WIDEVINE

License Duration and Renewal Updates

December 9th, 2019

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

Goals

1. Ease of understanding and use for content providers.
2. Avoid having to keep multiple nonces for multiple in-flight renewal requests.
3. Prevent an attacker from requesting multiple renewals at once, and then feeding them in slowly over time. I will call this an “extension attack”.
4. Prevent an attacker from saving a renewal from one license and using it later on. I will call this a “replay attack”.
5. Prevent an attacker from copying a renewal to another device and using it there. I will call this a “transfer attack”.
6. Allow the renewal server to do minimal work -- just a simple check or accounting to verify license may be renewed.
7. Satisfy all use cases mentioned in Widevine CDM - Use Cases - Time Restrictions

Design change:

There are several fields in a license that are currently in the message passed from the license server to the CDM layer on the device. However, this information has not been passed down to OEMCrypto for enforcement in the TEE. With OEMCrypto v16, the concept of a core message has been introduced. The core message is a collection of fields that have been serialized in a simple manner by the server, and directly parsed by code in the TEE. This document discusses the fields related to license duration and renewal.

Soft or Hard Expiry

A license already has a field soft _enforce_playback_duration, which specifies “Soft Expiry” or “Hard Expiry”. If the license uses “soft expiry”, then an active session will not be stopped during playback. This allows a user to finish watching a video that has already been started, but does not allow a user to restart a video after a license has expired. For OEMCrypto v16, this field will be included in the core license response message.

Clocks and Timers

OEMCrypto will keep two clocks and one timer for each session. A clock starts counting at 0 and increases. A timer starts counting at its initial value, and decreases until 0. A timer can be reset to a new value. Before version 16, OEMCrypto kept one timer for each key.

- **Rental Clock**. The rental clock starts at 0 when the license request is signed. Since we assume that the initial load of a license is within a few seconds of the request, we can think of this clock as starting when the license is granted or loaded. However, if the application does add a delay between the license request and the license load, then it may be important to understand that this clock is started on license request. This clock is used to enforce the rental window -- i.e. the time playback starts, or the first decrypt call. Its value is reported in the usage report as “seconds since license received”. Its start time is stored in the usage entry, and should continue if the session is closed and the license is loaded into a new session.
- **Playback Clock**. The playback clock starts at 0 when the first decrypt call is made. Its value is reported in the usage report as “seconds since first decrypt”. It is not used for enforcement. Its start time is stored in the usage entry, and the playback clock should be restored if the session is closed and the license is loaded into a new session.
- **Playback Timer.** The playback timer is used to enforce playback duration. On the first decrypt call in a session, it is initialized based on the initial_playback_duration_seconds specified in the license, the rental duration, and current time on the two clocks mentioned above. The initial value of the playback timer is described later in this document. When a license renewal is granted, the timer will be restarted with the value renewal_playback_duration_seconds specified in the license. This timer is only used if the license specifies “Hard Expiry”. For hard expiry, when the timer expires, decrypt operations will fail until it is reset by a renewal. For soft expiry the playback timer is not started. The playback timer is not stored in the usage table entry.

OEMCrypto implementers may use the ODK library to implement these clocks and timers on top of a single system clock. This is described below.

OEMCrypto Core License Fields

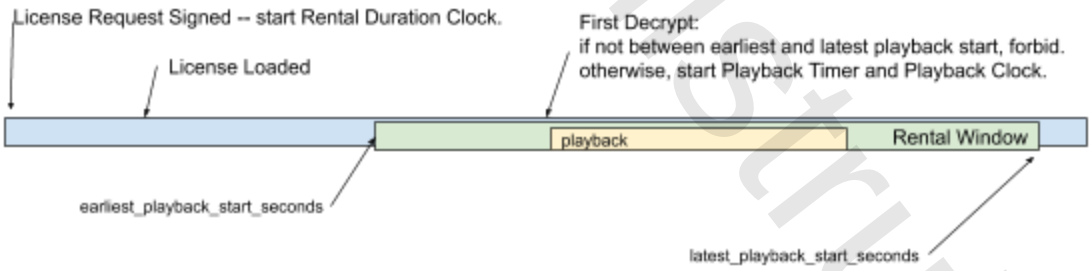
The license, sent by the license server, will have the following fields in the core message. These fields already exist in the Protobuf message which is parsed by the CDM code.

1. **earliest_playback_start_seconds.** Also known as the start of the rental window. This is a time on the rental clock -- i.e. seconds since the license was requested.
2. **latest_playback_start_seconds.** Also known as the end of the rental window. A value of 0 means unlimited. This is a time on the rental clock -- i.e. seconds since the license was requested.
3. **initial_playback_duration_seconds.** This is the initial value of the playback timer until a renewal is verified. A value of 0 means unlimited.
4. **renewal_playback_duration_seconds**. When a license renewal is successfully loaded, the playback timer is restarted at this value.
5. **license_duration_seconds**. Playback limit, starting from earliest_playback_start_seconds.
6. **soft_expiry** - if true, playback is not stopped when the playback timer expires.

