WV Modular DRM Version 13.1 Delta

Changes from Version 12 to 13.1

Jan. 31th, 2016

Table of 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 v12 library, and want to upgrade to OEMCrypto v13, then this document is for you. If you are starting from scratch, you should read WV Modular DRM Security Integration Guide for Common Encryption (CENC).

Overview

There are several new features required for OEMCrypto version 13. 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 …".

Definitions

CENC - Common Encryption

DASH - Dynamic Adaptive Streaming over HTTP

DCP - Digital Content Protection (https://digital-cp.com/)

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.

API Version Number

```
uint32_t OEMCrypto_APIVersion();
```

This function should now return 13. If it returns less than 13, then the calling code will assume that OEMCrypto does not support the new v13 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.

Support RSA keys with 3072 bits

Widevine plans to update the provisioning and license servers to support 3072 bit RSA keys for the DRM and OEM Certificates. Because some low end devices may not be able to support 3072 bit keys at a reasonable performance level, we are adding a new API:

```
uint32_t OEMCrypto_SupportedCertificates();
```

Returns the bitwise or of the following flags. It is likely that high end devices will support both 2048 and 3072 bit keys while the widevine servers transition to new key sizes.

- 0x1 = OEMCrypto_Supports_RSA_2048bit - the device can load a DRM certificate with a 2048 bit RSA key.
- 0x2 = OEMCrypto_Supports_RSA_3072bit - the device can load a DRM certificate with a 3072 bit RSA key.
- 0x10 = OEMCrypto_Supports_RSA_CAST - the device can load a CAST certificate. These certificate are used with OEMCrypto_GenerateRSASignature with padding type set to 0x2, PKCS1 with block type 1 padding.

HDCP SRM Update

Some content providers are requesting that Widevine deliver the HDCP SRM (System Renewability Message). This is a small file, currently about 5kB, that contains lists of Key Selection Vectors (i.e. key IDs) that should not be negotiated for HDCP. The device is supposed to validate the signature on the SRM, store the SRM in non-volatile memory and use it during authentication to decide if a downstream device is allowed to receive content, as required by the HDCP specification.

The SRM is signed by the DCP private key, and must be verified by the device. Each SRM has a version number, and the device must not install a less recent version of the file. This makes testing this feature problematic. With that in mind, the SRM update functions will only be superficially tested by the standard suite of unit tests. See the discussion about the function RemoveSRM below for more information.

In addition to functions for installing the SRM, the key control block will include a new field, the minimum SRM version number, **MinimumSRMVersion**. For content that requires HDCP version 2 or above, the device shall refuse to decrypt the content if the current SRM has a lower version number.

This only applies to external output -- the minimum SRM version will be ignored for content that is displayed on an internal display or when HDCP is not required.

New SRM Update Functions

The following functions should be implemented for all devices that support HDCP v2.2 or higher. For devices that support an SRM, the license may specify a minimum SRM version number. Some keys will be marked as requiring this SRM version in the key's control block. If the device does not meet this restriction, OEMCrypto will not allow those keys to be used for digital output. If the device supports updating the SRM via OEMCrypto, but it does not yet have a current SRM, the license may contain an updated SRM. The CDM layer will first install the new SRM before calling LoadKeys.

```
bool OEMCrypto_IsSRMUpdateSupported()
```

Returns true if the device supports SRM files and the file can be updated via this OEMCrypto API. This returns false for devices that do not support an SRM file, devices that do not support HDCP, and devices that have no external display support.

```
OEMCryptoResult OEMCrypto_GetCurrentSRMVersion(uint16_t*  version);
```

Returns the version number of the current SRM file. If the device does not support SRM files, this will return OEMCrypto_ERROR_NOT_IMPLEMENTED. If the device only supports local displays, it would return OEMCrypto_LOCAL_DISPLAY_ONLY. If the device has an SRM, but cannot use OEMCrypto to update the SRM, then this function would set version to be the current version number, and return OEMCrypto_SUCCESS, but it would return false from OEMCrypto_IsSRMUpdateSupported.

```
OEMCryptoResult OEMCrypto_LoadSRM(const uint8_t* buffer,
                                  size_t buffer_length);
```

