WIDEVINE

Widevine Core Message Serialization

April 6th, 2019 (API 16.2)

Document Status: Shared externally with partners who have signed the Widevine MDA. This document is being distributed as part of the OEMCrypto v16 design.

Introduction

This document explains the design of the Core Message Serialization feature for OEMCrypto v16. This design requires changes to both the OEMCrypto and CDM layers on the device, as well as changes to the provisioning and license servers and Widevine server SDK. We assume the reader is aware of the whole Widevine system and the purposes of various messages, and is familiar with at least part of Widevine system. In particular, a key fact is that OEMCrypto is delivered by SOC or OEM partners and runs in a Trusted Execution Environment (TEE) on the device, while the CDM layer is written by Widevine and frequently runs in a non-trusted Rich Execution Environment (REE). OEMCrypto runs on Android and other Consumer Electronic devices. Some devices, such as the desktop browser Chrome, support Widevine without using OEMCrypto. This document only applies to devices that use OEMCrypto.

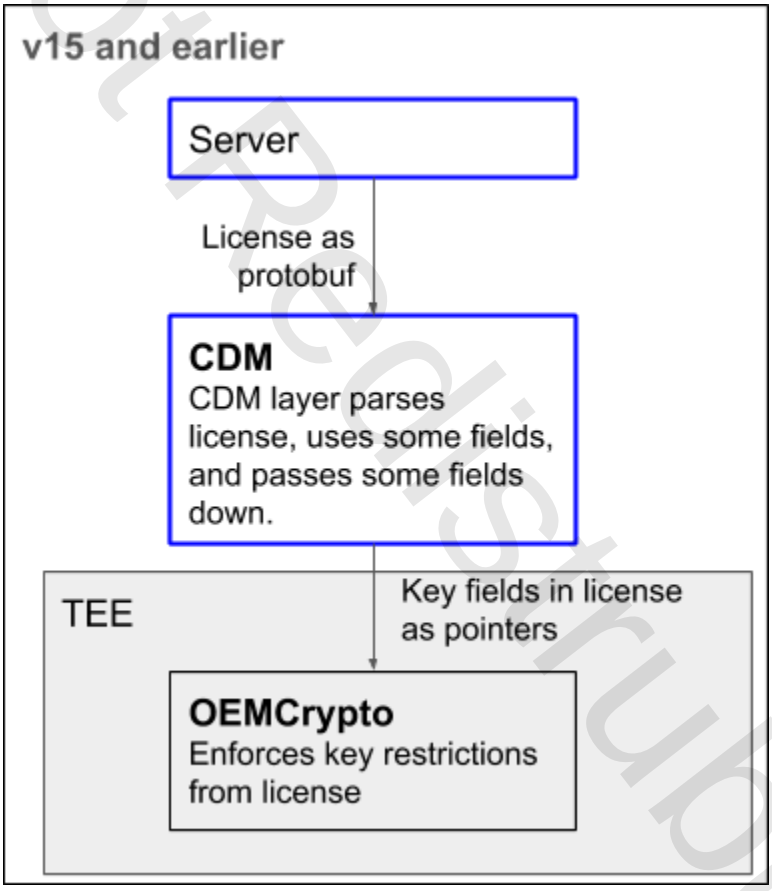
For Widevine Modular DRM, there are six message types between a server and a client device: license request and response, provisioning request and response, and renewal request and response. Some of the data in these messages should be protected or verified by the hardware protected processor on the device. Some of the data in these messages do not need to be protected or verified, and attempting to protect this data adds unnecessary complexity.

In this context, data is "**protected**" by OEMCrypto if no actors between the server and OEMCrypto can modify or access the data. Modification is prevented by having the server sign the message. Access is prevented by encrypting the data. For example, key data should be protected from modification and access limited; key control blocks should be protected from modification.

In this context, "**verified**" by OEMCrypto means that OEMCrypto checks that the data is correct before signing the message. For example, the license request message has a nonce field. This nonce is generated by OEMCrypto, so OEMCrypto can verify that the nonce is correct.

Design Change (v15 to v16)

In OEMCrypto v15 and earlier, messages from the server were parsed by the CDM layer above OEMCrypto and gave OEMCrypto a collection of pointers to protected data within the message. This design allowed complicated parsing code to be placed in the CDM layer. However, the pointers themselves were not signed by the server.