License will still contain a pair of mac keys for use with signing renewals. This pair of mac keys is unique to this rental and device.

First Playback for License

The first playback for a license is interpreted by OEMCrypto as the first decrypt call. When the first decrypt call is made, OEMCrypto will verify that the rental duration clock is between the two values “earliest_playback_start_seconds” and “latest_playback_start_seconds”. If not, the decrypt fails and an error is returned.
If the verification passes, then the playback clock is started. For sessions with a usage entry, the first playback time is stored in the usage entry. The playback timer is also started, and initialized as the minimum of time left in the license and the initial playback duration.

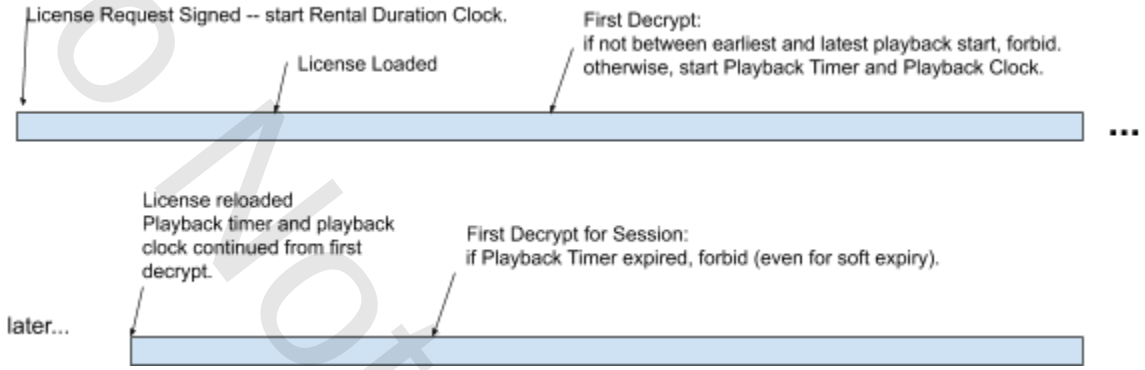


To help OEMCrypto implementers make these checks in a consistent way, Widevine will include helper functions in the ODK library, described below.

First Playback for Session

For an offline license that has been reloaded, there is a distinction between the first playback for a session and the first playback for the license -- i.e. for all sessions. OEMCrypto can tell if the first playback for a license has already occurred because the “first playback time” has been set and saved in the usage table entry. When the license is re-loaded, if the “first playback time” is

set, then the playback timer is started with an initial value set to the initial_playback_duration_seconds minus the time since first playback.
If the license allows renewals, then a valid renewal message will reset the playback timer to the renewal_playback_duration_seconds. This will allow playback to resume even if an earlier attempt failed.



Renewal Request

The core renewal request will have the following information inserted and signed by OEMCrypto:

1. OEMCrypto API level in case we modify the format in the future.
2. The current playback clock time. I.e. seconds since playback started.

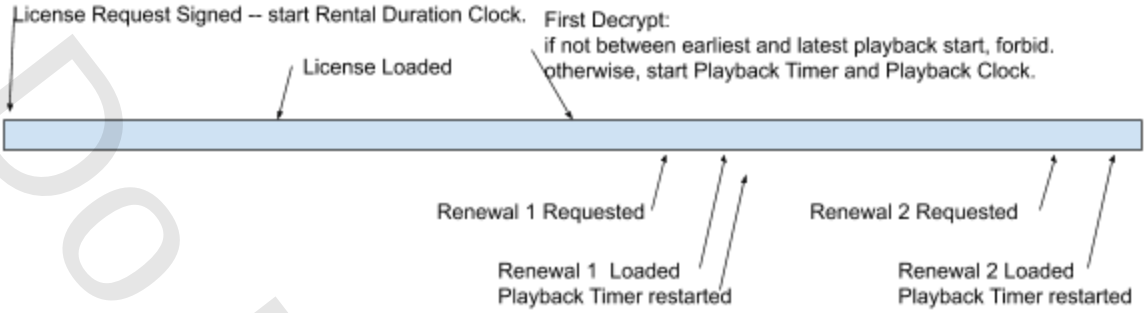
Renewal will be signed by the client mac key from license. Transfer and + attacks will be prevented by using a unique pair of mac keys. See the document “Widevine Core Message Serialization” for the format of the core message.