Verify and install a new SRM file. The device shall install the new file only if verification passes. If verification fails, the existing SRM will be left in place. Verification is defined by DCP, and includes verification of the SRM's signature and verification that the SRM version number will not be decreased. This may return:
OEMCrypto_SUCCESS - if the file was valid and was installed.
OEMCrypto_ERROR_INVALID_CONTEXT - if the SRM version is too low, or the file is corrupted.
OEMCrypto_ERROR_SIGNATURE_FAILURE - If the signature is invalid.
OEMCrypto_ERROR_BUFFER_TOO_LARGE - if the buffer is too large for the device.
OEMCrypto_ERROR_NOT_IMPLEMENTED

The SRM Version number will be passed into OEMCrypto_LoadKeys within the SRMRequirement, which is a new parameter:

```
OEMCryptoResult OEMCrypto_LoadKeys(...same  as before..,
                           const uint8_t* srm_requirement)
```

The pointer srm_requirement points to a block of data in the license message. The data is 12 bytes and contains:

```
struct SRMRequirement {
  uint8_t verification[8];   // must be "HDCPDATA"
  uint32_t  minimum_srm_version;   // version number in network byte order.
}
```

The key control block will have a new bit flag, **SRMVersionRequired**. If any key has this bit set, then OEMCrypto will require that srm_requirement is not null. OEMCrypto will verify that the device's current SRM is at least as high as the version in the srm_requirement. If it is not, then the key will be marked for use with a local display only -- no digital output. The key will behave as if the key control block field HDCP_Version is 0xF, "Local Display Only". If the key is selected while output is destined for an external display, OEMCrypto_SelectKey and OEMCrypto_DecryptCENC will return OEMCrypto_ERROR_INSUFFICIENT_HDCP.

Be aware that some content providers wish to require HDCP but do not wish to require a minimum SRM. The key control block flags SRMVersionRequired may be set or may be unset for various values of HDCP_Version. If SRMVersionRequired is not set, then the device should NOT enforce the SRM blacklist. This can be used to bypass a compromised SRM that has been installed on a device by a rogue entity at the discretion of the content provider.

In order to test this functionality, it will be necessary to install a new SRM file. In order to run several tests, or to run the test several times, the device will need to delete the SRM file. This functionality should **not** be available on production devices. Widevine will create a brief set of unit tests which will use this function. The OEM will need to take extra care verifying this feature, because there will be no automated tests.

```
OEMCryptoResult OEMCrypto_RemoveSRM();
```

Delete the current SRM. Any valid SRM, regardless of version number, will be installable after this. This function would not be implemented on production devices, and would only be used to verify unit tests on a test device.
This may return:
OEMCrypto_ERROR_NOT_IMPLEMENTED - always on production devices.
OEMCrypto_SUCCESS - if the SRM file was deleted.

Because this feature is very device dependent, it will be the OEM's responsibility to verify SRM files are updated correctly. The extra unit tests will only be able to verify that

