatternDesc* pattern,
                      uint8_t subsample_flags);
```

If OEMCrypto returns an error of OEMCrypto_ERROR_BUFFER_TOO_LARGE, then the caller will break the samples array into single samples and DecryptCENC will be called again for each one. If OEMCrypto returns OEMCrypto_ERROR_BUFFER_TOO_LARGE for any of these individual samples, then the sample will be broken into single subsamples and DecryptCENC will be called again for each one. If it returns OEMCrypto_ERROR_BUFFER_TOO_LARGE for any of these individual subsamples, then the subsample will be broken into smaller subsamples following the same behavior as in OEMCrypto v15. For best performance, implementers are encouraged to accept full samples in a single decrypt call. However, an implementer who is not concerned with performance can return OEMCrypto_ERROR_BUFFER_TOO_LARGE whenever subsample_length > 1 or samples_length > 1.

Each subsample has flags indicating if it is the first or last subsample in the sample. When passing full samples to DecryptCENC, these flags are redundant, since the first subsample in the array will always have the “first subsample” flag set and the last subsample in the array will always have the “last subsample” flag set. This will always be true unless OEMCrypto returns OEMCrypto_ERROR_BUFFER_TOO_LARGE enough times that a sample is broken into multiple subsamples that are passed in separate DecryptCENC calls. When subsamples are sent to OEMCrypto in separate DecryptCENC calls, the subsample array will only contain one subsample, but

the subsamples will still be flagged correctly as “first” or “last” as appropriate for their position in the full sample.

Clean up Error Codes

Specify Behavior When the License Contains Too Many Keys

A new error OEMCrypto_ERROR_TOO_MANY_KEYS will be returned if an attempt is made to load a license with too many keys for a single session.

Reloading License

An attempt to load two licenses into the same session shall return the new error code OEMCrypto_ERROR_LICENSE_RELOAD.

Multiple Usage Entries

Loading or creating a usage table entry into a session with an existing entry shall return the new error code OEMCrypto_ERROR_MULTIPLE_USAGE_ENTRIES.

Core Message Errors

When a core message does not parse correctly, or has incorrect fields, then an error of ODK_ERROR_CORE_MESSAGE. This will be returned by the ODK function.

Multiple Screen Warnings

If a call to OEMCrypto_DecryptCENC uses a key that is only allowed on some connected displays, but not all, the device may continue to decrypt as long as content is **not** sent to the restricted displays. For example, consider a phone that has a local display and is connected via a display port to an HDCP 1.0 device. If the key requires HDCP 2.0, the output shall not be sent to the display port. The device may display the output on the local display and return the warning OEMCrypto_WARNING_MIXED_OUTPUT_PROTECTION.

Devices that do not have the ability to disable the display port while still sending output to the local display shall return OEMCrypto_ERROR_INSUFFICIENT_HDCP and not display any content. This is the behavior for OEMCrypto v15 and earlier.

ODK Return Values

The following return codes are from ODK functions, and should be handled by OEMCrypto:

- ODK_SET_TIMER
- ODK_DISABLE_TIMER
- ODK_TIMER_EXPIRED

The following return codes are from ODK functions, and indicate an error. OEMCrypto shall pass them up to the CDM layer.

- ODK_ERROR_CORE_MESSAGE

Delayed Error Conditions -- Performance Improvement

On some devices, the HDCP subsystem is not directly connected to the OEMCrypto TA. This means that returning the error OEMCrypto_ERROR_INSUFFICIENT_HDCP at the time of the decrypt call is a performance hit. However, some devices have the ability to tag output buffers with security requirements, such as the required HDCP level.

For those devices, when a call to OEMCrypto_DecryptCENC is made using a key that requires HDCP output, and if the HDCP level on the output does not meet the required level.

- OEMCrypto may tag the output buffer as requiring HDCP at the required level and return OEMCrypto_SUCCESS.
- Output shall not be sent to the display.
- On the second or third call to OEMCrypto_DecryptCENC with the same key, OEMCrypto shall return OEMCrypto_ERROR_INSUFFICIENT_HDCP.

For those devices, when a call to OEMCrypto_DecryptCENC is made using a key that requires HDCP output, and if the HDCP level on some of the displays does not meet the required level.

- OEMCrypto may tag the output buffer as requiring HDCP at the required level and return OEMCrypto_SUCCESS.
- Output shall only be sent to the display with sufficient output control, e.g. the local display.