Renewal Message

The core renewal message will contain:

1. OEMCrypto API level
2. Echo of playback time from the request.

OEMCrypto will verify that the request was made after the most recent timer reset. This prevents an attacker from saving up multiple license requests and replaying them in order to extend a license.
OEMCrypto will verify the signature based on the server mac key. Since the server mac key is unique to this license, the signature prevents replay and transfer attacks.
If the renewal is not rejected, then the playback timer will be restarted at the renewal_playback_duration_seconds.



Backwards Compatibility

The license protobuf has the following existing fields:
license_start_time (in UTC) -- the time of the request.
rental_duration_seconds -
playback_duration_seconds -
license_duration_seconds -
renewal_delay_seconds - time to first renewal.
renewal_retry_interval_seconds - time between renewals.
soft_enforce_playback_duration -- same as soft expiry.

When loading an old v15 license, OEMCrypto only knows the license duration in the key control block. It will use
earliest_playback_start_seconds = 0
latest_playback_start_seconds = license_duration_seconds
initial_playback_duration_seconds = license_duration_seconds
renewal_playback_duration_seconds = license_duration_seconds
license_duration_seconds = license_duration_seconds
These values will be computed in ODK_InitializeV15Values, which shall be called by OEMCrypto_LoadKeys. The clock values will be updated in ODK_RefreshV15Values which shall be called by OEMCrypto_RefreshKeys.

License Server SDK Fields

license_start_time (default = now)-- the time of the request.
rental_duration - Time limit on starting playback.
playback_duration_seconds - Time limit on stopping playback, starting from first playback.

renewal_delay_seconds - time to first renewal. (Also called the license renewal interval)
renewal_retry_interval_seconds - time between renewals. (How long the CDM will wait to retry a renewal request if no response is received).
soft_enforce_playback_duration (default = false) -- If true, playback is not stopped when playback duration expires .
soft_enforce_rental_duration -- (default = true) -- If true, playback is not stopped when rental duration expires. Playback duration may still cause a hard stop. This is not passed to CDM, it is only used to compute license duration.
can_persist = determines if a license can be loaded multiple times.

CDM License (protobuf) Fields

The license protobuf has the following existing fields:
license_start_time - same as sdk.
rental_duration_seconds - same as sdk
playback_duration_seconds - same as sdk.
license_duration_seconds -

- if soft_enforce_rental_duration, license_duration = rental_duration + playback_duration
- if hard enforce rental duration, license_duration = rental_duration

renewal_delay_seconds - same as sdk
renewal_retry_interval_seconds - same as sdk.
soft_enforce_playback_duration -- same as sdk
can_persist = same as sdk.

OEMCrypto Core License Fields

earliest_playback_start = max(0, proto.license_start_time - now)
latest_playback_start = if(proto.rental_duration > 0) earliest_playback + proto.rental_duration; else 0.
initial_playback_duration - proto.playback_duration_seconds
renewal_playback_duration - proto.renewal_retry_interval
license_duration - proto.license_duration
soft_expiry - proto.soft_enforce_playback_duration

Clear Lead

There is a question about when do we start the playback clock: on first decrypt, or first decode. From OEMCrypto's point of view, this would mean do we use only DecryptCENC or do we also use CopyBuffer? What about generic?
For business purposes, it does not matter if the playback clock and timer start includes the clear lead or not. So the decision is based on which is easier to test and which is easier to code.
The decision was made to start the playback clock on the first decrypt call. This include OEMCrypto_DecryptCENC, and the generic crypto calls, but does **not** include OEMCrypto_CopyBuffer.

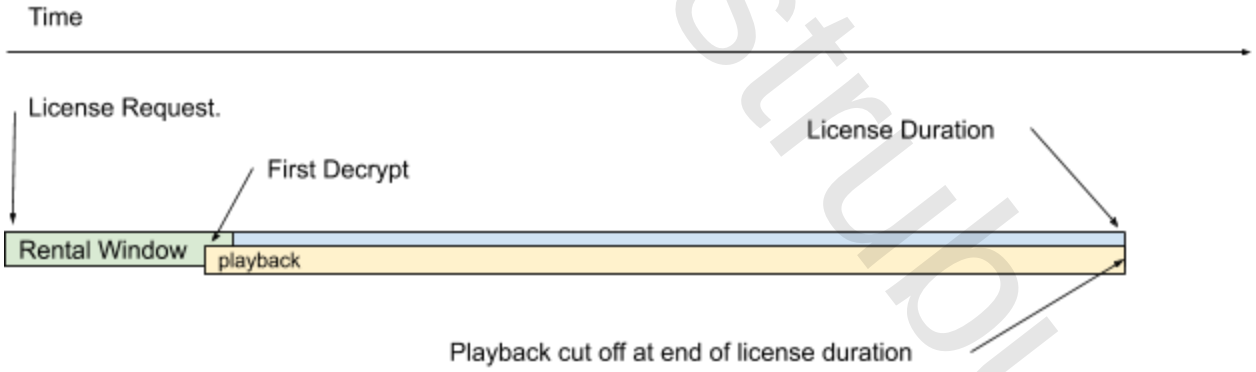
Use Cases

The most common use cases are as follows. Usually, earliest_playback_start = 0, and renewals are not used.