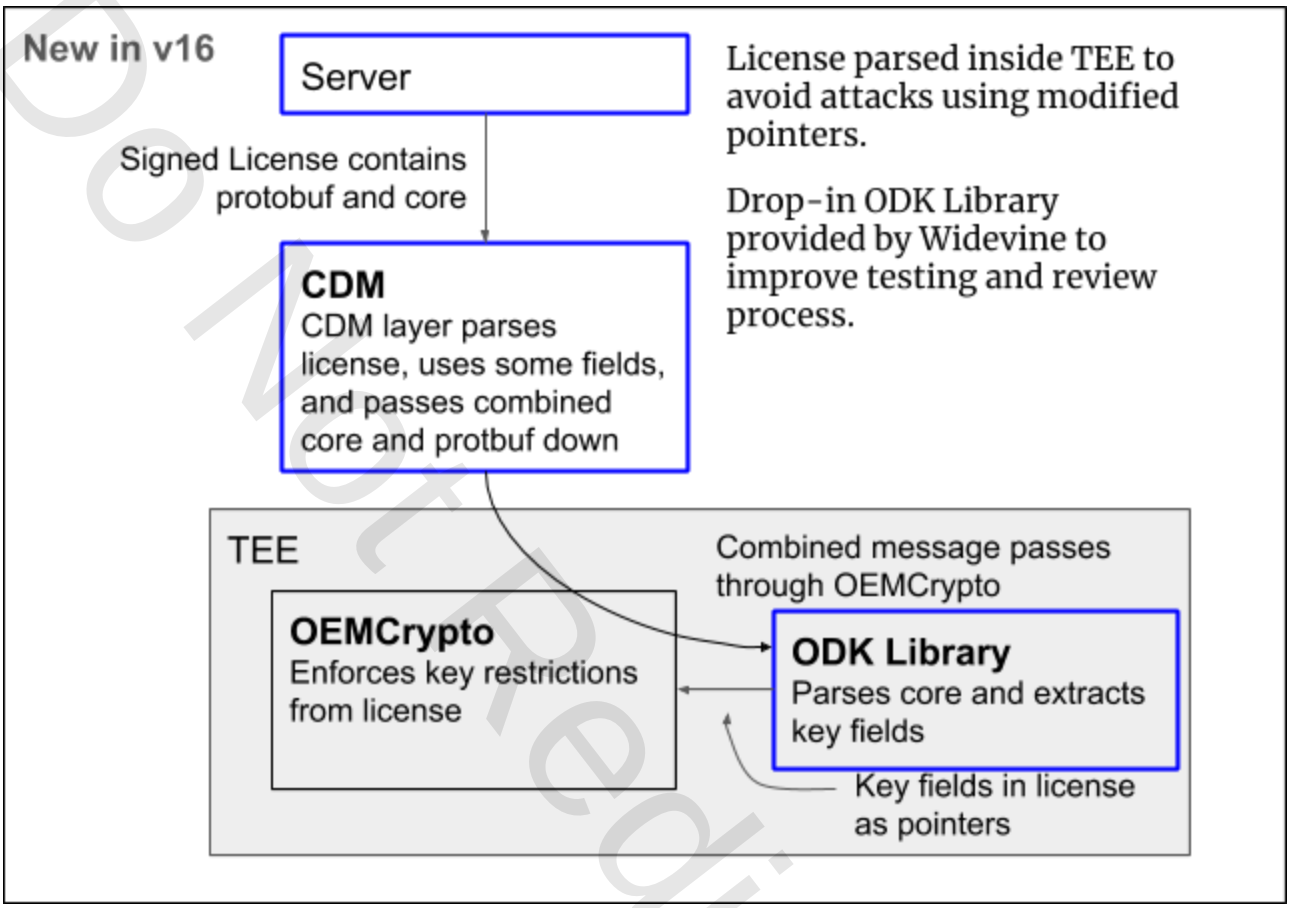


Similarly, the three request messages that are sent to the server and signed by OEMCrypto have data that was not verified by OEMCrypto before signing.

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

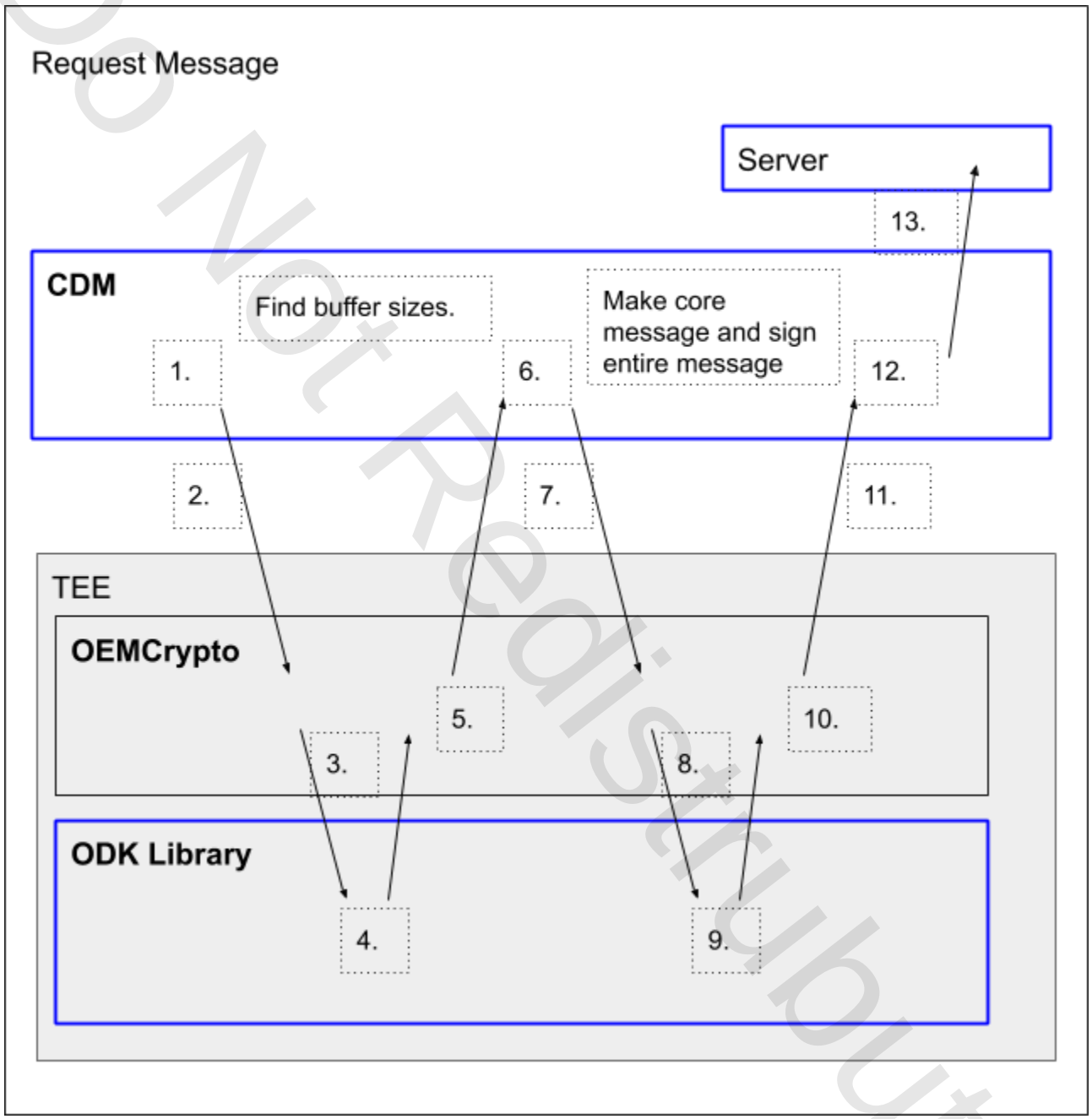
For request messages, OEMCrypto will generate the core message. Then it will sign the combined message.