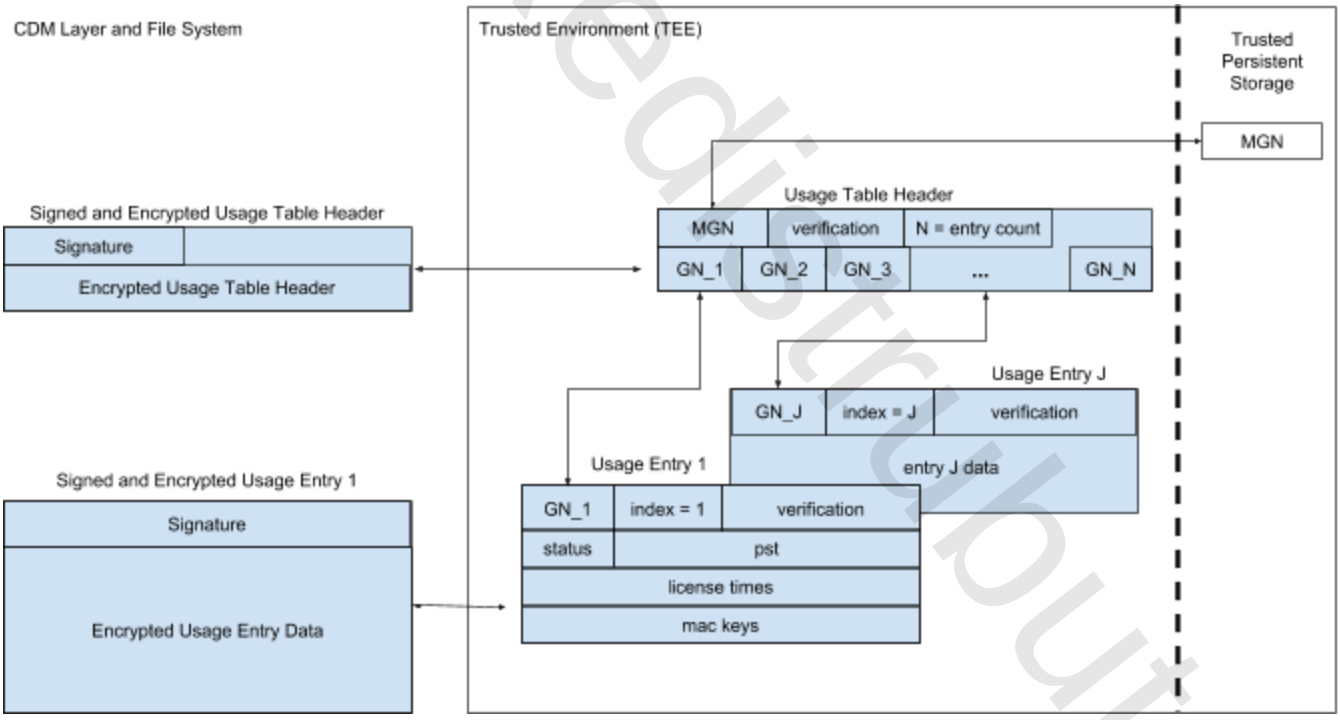
1. OEMCrypto_LoadSRM can load one of the existing sample files distributed by http://digital-cp.com/.
2. OEMCrypto_LoadSRM will return an error if an SRM file with a bad signature is loaded.
3. OEMCrypto_LoadSRM will return an error if a valid SRM file with lower version is loaded.
4. OEMCrypto_LoadSRM will return success if a valid SRM with a higher version is loaded.
5. SelectKey will return an error if the key control block requires a higher SRM version than is currently loaded.

Big Usage Tables

Some content providers are thinking it would be nice to prefetch many offline licenses. The number 200 has been bandied about. Some devices currently have some performance problems saving the usage table because the TEE is single threaded. This is causing enough performance issues, that some content providers are avoiding using the usage table.

To solve this, we are planning to have a variable sized usage table, with only active entries resident in memory. The usage table will be stored on the file system instead of in secure storage. The usage tables will be saved to the file system by the CDM layer, and OEMCrypto will be responsible for signing, encrypting and maintaining generation numbers.

The CDM layer will be responsible for making sure that applications do not access each other's usage table entries. This will prevent content providers from evicting usage entries from other applications. Each entry in the usage table will be encrypted and signed by OEMCrypto, and stored as a separate file by the CDM layer. To prevent rollback of license data, each entry will have a generation number. The whole table will also have a master generation number. Each entry's generation number will be stored with the entry, and will also be stored with the usage table header. Whenever an entry is updated, the header is also updated and both that entry, and the header will be encrypted and written to the file system by the CDM layer. The master generation number that is stored in the header will also be stored in secure nonvolatile memory by OEMCrypto.