Streaming License

For example, streaming must begin within 15 minutes, and license lasts for 6 hours.

earliest_playback_start = 0
latest_playback_start = 15 minutes
initial_playback_duration = 6 hours
license_duration = 6 hours + 15 minutes
soft_expiry = false



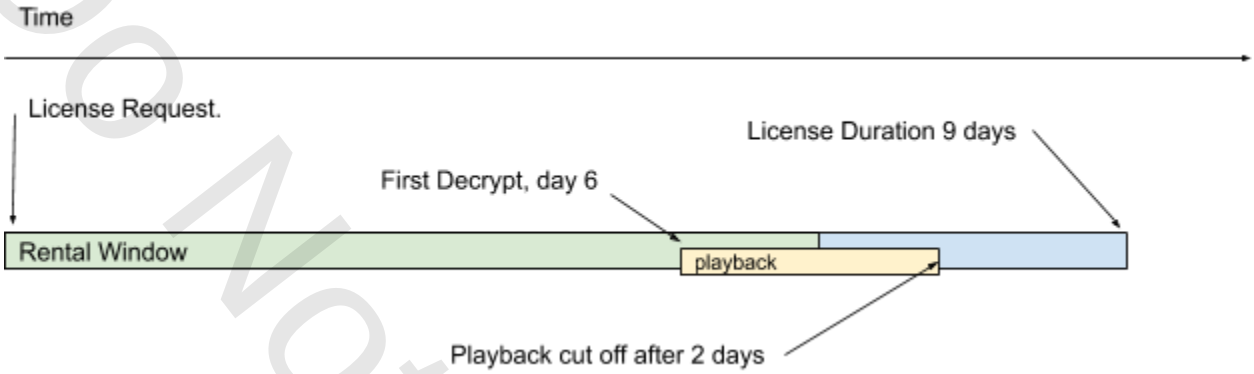
Limited First Start and Duration

For example, you have 7 days to start watching, and 2 days to finish once you start.

earliest_playback_start = 0
latest_playback_start = 7 days

initial_playback_duration = 2 days
license_duration = 9 days
soft_expiry = false

If the user starts watching on day 6, they may continue for 2 days.