Data Flow

The diagram below shows the data flow for signing a request message. Since the flow is the same for all three requests, we use OEMCrypto_PrepAndSignRequest to represent OEMCrypto_PrepAndSignLicenseRequest, OEMCrypto_PrepAndSignProvisioningRequest, or OEMCrypto_PrepAndSignRenewalRequest. Similarly we'll use ODK_PrepareCoreMessage to represent each of the three prepare functions.

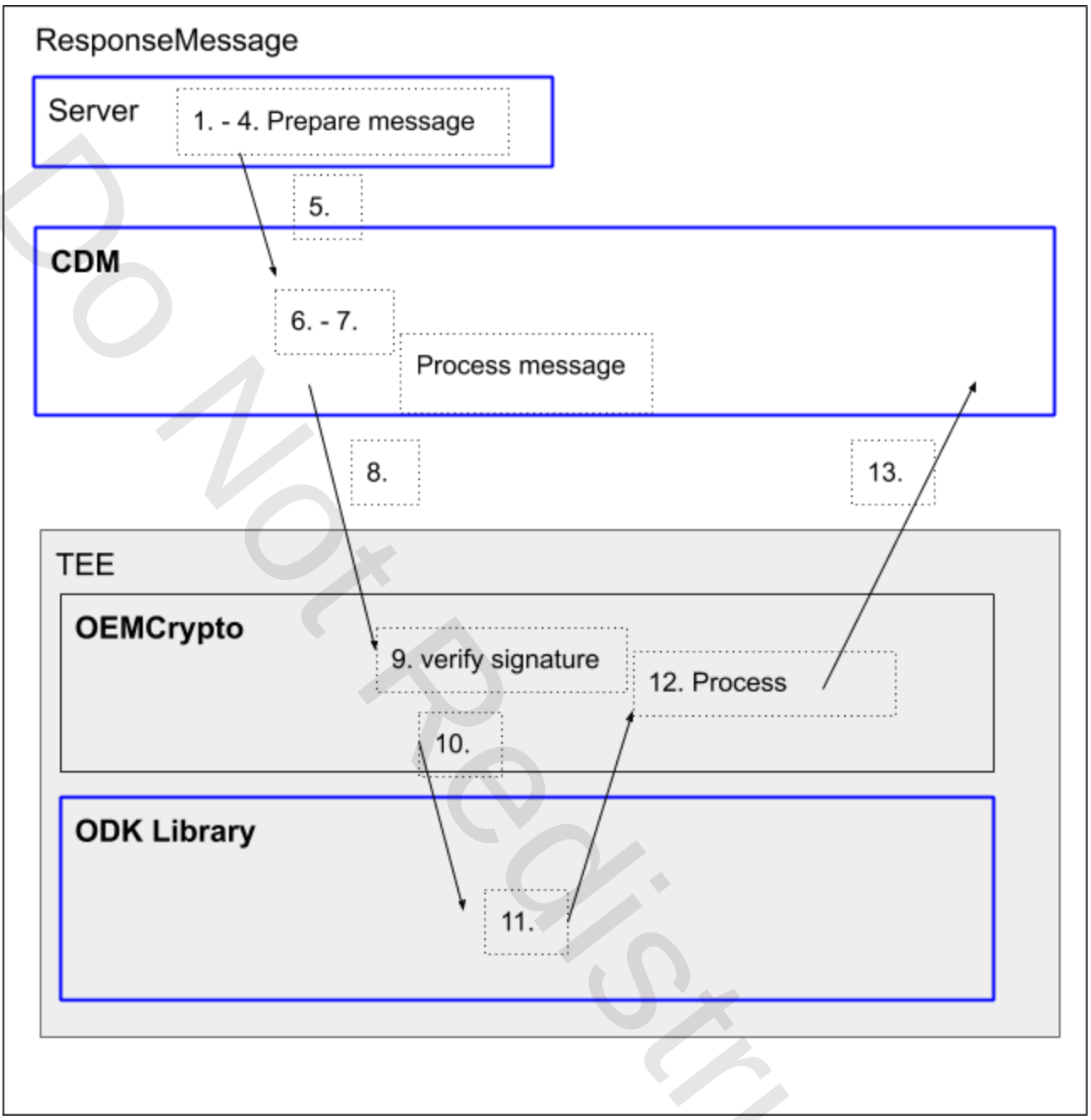
1. CDM prepares the protobuf message with request information.
2. CDM calls OEMCrypto_PrepAndSignRequest with zero length for signature and core message.
3. OEMCrypto calls ODK_PrepareRequest with zero length for core message.
4. ODK sets core message length.
5. OEMCrypto sets signature length and returns both lengths to CDM.
6. CDM allocates buffers for the signature and message.
7. CDM calls OEMCrypto_PrepAndSignRequest again.
8. OEMCrypto calls ODK_PrepareRequest.

9. ODK modifies the header of the message by placing the core message at the start.
10. OEMCrypto signs the entire message.
11. OEMCrypto passes the signature and the modified message back to the CDM layer.
12. CDM separates the core message from the protobuf message and builds a signed message.
13. CDM sends the signed message to the license server.