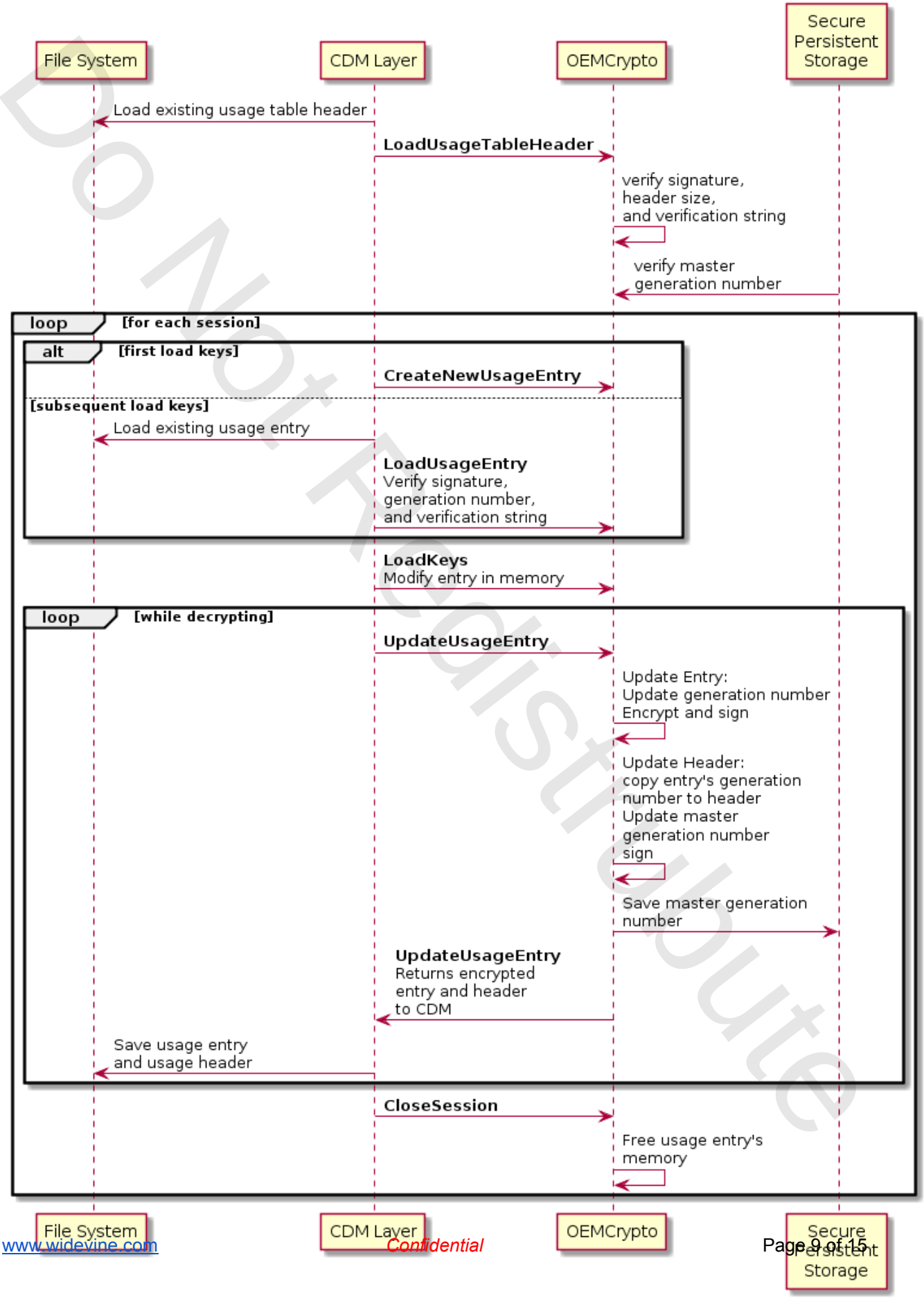
Notice that license times and mac keys are described in the existing documentation for usage tables. These values, and their usage is not changed from OEMCrypto v12 to v13.