- On the second or third call to OEMCrypto_DecryptCENC with the same key, OEMCrypto shall return OEMCrypto_WARNING_MIXED_OUTPUT_PROTECTION.

In either case, a call to OEMCrypto_GetHDCPCapability shall return the current HDCP level.

New Resource Rating Tier

There will be a fourth resource tier which allows buffer sizes that are needed to support 8k content at a new Resource Rating tier.

Also, there are stricter requirements for devices that support the AV1 codec. The AV1 codec has many more subsamples per sample than other codecs for similar video resolution.

The message size that is needed for a license with a large number of keys is larger than in previous versions. The message size limit applies to all functions that sign or verify messages. It also applies to the size of context buffers in the derive key functions.

Resource Rating Tier	1 - Low	2 - Medium	3 - High	4 - Very 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.	An 8k television or home entertainment device would probably be in this tier.
Minimum Sample size	1 MiB	2 MiB	4 MiB	16 MiB
Minimum Number of Subsamples - H264 or HEVC	10	16	32	64
Minimum Number of Subsamples - VP9	9	9	9	9
Minimum Number of Subsamples - AV1	72	144	288	576
Minimum subsample buffer size	100 KB	500 KB	1 MiB	4 MiB
Minimum Generic crypto buffer size	10 KB	100 KB	500 KB	1 MiB
Minimum Number of concurrent sessions	10	20	30	40
Minimum Number of keys per session	4	20	20	30
Minimum Total Number of Keys (all sessions)	16	40	80	90
Message Size	8 KiB	8 KiB	16 KiB	32 KiB
Decrypted Frames per Second	30 fps SD	30 fps HD	60 fps UHD	60 fps at 8k

Increase Number of Keys Per Session

Since OEMCrypto v11, we have required 20 keys per session at a minimum. This number was arrived at by adding up how many keys we thought a single piece of content might require across audio/video, different resolution tiers, HDR/non-HDR, and so on. Then we multiplied by 3 for live content needing “previous,” “current,” & “next” key sets. And finally, we round up to reach a nice round number. Adding this generous and future-looking minimum has paid us dividends because as new content requiring more keys has come out, we’ve been able to support it even on older devices.

Some content providers wish to use key rotation and also have up to 9 keys per time segment. If we multiply by 3 and round up, a session should support 30 keys.

This requirement is for resource rating tier 4, only.

There will also be a requirement on how many total keys shall be loaded simultaneously. This number is lower than the total number of sessions. This allows OEMCrypto to support an application opening many sessions and requesting licenses with only a few keys, or opening a few sessions with many keys.

Resource Rating Tier	**1 - Low**	**2 - Medium**	**3 - High**	**4- Very High**
Number of concurrent sessions	10	20	20	30
Number of keys per session	4	20	20	30
Total Number of Keys	16	40	80	90
	16 < 10*4	40 < 20*20	80 < 20*20	90 < 30*30

Clarify HDCP 2.3 Even More Clearly

Previous attempts to clarify HDCP requirements were not correct. Version 2.2 and 2.3 are indistinguishable across HDMI/DP etc. The new wording should be, with the change in bold:

> 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 **2.2 or 2.3** receiver or repeater.

Clarify that Implementations Cannot Rely on Threading Guarantees for Security

Even though we make guarantees to partners about how the CDM layer will and won't call OEMCrypto as regards threading, partners cannot rely on only being called in that way for security, as malicious actors could call OEMCrypto in other ways. At best, our Threading Guarantees say "If we call you in this manner, you should not have errors because of threading." OEMCrypto must not leak or become compromised if they are called outside those guarantees. The guarantees merely permit them to fail to work correctly when called outside those bounds.

Clarify Robustness Rules

If the primary bootloader for the Trusted Execution Environment is signed and trusted, then OEMCrypto can run and use the keybox. If not, then the keybox/cert and other keys should be disabled.

This does not apply to a secondary boot loader that loads an OS outside of the TZ. For example, it is OK to run an Android Open Source Project (AOSP) OS as long as the TZ is still secure.

Relax Requirement on CENC Patterns

Widevine will not require all four CENC decryption formats to be supported. All platforms are required to support "cenc" and "cbcs" modes. As a reminder:

- "cenc" is AES Counter mode, with no pattern.
- "cbcs" is AES CBC mode, with a pattern.

Function Prototype Changes

The following function's prototype is changing to be consistent with other functions. The lengths are changing from uint32_t to size_t:

```
OEMCryptoResult OEMCrypto_GenerateDerivedKeys(OEMCrypto_SESSION session,
                                              const uint8_t* mac_key_context,
                                              size_t mac_key_context_length,
                                              const uint8_t* enc_key_context,
                                              size_t enc_key_context_length);
```

The output buffer description in OEMCrypto_CopyBuffer should not be modified by OEMCrypto, so the parameter now has a const modifier. The data to which the descriptor points, of course, should be modified by OEMCrypto.

```
OEMCryptoResult OEMCrypto_CopyBuffer(
    OEMCrypto_SESSION session,
    const SharedMemory* data_addr,
    size_t data_addr_length,
    const OEMCrypto_DestBufferDesc* out_buffer_descriptor,
    uint8_t subsample_flags);
```

The variable type SharedMemory will be used to indicate a buffer that can be shared by OEMCrypto in

the TEE and code in the REE. Buffers that are not pointers to SharedMemory shall be protected from being written to by the REE while OEMCrypto is processing the buffer.

```
typedef uint8_t SharedMemory;
```

Wrapped RSA Key Padding

The private key in the provisioning message is encrypted with AES with PKCS#7 padding. This key is decrypted and verified in the function OEMCrypto_LoadProvisioning. Before v16, it was decrypted and verified in the function OEMCrypto_LoadProvisioning. To avoid a padding oracle attack, the padding should NOT be verified. Instead, the RSA key itself should be verified, and the signature of the entire message should be verified.

Create Secure Buffers for Testing

In order to run unit tests with secure buffers, the following two new functions should be implemented. These functions are optional, and will only be used by the unit tests to verify decryption to secure buffers. If they return OEMCrypto_ERROR_NOT_IMPLEMENTED, then the unit tests will only use clear buffers for decryption tests.

```
OEMCryptoResult OEMCrypto_AllocateSecureBuffer(
   OEMCrypto_SESSION session,
   size_t buffer_size,
   OEMCrypto_DestBufferDesc *output_descriptor,
   int *secure_fd);
```

Allocates a secure buffer and fills out the destination buffer information in output. The integer secure_fd may also be set to indicate the source of the buffer. OEMCrypto may use the secure_fd to help track the buffer if it wishes. The unit tests will pass a pointer to the same destination buffer description and the same secure_fd to OEMCrypto_FreeSecureBuffer when the buffer is to be freed.

Valid return values are:
OEMCrypto_ERROR_NOT_IMPLEMENTED - if this is returned, secure buffer unit tests will be skipped.
OEMCrypto_SUCCESS - the buffer was allocated.
OEMCrypto_ERROR_BUFFER_TOO_LARGE - the buffer was too large.
OEMCrypto_ERROR_INSUFFICIENT_RESOURCES - if too many buffers have been allocated.
OEMCrypto_ERROR_UNKNOWN_FAILURE - any other failure.

```
OEMCryptoResult OEMCrypto_FreeSecureBuffer(
   OEMCrypto_SESSION session,
   const OEMCrypto_DestBufferDesc *output_descriptor,
   int secure_fd);
```

Frees a secure buffer that had previously been created with OEMCrypto_AllocateSecureBuffer. Any return value except OEMCrypto_SUCCESS will cause the unit test using secure buffers to fail.