The diagram below shows the data flow for processing a message from the server. Since the flow is the same for all three message types, we use OEMCrypto_LoadMessage to represent OEMCrypto_LoadLicense and OEMCrypto_ReloadLicense, OEMCrypto_LoadRenewal, or OEMCrypto_LoadProvisioning. Similarly we'll use ODK_ParseMessage to represent each of the three parse functions. Both OEMCrypto_LoadLicense and OEMCrypto_ReloadLicense call ODK_ParseLicense because they handle the same message -- LoadLicense for the first load and ReloadLicense for all subsequent loads.

1. Server extracts the signature, protobuf message, and core message from the signed message and verifies the signature.
2. Server validates the message and prepares the protobuf response message.
3. Server creates the core message for response.
4. Server computes signature of the concatenation of the core message with the protobuf message. Server creates the signed message.
5. Server sends the signed message to the device.
6. CDM extracts core message, signature, and protobuf message from signed message.
7. CDM concatenates the core message with the protobuf message.
8. CDM calls OEMCrypto_LoadLicense with combined message and signature.
9. OEMCrypto verifies the signature of the message.
10. OEMCrypto passes message and data structure to ODK_ParseMessage.
11. ODK fills in data structure based on fields from message.
12. OEMCrypto processes data structure. This is data flow is the same as v15 functionality.
13. OEMCrypto returns status to CDM.



CDM and Server Changes

The SignedMessage protobuf that is sent between the device and the server will have a new field:

```
optional bytes oemcrypto_core_message = 9;
```

If core_message is present in a request, the server will verify the signature of the concatenation of `(core_message | protobuf)` instead of just the serialized protobuf message. Second, the server will extract any necessary fields from the core message that it needs to process the message.

When generating a response to a message that had a core_message, the server will generate a core message appropriate for the response. Each of the six message types has a different format and is tagged with an API

version. As with the request, the server will compute the signature of the concatenation of `(core_message | protobuf)` instead of just the serialized protobuf message.

CDM Changes

When the CDM layer generates a request message, it follows the same data flow as is currently used. However, in order to sign the message it will call the `OEMCrypto_PrepAndSign*` function as described below. The OEMCrypto signature functions described below modify the message to include the core message and generate the signature string. The CDM layer will concatenate an empty buffer for the core message together with the serialized protobuf message. The CDM layer will also allocate a buffer for the signature. Both buffers are passed into the OEMCrypto signature function. After OEMCrypto has updated the core message and signed the buffer, the CDM layer will extract the core message and will put it in the signed message's core_message field. The second string is the signature, which will be placed in the signed messages's signature field. The only change from existing functionality is that there is a separate signature function for each message type, and there is a new string of bytes passed from OEMCrypto to the CDM to layer.

When the CDM layer receives a signed message from the server, it follows a similar data flow as is currently used. The exception is that the CDM layer does not extract pointers to key fields. Instead, it concatenates `(core_message | msg)` and passes that into the `OEMCrypto_Load*Message` function described below.

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 Trusted Application running in the TEE. 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. It shall check for memory permissions, memory overlaps, integer overflows etc.

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.

`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 document “License Duration and Renewal”.. Other ODK functions are as follows:

```
OEMCryptoResult ODK_InitializeSessionValues(
                          ODK_TimerLimits *timer_limits,
                          ODK_ClockValues *clock_values,
                          ODK_NonceValues *nonce_values,
                          uint32_t api_major_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_major_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.

OEMCrypto Changes

Signature of Request Messages

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. This behaviour is different from that described above in the overview, and will be removed in future versions of the spec.

OEMCrypto shall compute a hash 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.

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.

The behaviour of this function is slightly different depending on if the license has an API level of 16 or earlier. The API level of the license is found in the sessions ODK_NonceValues field, nonce_values.

If nonce_values.api_major_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_major_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/out] 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 request 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.

Process Response Messages

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 license 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 the replay control and the nonce value of each key 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_major_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. 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.
4. 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.
5. 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.
6. OEMCrypto shall perform the same verification of the key control block as it did in the v15 spec. This includes verifying the SRM version if required.

Usage Entry Verification

The function ODK_ParseLicense takes several parameters that may need more explanation.

The parameter usage_entry_present shall be set to true if a usage entry was created or loaded for this session. This parameter is used by ODK_ParseLicense 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 by OEMCrypto 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;
  bool nonce_required;
  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. In particular, if the this is not an initial license load, the values will be set to match those in the license.
- [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_major_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 signature of the message, as before.
2. 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,
                                     uint64_t* timer_value);
```

This function updates the clock_values as needed if the renewal is accepted. The field nonce_values.api_major_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".
- [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 entire message using the session's renewal mac key for the server. The entire message is the buffer starting at `message` with length `message_length`. If the signature does not match, OEMCrypto returns OEMCrypto_ERROR_SIGNATURE_FAILURE.

OEMCrypto shall verify that nonce_values.api_major_version is 16. If not, return the error OEMCrypto_ERROR_INVALID_CONTEXT. Legacy licenses will use the function OEMCrypto_RefreshKeys instead of OEMCrypto_LoadRenewal.

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 ODK_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 disabled. 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 functions OEMCrypto_RewrapDeviceRSAKey or OEMCrypto_RewrapDeviceRSAKey30 depending on if the device has a keybox (Provisioning 2.0) or has an OEM Certificate (Provisioning 3.0).

```
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;
  OEMCrypto_Substring encrypted_message_key; // Used for Prov 3.0
} ODK_ParsedProvisioning;
```

ODK Core Message Formats

The ODK Core Messages are parsed and generated only by the server and the ODK library. The details of these messages are not needed to implement OEMCrypto.

Each message has a fixed format with no optional fields. Integers are stored in network byte order and are either 32 or 64 bits. Substrings are stored as a pair of 32 bit integers representing offset and length. The offset is relative to the protobuf message. Some substrings are optional -- an offset of 0 and a length of 0 indicate the substring is not present. The ODK library will verify that each substring is contained in the message.

The core message format for all the messages is the same for the first four fields. The first four fields are required and are always in this order.