Before loading any license that requires a usage entry, the CDM layer will install the Usage Table Header. OEMCrypto will verify that the header was encrypted and signed by OEMCrypto, and that the

verification field is correct. OEMCrypto will store the Usage Table in volatile memory and update its master generation number whenever any usage entry is updated. Whenever the Usage Table's master generation number is updated, it will also be stored in trusted persistent storage. This still requires trusted persistent storage, but it reduces the amount of storage to 8 bytes. The CDM layer will also install each usage entry that will be used by an open session. Each usage entry will also have a generation number, and that generation number must match the entry in the usage table header.

Whenever a usage entry must be updated, the CDM layer will pass buffers to OEMCrypto for storage of the entry and the header. OEMCrypto will update that entry and increment its generation number. It will then store that generation number in the header. It will then update the usage table's master generation number, and store the master generation number in secure storage. OEMCrypto will then encrypt and sign the entry data into the buffer that was given it by the CDM layer. OEMCrypto will also encrypt and sign the header and store it in the other buffer given it by the CDM layer. After this, the CDM layer will save these buffers to the filesystem.

Big Usage Tables



The following functions will be created or modified:

```
OEMCryptoResult  OEMCrypto_CreateUsageTableHeader( uint8_t* header_buffer,
                                                   size_t* header_buffer_length);
```

This creates a new Usage Table Header with no entries. If there is already a generation number stored in secure storage, it will be incremented by 1 and used as the new Master Generation Number. This will only be called if the CDM layer finds no existing usage table on the file system.

```
OEMCryptoResult  OEMCrypto_LoadUsageTableHeader(const  uint8_t* buffer,
                                                size_t buffer_length);
```

This loads the Usage Table Header. The buffer's signature is verified and the buffer is decrypted. OEMCrypto will verify the verification string. If the Master Generation Number is more than 1 off, the table is considered bad, the headers are NOT loaded, and the error OEMCrypto_ERROR_GENERATION_SKEW is returned. If the generation number is off by 1, the warning OEMCrypto_WARNING_GENERATION_SKEW is returned but the header is still loaded. This warning may be logged by the CDM layer.
This may return:
OEMCrypto_ERROR_UNKNOWN_FAILURE
OEMCrypto_WARNING_GENERATION_SKEW - if the generation number is off by exactly 1.
OEMCrypto_ERROR_GENERATION_SKEW - if the generation number is off by **more** than 1.
OEMCrypto_ERROR_SIGNATURE_FAILURE - if the signature failed.
OEMCrypto_ERROR_BAD_MAGIC - verification string does not match.
Entries are created or loaded using:

```
OEMCryptoResult  OEMCrypto_CreateNewUsageEntry( OEMCrypto_SESSION session,
                                                uint32_t *usage_entry_number)
```

This creates a new usage entry. The size of the header will be increased by 8 bytes, and secure volatile memory will be allocated for it. The new entry will be associated with the given session. The status of the new entry will be set to "unused". OEMCrypto will set *usage_entry_number to be the index of the new entry. The first entry created will have index 0. The new entry will be initialized with a generation number equal to the master generation number, which will also be stored in the header's new slot. Then the master generation number will be incremented. Since each entry's generation number is less than the master generation number, the new entry will have a generation number that is larger than all other entries and larger than all previously deleted entries. This helps prevent a rogue application from deleting an entry and then loading an old version of it.
This may return:
OEMCrypto_SUCCESS
OEMCrypto_ERROR_INSUFFICIENT_RESOURCES if there is no room in memory to increase the size of the usage table header. The CDM layer can delete some entries and then try again, or it can pass the error up to the application.

```
OEMCryptoResult  OEMCrypto_LoadUsageEntry( OEMCrypto_SESSION session,
                                           uint32_t index,
                                           const uint8_t *buffer,
                                           size_t buffer_length)
```