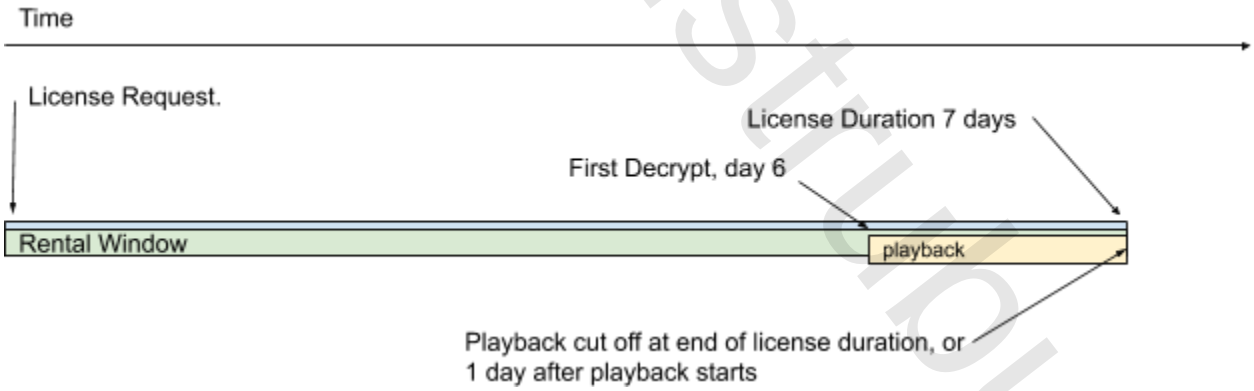


More Strict Limited First Start and Duration

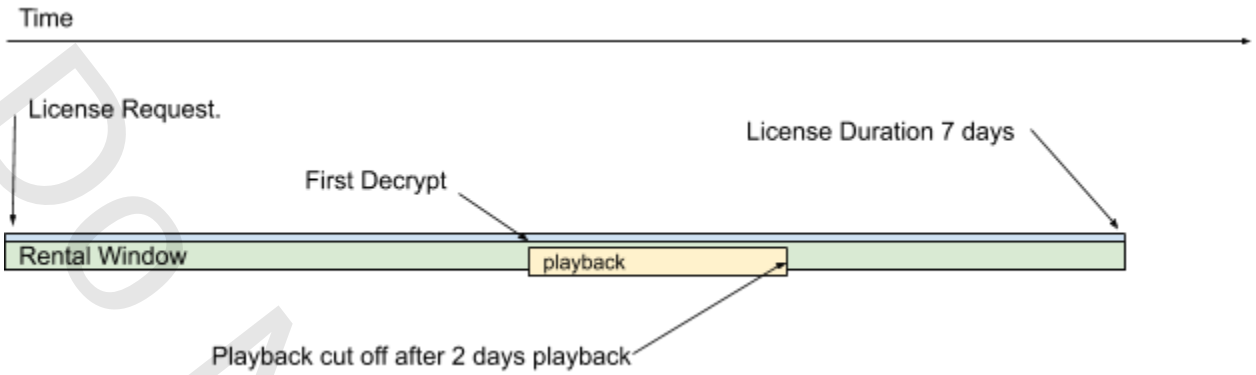
For example, you have 7 days to start and finish watching, and 2 days to finish once you start

earliest_playback_start = 0
latest_playback_start = 7 days
initial_playback_duration = 2 days
license_duration = 7 days
soft_expiry = false

If the user starts watching at day 6, they only have 1 day to finish watching.



If the user starts watching earlier, then they are cut off after 2 days of playback:

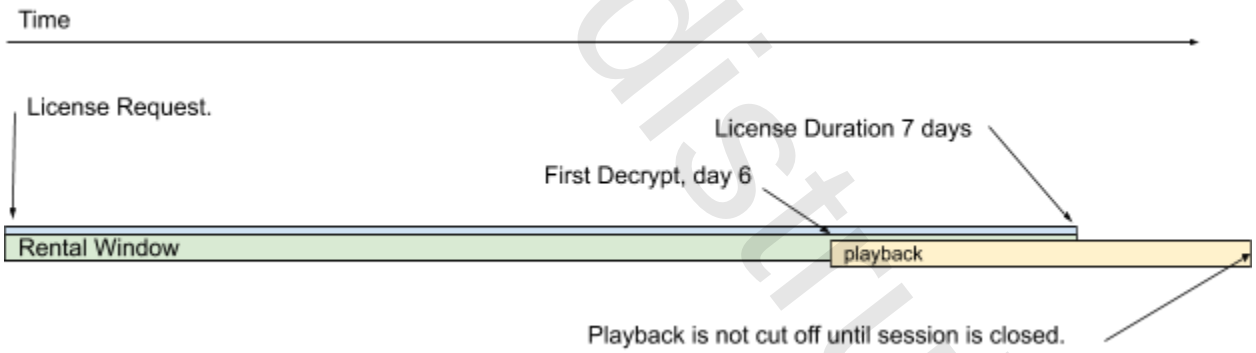


Less Strict Limited First Start and Duration

For example, you have 7 days to start and finish watching, and 2 days to finish once you start

earliest_playback_start = 0
latest_playback_start = 7 days
initial_playback_duration = 2 days
license_duration = 7 days
soft_expiry = **true**

If the user starts watching at day 6, they may continue watching until the current session is closed.

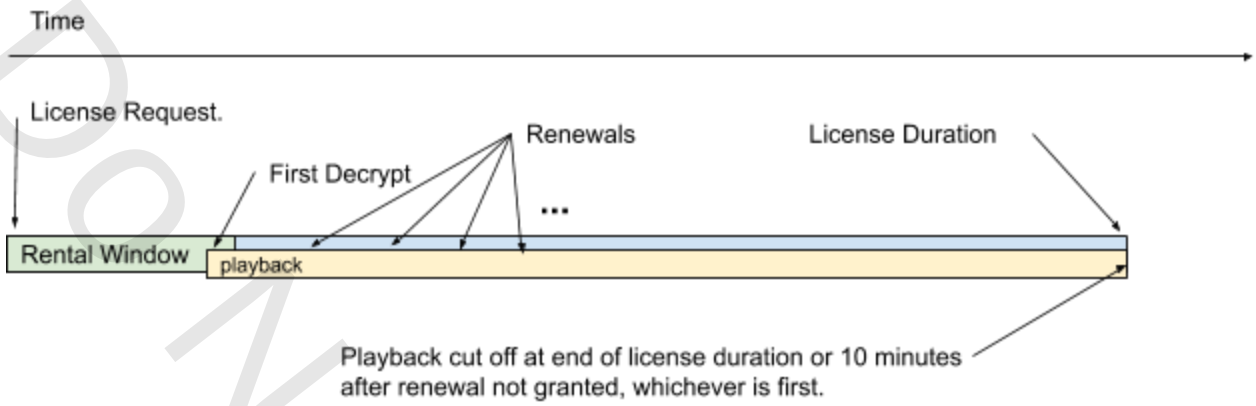


Streaming with Renewal, Hard Expiry

For example, streaming must begin within 15 minutes, and may continue for 12 hours as long as renewals are sent every 10 minutes.