Type	**Size**	**Name**	**Description**
uint32	4	message_type	Enumeration for the type of message
uint32	4	message_size	The total size of the core message -- including the message_type, message_size, and all following fields. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
		...	The rest of the core message are described below.

The message_type can be one of the following:

```
enum {
  ODK_License_Request_Type = 1;
  ODK_License_Response_Type = 2;
  ODK_Renewal_Request_Type = 3;
  ODK_Renewal_Response_Type = 4;
  ODK_Provisioning_Request_Type = 5;
  ODK_Provisioning_Response_Type = 6;
};
```

OEMCrypto should not accept a message with an API version greater than its own. For devices which support offline license that may be reloaded after a system update, OEMCrypto should accept older API versions than its own for license response messages.

License Request

Type	Size	Name	Description
uint32	4	message_type	Always ODK_License_Request_Type for license request. (always 1) Unsigned integer, network byte order.
uint32	4	message_size	Always 20 for v16 license request. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
uint32	4	nonce	The nonce. Unsigned integer, network byte order.
uint32	4	session_id	The OEMCrypto session id. Unsigned integer, network byte order.

Renewal Request

Type	Size	Name	Description
uint32	4	message_type	Always ODK_Renewal_Request_Type for renewal request. (always 3) Unsigned integer, network byte order.

uint32	4	message_size	Always 28 for v16 renewal request. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
uint32	4	license_nonce	The nonce from the original license. Unsigned integer, network byte order.
uint32	4	session_id	The OEMCrypto session id. Unsigned integer, network byte order.
uint64	8	playback_time	The time since playback began -- i.e. the time on the playback clock. Unsigned integer, network byte order.

Provisioning Request

Type	**Size**	**Name**	**Description**
uint32	4	message_type	Always ODK_Provisioning_Request_Type for provisioning request. (always 5) Unsigned integer, network byte order.
uint32	4	message_size	The total size of the core message -- including all fields. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
uint32	4	nonce	The nonce. Unsigned integer, network byte order.
uint32	4	session_id	The OEMCrypto session id. Unsigned integer, network byte order.
uint32	4	device_id_length	Length of the device ID. Unsigned integer, network byte order.
bytes	64	device_id	Device unique id string. Padded by 0s.

License Response

Note: Fields are defined and their uses explained in the description of the ODK_ParsedLicense structure and the ODK_ParseLicense function.

Type	Size	Name	Description
uint32	4	message_type	Always ODK_License_Response_Type for license response. (always 2) Unsigned integer, network byte order.
uint32	4	message_size	The total size of the core message -- including all fields. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
uint32	4	nonce	An echo of the nonce -- copied from the request. Unsigned integer, network byte order.
uint32	4	session_id	An echo of the OEMCrypto session id -- copied from the request. Unsigned integer, network byte order.
Substring	8	enc_mac_keys_iv	See description of ODK_ParsedLicense for definition of fields.
Substring	8	enc_mac_keys	
Substring	8	pst	
Substring	8	srm_restriction_data	
uint32	4	license_type	Enumeration.
uint32	4	nonce_required	enumeration: NoNonce, SingleUse, AllowPersist.
uint32	4	soft_rental_expiry	Boolean: 0 = false, 1 = true.
uint32	4	soft_playback_expiry	Boolean: 0 = false, 1 = true.
uint64	8	earliest_playback_start	Start of the rental window, in seconds since the license was signed. Unsigned integer, network byte order.
uint64	8	rental_duration	Unsigned integer, network byte order.
uint64	8	total_playback_duration	Unsigned integer, network byte order.
uint64	8	initial_renewal_duration	Unsigned integer, network byte order.

uint32	4	key_array_length	Unsigned integer, network byte order
			The following 5 rows are repeated **num_key** times.
Substring	8	key_id	
Substring	8	key_data_iv	
Substring	8	key_data	Encrypted by session enc key.
Substring	8	key_control_iv	
Substring	8	key_control	Encrypted by content key.
bytes	32	request_hash	SHA256 hash of license request.

Renewal Response

Type	**Size**	**Name**	**Description**
uint32	4	message_type	Always ODK_Renewal_Response_Type for renewal response. (always 4) Unsigned integer, network byte order.
uint32	4	message_size	The total size of the core message -- including all fields. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
uint32	4	license_nonce	An echo of the nonce -- copied from the request. Unsigned integer, network byte order.
uint32	4	session_id	The OEMCrypto session id. Unsigned integer, network byte order.
uint64	8	playback_time	The time since playback began -- i.e. the time on the playback clock when the renewal was requested. Unsigned integer, network byte order.
uint64	8	renewal_duration	New duration for the playback timer. Unsigned integer, network byte order.

Provisioning Response

Type	Size	Name	Description
uint32	4	message_type	Always ODK_Provisioning_Response_Type for provisioning response. (always 6) Unsigned integer, network byte order.
uint32	4	message_size	The total size of the core message -- including all fields. Unsigned integer, network byte order.
uint16	2	api_minor_version	The API minor version of the message. Unsigned integer, network byte order.
uint16	2	api_major_version	The API major version of the message. Unsigned integer, network byte order.
uint32	4	nonce	An echo of the nonce -- copied from the request. Unsigned integer, network byte order.
uint32	4	session_id	The OEMCrypto session id. Unsigned integer, network byte order.
uint32	4	device_id_length	Length of the device ID. Unsigned integer, network byte order.
bytes	64	device_id	Device unique id string. Padded by 0s.
uint32	4	key_type	ECC or RSA.
Substring	8	enc_private_key	
Substring	8	enc_private_key_iv	
Substring	8	encrypted_message_key	

Risk Mitigation

Widevine will fuzz test the ODK library and will provide source for partners to perform their own security review.

Complete ODK API

The full ODK API is gathered below for completeness.

ODK_TimerLimits Structure

```
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;
```

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. The fields in this structure are directly related to the fields in the core license message. The fields are set when OEMCrypto calls the function ODK_ParseLicense or ODK_InitializeV15Values.