This loads a usage table saved previously by UpdateUsageEntry. The signature at the beginning of the buffer is verified and the buffer will be decrypted. Then the verification field in the entry will be verified. The index in the entry must match the index passed in. The generation number in the entry will be compared against that in the header. If it is off by 1, a warning is returned, but the entry is still loaded. This warning may be logged by the CDM layer. If the generation number is off by more than 1, an error is returned and the entry is not loaded.
This may return:
OEMCrypto_ERROR_INSUFFICIENT_RESOURCES if there is not enough memory to hold the new entry.
OEMCrypto_WARNING_GENERATION_SKEW - if the generation number is off by exactly 1.
OEMCrypto_ERROR_GENERATION_SKEW - if the generation number is off by **more** than 1.
OEMCrypto_ERROR_SIGNATURE_FAILURE if the signature is bad.

```
OEMCryptoResult OEMCrypto_LoadKeys(OEMCrypto_SESSIONsession, ...
```

This behaves as before, except regarding usage table entries. If the session has been associated with a usage entry, using LoadUsageEntry or CreateNewUsageEntry, then LoadKeys will expect a nonempty pst. Conversely, if there is a nonempty pst, then LoadKeys will expect the session to be associated with a usage table entry.

If the session was associated with an entry using CreateNewUsageEntry, then the pst will be copied to the entry. The license received time of the entry will be updated. In this case, the nonce will be validated as usual.

If the sesion was associated with an entry using LoadUsageEntry, then the pst will be compared with the one already in the entry. If the pst does not match, OEMCrypto_ERROR_WRONG_PST is returned. Also, if the mac keys do not match, then OEMCrypto_ERROR_WRONG_KEYS is returned.

If any key control block requires a pst, and neither LoadUsageEntry nor CreateNewUsageEntry had been called for this session, then OEMCrypto_ERROR_UNKNOWN_FAILURE is returned.

Nonce verification behaves as before: online licenses always check the nonce, and offline licenses only check the nonce for new usage table entries.

LoadKeys will set the ForbidReport flag, which prevents an application from generating a usage report without updating the table first.

```
OEMCryptoResult OEMCrypto_UpdateUsageEntry(OEMCrypto_SESSIONsession,
                      uint8_t* header_buffer,
                      size_t* header_buffer_length,
                      uint8_t* entry_buffer,
                      size_t* entry_buffer_length);
```

Updates the session's usage entry and fills buffers with the encrypted and signed entry and usage table header. OEMCrypto will update all time and status values in the entry, and then increment the entry's generation number. The corresponding generation number in the usage table header is also incremented so that it matches the one in the entry. The master generation number in the usage table header is incremented and is copied to secure persistent storage. OEMCrypto will encrypt and sign the entry into the entry_buffer, and it will encrypt and sign the usage table header into the header_buffer.

Some actions, such as the first decrypt and deactivating an entry, will also increment the entry's generation number as well as changing the entry's status and time fields. As in OEMCrypto v12, the first decryption will change the status from Inactive to Active, and it will set the time stamp "first decrypt".