earliest_playback_start = 0
latest_playback_start = 15 minutes
initial_playback_duration = 10 minutes
renewal_playback_duration = 10 minutes
license_duration = 12 hours

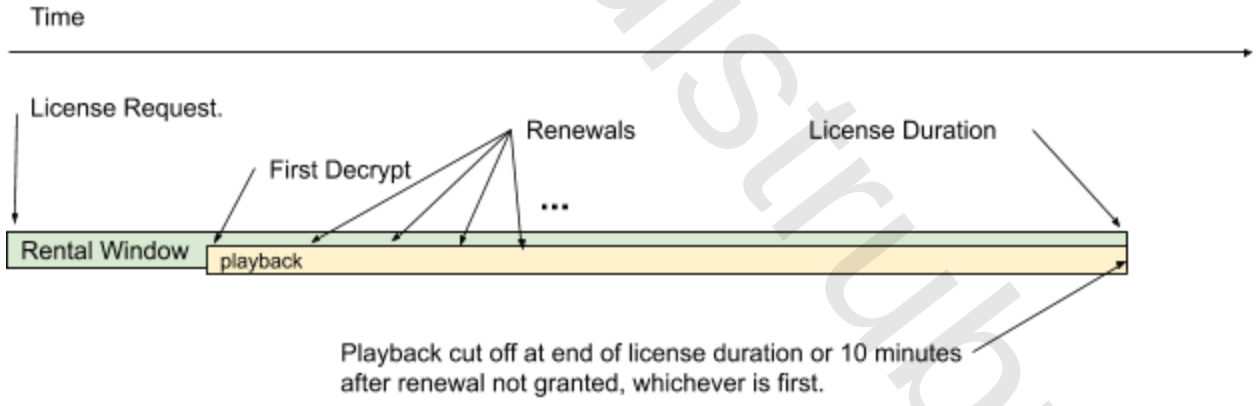
soft_expiry = false



Hard Rental with Renewal

Users have 48 hours to finish watching the video once the license is generated. Renewals are required every hour to continue watching.

earliest_playback_start = 0
latest_playback_start = 48 hours
initial_playback_duration = 1 hour
renewal_playback_duration = 1 hour
license_duration = 48 hours
soft_expiry = false

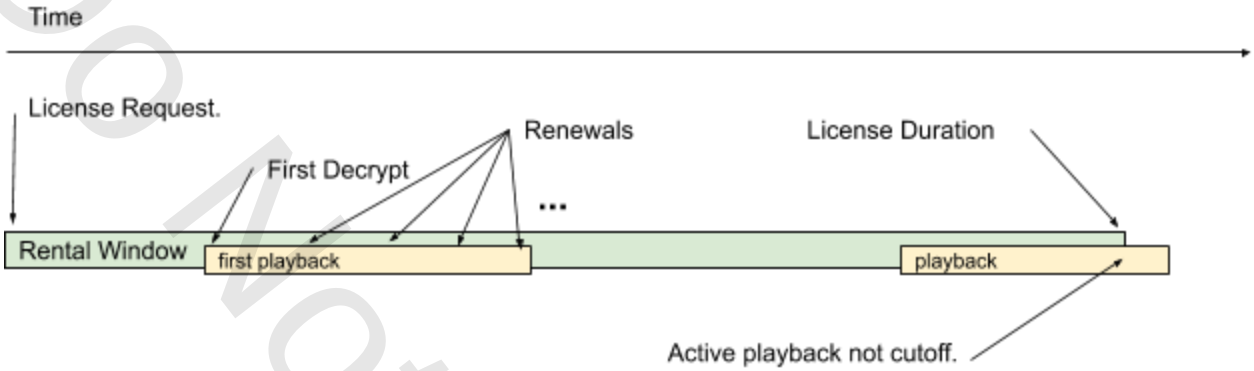


Soft Rental with Renewal

Users have 48 hours to finish watching the video once the license is generated. Renewals are required every hour to continue watching. In this case, renewals are used for accounting, but do not affect enforcement.