Fields

soft_enforce_rental_duration: A boolean controlling the soft or hard enforcement of rental duration.

soft_enforce_playback_duration: A boolean controlling the soft or hard enforcement of playback duration.

earliest_playback_start_seconds: The earliest time that the first playback is allowed. Measured in seconds since the license request was signed. For most use cases, this is zero.

rental_duration_seconds: Window of time for the allowed first playback. Measured in seconds since the **earliest** playback start. If soft_enforce_rental_duration is true, this applies only to the first playback. If soft_enforce_rental_duration is false, then this restricts any playback. A value of zero means no limit.

total_playback_duration_seconds: Window of time for allowed playback. Measured in seconds since the **first** playback start. If soft_enforce_playback_duration is true, this applies only to the start of playback for any session. If soft_enforce_playback_duration is false, then this restricts any playback. A value of zero means no limit.

initial_renewal_duration_seconds: Window of time for allowed playback. Measured in seconds since the **first** playback start. This value is only used to start the renewal timer. After a renewal message is loaded, the timer will be reset. A value of zero means no limit.

Version

This struct changed in API version 16.2.

ODK_ClockValues Structure

```
typedef struct {
  uint64_t time_of_license_signed;
  uint64_t time_of_first_decrypt;
  uint64_t time_of_last_decrypt;
  uint64_t time_of_renewal_request;
  uint64_t time_when_timer_expires;
  uint32_t timer_status;
  enum OEMCrypto_Usage_Entry_Status status;
} ODK_ClockValues;
```

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 document "License Duration and Renewal".

Fields

time_of_license_signed: Time that the license request was signed, based on OEMCrypto's system clock. This value shall be stored and reloaded with usage entry as time_of_license_received.

time_of_first_decrypt: Time of the first decrypt or call select key, based on OEMCrypto's system clock. This is 0 if the license has not been used to decrypt any data. This value shall be stored and reloaded with usage entry.

time_of_last_decrypt: Time of the most recent decrypt call, based on OEMCrypto's system clock. This value shall be stored and reloaded with usage entry.

time_of_renewal_request: Time of the most recent renewal request, based on OEMCrypto's system clock. This is used to verify that a renewal is not stale.

time_when_timer_expires: Time that the current timer expires, based on OEMCrypto's system clock. If the timer is active, this is used by the ODK library to determine if it has expired.

timer_status: Used internally by the ODK library to indicate the current timer status.

status: The license or usage entry status. This value shall be stored and reloaded with usage entry.

Version

This struct changed in API version 16.2.

ODK_NonceValues Structure

```
typedef struct {
  uint16_t api_minor_version;
  uint16_t api_major_version;
  uint32_t nonce;
  uint32_t session_id;
} ODK_NonceValues;
```

Nonce values are used to match a license or provisioning request to a license or provisioning response. They are also used to match a renewal request and response to a license. 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.

Fields

api_major_version: the API version of the license. This is initialized to the API version of the ODK library, but may be lower.

api_minor_version: the minor version of the ODK library. This is used by the server to verify that device is not using an obsolete version of the ODK library.

nonce: a randomly generated number used to prevent replay attacks.

session_id: the session id of the session which signed the license or provisioning request. It is used to prevent replay attacks from one session to another.

Version

This struct changed in API version 16.2.

ODK_InitializeSessionValues

```
OEMCryptoResult ODK_InitializeSessionValues(
                            ODK_TimerLimits *timer_limits,
                            ODK_ClockValues *clock_values,
```

```
                                ODK_NonceValues *nonce_values,
                                uint32_t api_major_version,
                                uint32_t session_id);
```

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

Parameters

[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_major_version: the API version of OEMCrypto.
[in] session_id: the session id of the newly created session.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_INVALID_CONTEXT

Version

This method is new in version 16 of the API.

ODK_SetNonceValues

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

This function sets the nonce value in the session's nonce structure. It shall be called from OEMCrypto_GenerateNonce.

Parameters

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

Returns

true on success

Version

This method is new in version 16 of the API.

ODK_InitializeClockValues

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

This function initializes the clock values in the session clock_values structure. It shall be called from OEMCrypto_PrepAndSignLicenseRequest.

Parameters

[in/out] clock_values: the session's clock data.
[in] system_time_seconds: the current time on OEMCrypto's monotonic clock.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_INVALID_CONTEXT

Version

This method is new in version 16 of the API.

ODK_ReloadClockValues

```
OEMCryptoResult 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);
```

This function sets the values in the clock_values structure. It shall be called from OEMCrypto_LoadUsageEntry. When a usage entry from a v15 or earlier license is loaded, the value time_of_license_loaded shall be used in place of time_of_license_signed.

Parameters

[in/out] clock_values: the session's clock data.
[in] time_of_license_signed: the value time_license_received from the loaded usage entry.
[in] time_of_first_decrypt: the value time_of_first_decrypt from the loaded usage entry.
[in] time_of_last_decrypt: the value time_of_last_decrypt from the loaded usage entry.
[in] status: the value status from the loaded usage entry.
[in] system_time_seconds: the current time on OEMCrypto's monotonic clock.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_INVALID_CONTEXT

Version

This method is new in version 16 of the API.