If the usage entry has the flag ForbidReport set, then the flag is cleared. It is the responsibility of the CDM layer to call this function and save the usage table before the next call to ReportUsage and before the CDM is terminated. Failure to do so will result in generation number skew, which will invalidate all of the usage table.
Returns:
OEMCrypto_ERROR_SHORT_BUFFER - if header_buffer_length or entry_buffer_length is too small.
OEMCrypto_ERROR_UNKNOWN_FAILURE
OEMCrypto_SUCCESS

```
OEMCryptoResult OEMCrypto_DeactivateUsageEntry(OEMCrypto_SESSIONsession,
                                               const uint8_t* pst,
                                               size_t pst_length);
```

This deactivates the usage entry associated with the current session. This means that the state of the usage entry is changed to InactiveUsed if it was Active, or InactiveUnused if it was Unused. This also increments the entry's generation number, and the header's master generation number. The entry's flag ForbidReport will be set. This flag prevents an application from generating a report of a deactivated license without first saving the entry. Notice that in v12 of this API, there was no distinction between InactiveUsed and InactiveUnused, but there is now.

```
OEMCryptoResult OEMCrypto_ReportUsage(OEMCrypto_SESSIONsession,
                                      const uint8_t* pst, size_t pst_length,
                                      uint8_t *buffer,
                                      size_t *buffer_length);
```

This function generates and signs a usage report as before. If the entry's flag ForbidReport is set, the report is not generated and an error is returned. SImilarly, if any key has been used in a DecryptCENC or any generic crypto functions since the last call to UpdateUsageEntry, then the report is not generated and an error is returned.
This also returns:
OEMCrypto_ERROR_ENTRY_NEEDS_UPDATE - if the ForbidReport flag is set, or decrypt times need to be updated.

```
OEMCryptoResult OEMCrypto_CloseSession(OEMCrypto_SESSIONsession);
```

If the session is associated with a usage table entry, then this frees all memory associated with that entry. It is the CDM layer's responsibility to call UpdateUsageEntry before closing the session.

The following two function can be used to clean up usage tables. This would be used by the CDM layer when an entry is no longer needed.

```
OEMCryptoResult  OEMCrypto_ShrinkUsageTableHeader(u int32_t new_table_size,
                                      uint8_t* header_buffer,
                                      size_t* header_buffer_length);
```

This shrinks the usage table and the header. It is an error if any open session is associated with an entry that will be erased. If new_table_size is larger than the current size, then the header is not changed and the error is returned. If the header has not been previously loaded, then an error is returned. OEMCrypto will increment the master generation number in the header and store the new

value in secure persistent storage. Then, OEMCrypto will encrypt and sign the header into the provided buffer. The generation numbers of all remaining entries will remain unchanged. The next time OEMCrypto_CreateNewUsageEntry is called, the new entry will have an index of new_table_size.
It is an error to shrink the usage table while any open sessions are associated with an entry whose index is greater than or equal to new_table_size. In this case, the new error code OEMCrypto_ERROR_ENTRY_IN_USE is returned.

```
OEMCryptoResult  OEMCrypto_MoveEntry( OEMCrypto_SESSION session,
                                      uint32_t  new_index);
```

Moves the entry associated with the current session from one location in the usage table header to another. This does not modify any data in the entry, except the index and the generation number. The generation number in the header for new_index will be increased to the master generation number, and then the master generation number is incremented. If there was an existing entry at the new location, it will be overwritten. It is an error to call this when the entry that was at new_index is associated with a currently open session. In this case, the new error code OEMCrypto_ERROR_ENTRY_IN_USE is returned. It is the CDM layer's responsibility to call UpdateUsageEntry after moving an entry.

Usage Table Backwards Compatibility Update

For a device in the field which already supports Usage Tables before v13, there may be existing entries for licenses that need to be kept. The CDM layer will be aware of this because there will be offline license files in the file system, and there will be no Usage Table Header file. The following functions will be used after a system upgrade the next time the CDM is run.

```
OEMCryptoResult  OEMCrypto_CopyOldUsageEntry( OEMCrypto_SESSION session,
                                              const uint8_t*pst,
                                              size_t pst_length)
```

This function copies an entry from the old table to the new table. The new entry will already have been loaded by CreateNewUsageEntry. This may return OEMCrypto_ERROR_NOT_IMPLEMENTED if the device did not support pre-v13 usage tables.

```
OEMCryptoResult  OEMCrypto_DeleteOldUsageTable();
```