earliest_playback_start = 0
latest_playback_start = 48 hours

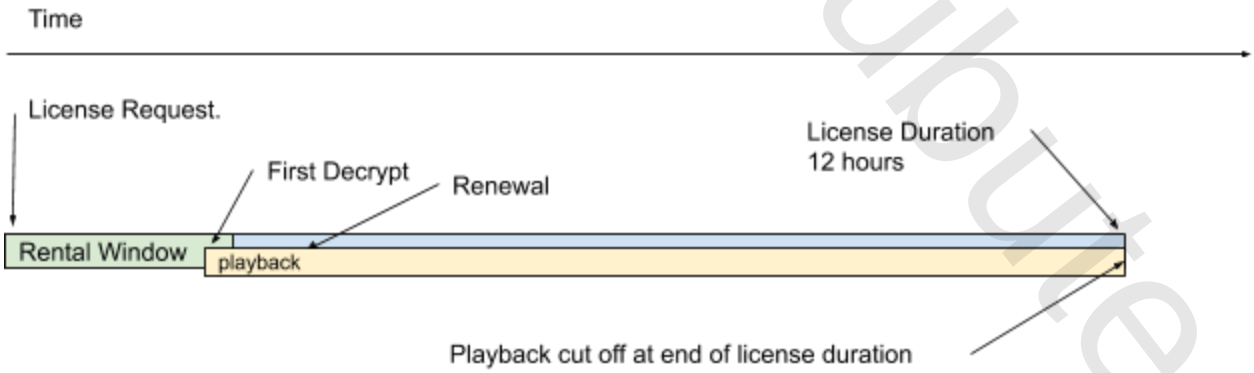
initial_playback_duration = 1 hour
renewal_playback_duration = 1 hour
license_duration = 48 hours
soft_expiry = true



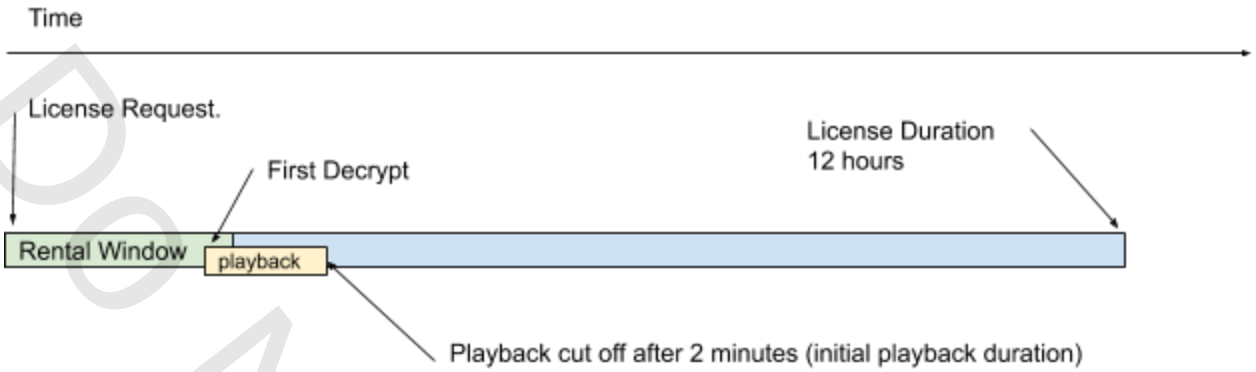
Limited Duration License

For example, the user must begin playback within 15 minutes. Playback must stop within 2 minutes unless a renewal is received. Once a renewal is received, playback may continue for 12 hours.

earliest_playback_start = 0
latest_playback_start = 15 minutes
initial_playback_duration = 2 minutes
renewal_playback_duration = 12 hours
license_duration = 12 hours
soft_expiry = false



If the renewal is not granted, then playback is limited to 2 minutes.



OEMCrypto and ODK Library Functions

OEMCrypto shall use the ODK library functions to maintain timer and clock values. There are two new data structures that shall be part of an OEMCrypto Session. The first is filled out by the function ODK_ParseLicense.

```
typedef struct {
  uint32_t soft_expiry;
  uint64_t earliest_playback_start_seconds;
  uint64_t latest_playback_start_seconds;
  uint64_t initial_playback_duration_seconds;
  uint64_t renewal_playback_duration_seconds;
  uint64_t license_duration_seconds;
} ODK_TimerLimits;
```

The fields in this structure are defined in the core message of the license. This structure should be kept as part of the session and used when calling the ODK timing functions below.