ODK_AttemptFirstPlayback

```
OEMCryptoResult 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. It uses the values in clock_values to determine if this is the first playback for the license or the first playback for just this session.

This shall be called from the first call in a session to any of OEMCrypto_DecryptCENC or any of the OEMCrypto_Generic* functions.

If OEMCrypto uses a hardware timer, and this function returns ODK_SET_TIMER, then the timer should be set to the value pointed to by timer_value.

Parameters

[in] system_time_seconds: the current time on OEMCrypto's monotonic 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. This must be non-null if OEMCrypto uses a hardware timer.

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.

Version

This method is new in version 16 of the API.

ODK_UpdateLastPlaybackTime

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

Vendors that do not implement their own timer should call ODK_UpdateLastPlaybackTime regularly during playback. This updates the clock values, and determines if playback may continue based on the given system time. This shall be called from any of OEMCrypto_DecryptCENC or any of the OEMCrypto_Generic* functions.

All Vendors (i.e. those that do or do not implement their own timer) shall call ODK_UpdateLastPlaybackTime from the function OEMCrypto_UpdateUsageEntry before updating the usage entry so that the clock values are accurate.

Parameters

[in] system_time_seconds: the current time on OEMCrypto's monotonic clock, in seconds.
[in] timer_limits: timer limits specified in the license.
[in/out] clock_values: the sessions clock values.

Returns

OEMCrypto_SUCCESS: Success. Playback is allowed.
ODK_TIMER_EXPIRED: Set timer as disabled. Playback is **not** allowed.

Version

This method is new in version 16 of the API.

ODK_DeactivateUsageEntry

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

This function modifies the session's clock values to indicate that the license has been deactivated. It shall be called from OEMCrypto_DeactivateUsageEntry

Parameters

[in/out] clock_values: the sessions clock values.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_INVALID_CONTEXT

Version

This method is new in version 16 of the API.

ODK_PrepareCoreLicenseRequest

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

Modifies the message to include a core license request at the beginning of the message buffer. The values in nonce_values are used to populate the message.

This shall be called by OEMCrypto from OEMCrypto_PrepAndSignLicenseRequest.

NOTE: if the message pointer is null and/or input core_message_size is zero, this function returns OEMCrypto_ERROR_SHORT_BUFFER and sets output core_message_size to the size needed.

Parameters

[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.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_SHORT_BUFFER: core_message_size is too small
OEMCrypto_ERROR_INVALID_CONTEXT

Version

This method is new in version 16 of the API.

ODK_PrepareCoreRenewalRequest

```
OEMCryptoResult ODK_PrepareCoreRenewalRequest(
                            uint8_t *message,
                            size_t message_length,
```

```
                            size_t *core_message_size,
                            ODK_NonceValues *nonce_values,
                            ODK_ClockValues *clock_values,
                            uint64_t system_time_seconds);
```

Modifies the message to include a core renewal request at the beginning of the message buffer. The values in nonce_values, clock_values and system_time_seconds are used to populate the message. The nonce_values should match those from the license.

This shall be called by OEMCrypto from OEMCrypto_PrepAndSignRenewalRequest.

If status in clock_values indicates that a license has not been loaded, then this is a license release. The ODK library will change the value of nonce_values.api_major_version to 15. This will make OEMCrypto_PrepAndSignRenewalRequest sign just the message body, as it does for all legacy licenses.

NOTE: if the message pointer is null and/or input core_message_size is zero, this function returns OEMCrypto_ERROR_SHORT_BUFFER and sets output core_message_size to the size needed.

Parameters

[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/out] nonce_values: pointer to the session's nonce data.
[in/out] clock_values: the session's clock values.
[in] system_time_seconds: the current time on OEMCrypto's clock, in seconds.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_SHORT_BUFFER: core_message_size is too small
OEMCrypto_ERROR_INVALID_CONTEXT

Version

This method is new in version 16 of the API.

ODK_PrepareCoreProvisioningRequest

```
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);
```

Modifies the message to include a core provisioning request at the beginning of the message buffer. The values in nonce_values are used to populate the message.

This shall be called by OEMCrypto from OEMCrypto_PrepAndSignProvisioningRequest.

The buffer device_id shall be the same string returned by OEMCrypto_GetDeviceID. The device ID shall be unique to the device, and stable across reboots and factory resets for an L1 device.

NOTE: if the message pointer is null and/or input core_message_size is zero, this function returns OEMCrypto_ERROR_SHORT_BUFFER and sets output core_message_size to the size needed.

Parameters

[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.

Returns
OEMCrypto_SUCCESS
OEMCrypto_ERROR_SHORT_BUFFER: core_message_size is too small
OEMCrypto_ERROR_INVALID_CONTEXT

Version
This method is new in version 16 of the API.

ODK_InitializeV15Values

```
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_major_version will be set to 15. It shall be called from OEMCrypto_LoadKeys when loading a legacy license.

Parameters

[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".

Returns
OEMCrypto_SUCCESS
OEMCrypto_ERROR_INVALID_CONTEXT

Version
This method is new in version 16 of the API.

ODK_RefreshV15Values

```
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 a v15 renewal is accepted. The field nonce_values.api_major_version is verified to be 15.

This is called from OEMCrypto_RefreshKeys for a valid license renewal. OEMCrypto shall pass in the current system time, and the key duration from the first object in the OEMCrypto_KeyRefreshObject.

Parameters

[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. This must be non-null if OEMCrypto uses a hardware timer.

Returns

OEMCrypto_SUCCESS
OEMCrypto_ERROR_UNKNOWN_FAILURE
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.

Version

This method is new in version 16 of the API.

ODK_ParseLicense

```
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);
```

The function ODK_ParseLicense will parse the message and verify fields in the message.
If the message does not parse correctly, ODK_VerifyAndParseLicense will return

ODK_ERROR_CORE_MESSAGE that OEMCrypto should return to the CDM layer above.

If the API in the message is not 16, then ODK_UNSUPPORTED_API is returned.

If initial_license_load is true, and nonce_required in the license is true, then the ODK library shall verify that nonce_values->nonce and nonce_values->session_id are the same as those in the message. If verification fails, then it shall return OEMCrypto_ERROR_INVALID_NONCE.

If initial_license_load is false, and nonce_required is true, then ODK_ParseLicense will set the values in nonce_values from those in the message.

The function ODK_ParseLicense will verify that each substring points to a location in the message body. The message body is the buffer starting at `message + core_message_length` with size `message_length - core_message_length`.

If initial_license_load is true, then ODK_ParseLicense shall verify that the parameter request_hash matches request_hash in the parsed license. If verification fails, then it shall return ODK_ERROR_CORE_MESSAGE. This was computed by OEMCrypto when the license was requested.

If usage_entry_present is true, then ODK_ParseLicense shall verify that the pst in the license has a nonzero length.

Parameters

[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 nonce values. These will be updated.
[out] parsed_license: the destination for the data.

Returns

OEMCrypto_SUCCESS
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_UNSUPPORTED_API
OEMCrypto_ERROR_INVALID_NONCE

Version

This method is new in version 16 of the API.

ODK_ParsedLicense Structure