This function will delete the old usage table, if possible, freeing any nonvolatile secure memory. This may return OEMCrypto_ERROR_NOT_IMPLEMENTED if the device did not support pre-v13 usage tables.

```
OEMCryptoResult OEMCrypto_CreateOldUsageEntry(uint64_ttime_since_license_received,
                                              uint64_t time_since_first_decrypt,
                                              uint64_t time_since_last_decrypt,
                                              OEMCrypto_Usage_Entry_Status status,
                                              uint8_t *server_mac_key,
                                              uint8_t *client_mac_key,
                                              const uint8_t* pst,
                                              size_t pst_length);
```

This forces the creation of an entry in the old usage table in order to test OEMCrypto_CopyOldUsageTable. This may return OEMCrypto_ERROR_NOT_IMPLEMENTED if the device did not support pre-v13 usage tables. OEMCrypto will create a new entry, set the status and

compute the times at license receive, first decrypt and last decrypt. The mac keys are NOT encrypted, and will only correspond to a test license.

Usage Table Errors

The error condition of a full usage table shall generate an actionable error message. If CreateNewUsageEntry fails because there are too many usage table entries, it will return OEMCrypto_ERROR_INSUFFICIENT_RESOURCES. This error will be passed up to the application by the CDM layer. The usage table will hold at least 400 entries.

Usage Report Update

The enumeration, OEMCrypto_Usage_Entry_Status, which is used in the usage report to indicate an entries status, shall have a new value: kInactiveUnused. Currently, there are already values for active, unused, and inactive. The previous value kInactive will be renamed to kInactiveUsed. This allows us to tell if a deactivated license was never used or not.

Also, the structure for the Usage Report used the C++ attribute packed, which is not portable. This will be replaced with a buffer of the appropriate size, so that OEMCrypto may pack the data by hand.

Android Hardware Abstraction Layer

The Android Hardware Abstraction Layer is a new feature in Android O. It will be made at the DRM plugin level. There will be no changes to OEMCrypto.

Unit Tests

Source and destination buffers may point to same buffer, where applicable: Generic Encrypt and Decrypt, DecryptCENC.

Shared License

A service provider can use shared licenses as a way to group permissions. There is a single master license for the device that will have an entry in the usage table. This license will be loaded in the normal way, with nonce checks following the normal rules. Then, the session will have LoadKeys called with a second, shared, license. This behaviour has been previously described in OEMCrypto v11. The new functionality is that there is a bit in the key control block, **Shared_License**. The second license will have this bit set.

If the bit **Shared_License** is set, then it is an error for LoadKeys to be called on this license without first loading a master license. In this case, LoadKeys returns OEMCrypto_ERROR_MISSING_MASTER. The shared license will not have the replay control flag set, but if the session is already associated with an an entry in the usage table, and that entry is marked inactive, then it is an error to load the shared license, and LoadKeys will return the error OEMCrypto_ERROR_MISSING_MASTER.

Analog Output Restriction

The key control block will have a new flag to indicate that a key cannot be used for analog output, called Disable_Analog_Output. This applies to both video and audio output. A key with this bit set cannot be expected to decrypt audio output. For video output, this flag will only be set for keys that already require a secure video path. If a device has analog video output ports, these ports must be disabled. If the device cannot disable these ports, then any call to OEMCrypto_DecryptCENC for this key will return OEMCrypto_ERROR_ANALOG_OUTPUT.

Key Control Block Changes

The following new flags are added to the key control block:

bit 21: Disable_Analog_Output. Data decrypted with this key may not be sent to analog output. This applies to video and audio data.

bit 22: SRM_Version_Required. The SRM Version number is passed into load keys as a separate pointer. It will be checked once at load keys. Like pst and whitelist, the version will be passed in as a separate pointer to load keys.

bit 23: Shared_License. The key can only be loaded if a previous license, the master license, has already been loaded and verified in this session.