```
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;
  bool nonce_required;
  ODK_TimerLimits timer_limits;
  uint32_t key_array_length;
  OEMCrypto_KeyObject key_array[ODK_MAX_NUM_KEYS];
} ODK_ParsedLicense;
```

The parsed license structure contains information from the license message. The function ODK_ParseLicense will fill in the fields of this message. All substrings are contained within the message body.

Fields

enc_mac_keys_iv: IV for decrypting new mac_key. Size is 128 bits.

enc_mac_keys: encrypted mac_keys for generating new mac_keys. Size is 512 bits.

pst: the Provider Session Token.

srm_restriction_data: optional data specifying the minimum SRM version.

license_type: specifies if the license contains content keys or entitlement keys.

nonce_required: indicates if the license requires a nonce.

timer_limits: time limits of the for the license.

key_array_length: number of keys present.

key_array: set of keys to be installed.

Version

This struct changed in API version 16.2.

ODK_ParseRenewal

```
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 ODK_TimerLimits* timer_limits,
                            ODK_ClockValues* clock_values,
                            uint64_t *timer_value);
```

The function ODK_ParseRenewal will parse the message and verify its contents. If the message does not parse correctly, an error of ODK_ERROR_CORE_MESSAGE is returned.

ODK_ParseRenewal shall verify that all fields in nonce_values match those in the license. Otherwise it shall return OEMCrypto_ERROR_INVALID_NONCE.

After parsing the message, this function updates the clock_values based on the timer_limits and the current system time. If playback may not continue, then ODK_TIMER_EXPIRED is returned.

If playback may continue, a return value of ODK_SET_TIMER or ODK_TIMER_EXPIRED is returned. If the return value is ODK_SET_TIMER, then playback may continue until the timer expires. If the return value is ODK_DISABLE_TIMER, then playback time is not limited.

If OEMCrypto uses a hardware timer, and this function returns ODK_SET_TIMER, then OEMCrypto shall set the timer to the value pointed to by timer_value.

Parameters

[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. This must be non-null if OEMCrypto uses a hardware timer.

Returns

ODK_ERROR_CORE_MESSAGE: 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 disabled. Playback is **not** allowed.
ODK_UNSUPPORTED_API
ODK_STALE_RENEWAL: This renewal is not the most recently signed. It is rejected.
OEMCrypto_ERROR_INVALID_NONCE

Version

This method is new in version 16 of the API.

ODK_ParseProvisioning

```
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 function ODK_ParseProvisioning will parse the message and verify the nonce values match those in the license.

If the message does not parse correctly, ODK_ParseProvisioning will return an error that OEMCrypto should return to the CDM layer above.

If the API in the message is larger than 16, then ODK_UNSUPPORTED_API is returned.

ODK_ParseProvisioning shall verify that nonce_values->nonce and nonce_values->session_id are the same as those in the message. Otherwise it shall return OEMCrypto_ERROR_INVALID_NONCE.

The function ODK_ParseProvisioning will verify that each substring points to a location in the message body. The message body is the buffer starting at `message + core_message_length` with size `message_length - core_message_length`.

Parameters

[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.

Returns

OEMCrypto_SUCCESS
ODK_ERROR_CORE_MESSAGE: the message did not parse correctly, or there were other incorrect values. An error should be returned to the CDM layer.
ODK_UNSUPPORTED_API
OEMCrypto_ERROR_INVALID_NONCE

Version

This method is new in version 16 of the API.

ODK_ParsedProvisioning Structure

```
typedef struct {
  OEMCrypto_PrivateKeyType key_type;
  OEMCrypto_Substring enc_private_key;
  OEMCrypto_Substring enc_private_key_iv;
  OEMCrypto_Substring encrypted_message_key; /* Used for Prov 3.0 */
} ODK_ParsedProvisioning;
```

The parsed provisioning structure contains information from the license message. The function ODK_ParseProvisioning will fill in the fields of this message. All substrings are contained within the message body.

Fields

key_type: indicates if this key is an RSA or ECC private key.

enc_private_key: encrypted private key for the DRM certificate.

enc_private_key_iv: IV for decrypting new private key. Size is 128 bits.

encrypted_message_key: used for provisioning 3.0 to derive keys.

Version

This struct changed in API version 16.2.

Errors

Return Codes

This is a list of return codes and their uses.

0	OEMCrypto_SUCCESS	No error.
7	OEMCrypto_ERROR_SHORT_BUFFER	Indicates an output buffer is not long enough to hold its data. Function can be called again with a larger buffer.

29	OEMCrypto_ERROR_INVALID_CONTEXT	Context for signing or verification is not valid, or other sanity check failed.
32	OEMCrypto_ERROR_INVALID_NONCE	Nonce in server response does not match that stored in nonce_values.
1000	ODK_ERROR_CORE_MESSAGE	Core message did not parse correctly.
1001	ODK_SET_TIMER	Playback is allowed for a limited time.
1002	ODK_DISABLE_TIMER	Playback is allowed with no time limit.
1003	ODK_TIMER_EXPIRED	Playback time limit has expired.
1004	ODK_UNSUPPORTED_API	Parsed message has unsupported API
1005	ODK_STALE_RENEWAL	Renewal message was not the most recently signed message.