The second data structure keeps information about the current clock values and timers.

```
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;
```

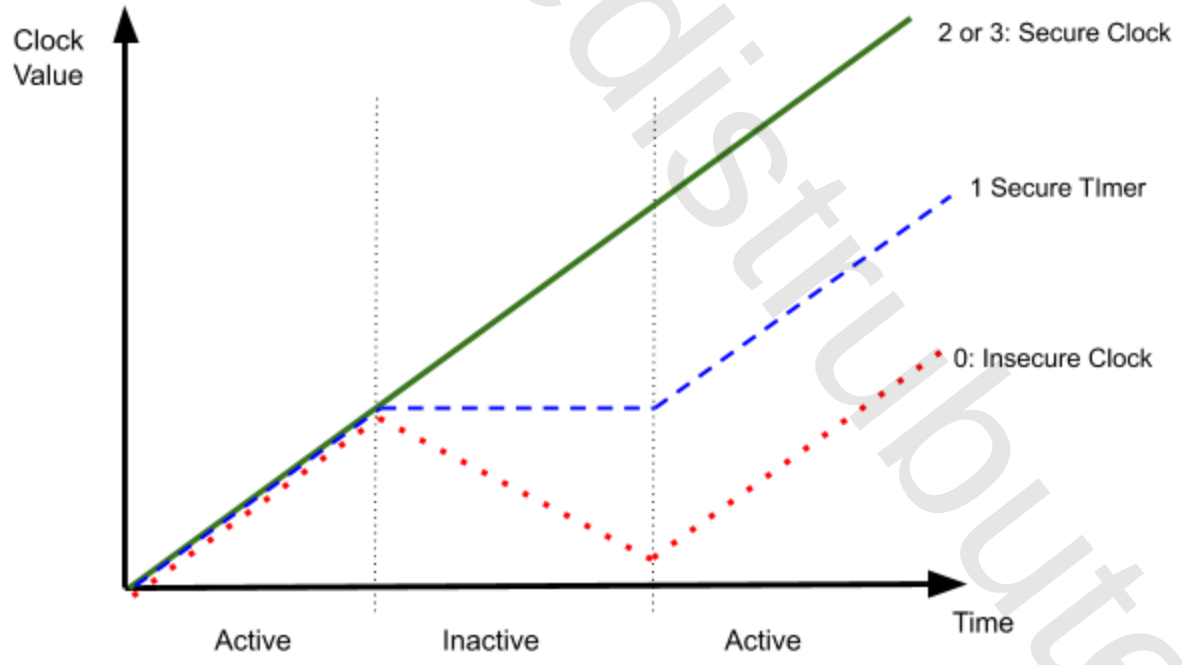
All times are in seconds. Most of the fields in this structure are saved in the usage entry. This structure should be initialized when a usage entry is created or loaded, and should be used to

save a usage entry. It is updated using the ODK functions listed below. The time values are based on OEMCrypto's system clock, as described in the next section.

Timer Implementation

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

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



A common way to implement a secure timer is to periodically save the current time to the secure persistent storage. When the system initializes, it will start the secure timer at the last saved

clock value. For such a design the current time shall be saved at least once every five minutes, approximately.

Vendor Provided Timer

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

Using ODK Timer

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

ODK Time Functions

Setting TImer Limits

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

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

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

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

Setting and Updating Clock Values

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

Initializing Clock Values

The clock values are initialized in the OEMCrypto function
```
OEMCryptoResult OEMCrypto_SignLicenseRequest(...)
```
It does this by calling the ODK function
```
void ODK_InitializeClockValues(ODK_ClockValues* clock_values,
                               uint64_t system_time_seconds);
```
This function initializes the license status to unused and starts the rental clock.

Reloading Clock Values

For an offline license, the clock values are reloaded from the usage entry. When the existing OEMCrypto function OEMCrypto_LoadUsageEntry is called, it shall initialize the session's clock values using the ODK function:
```
void ODK_ReloadClockValues(ODK_ClockValues* clock_values,
                           uint64_t time_of_license_signed,
                           uint64_t time_of_first_decrypt,
                           uint64_t time_of_last_decrypt,
                           enum OEMCrypto_Usage_Entry_Status status,
                           uint64_t system_time_seconds);
```
The time values are taken from the usage entry. The times are based on the system clock.

First Playback -- Enabling Timer

The first time a call is made to OEMCrypto_DecryptCENC, OEMCrypto_Generic_Encrypt, OEMCrypto_Generic_Decrypt, OEMCrypto_Generic_Sign, or OEMCrypto_Generic_Verify for a session, the clock values are updated and the timer may be started. OEMCrypto does this by calling ODK_AttemptFirstPlayback:
```
uint32_t ODK_AttemptFirstPlayback(uint64_t system_time_seconds,
                                  const ODK_TimerLimits* timer_limits,
                                  ODK_ClockValues* clock_values,
                                  uint64_t* timer_value);
```
This updates the clock values, and determines if playback may start based on the given system time. The variables passed in are verified by the ODK function:

- [in] system_time_seconds: the current time on OEMCrypto's clock, in seconds.
- [in] timer_limits - timer limits specified in the license.
- [in/out] clock_values: the sessions clock values.
- [out] timer_value: set to the new timer value. Only used if the return value is ODK_SET_TIMER.

ODK_AttemptFirstPlayback returns:

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

During Playback -- Updating Timer

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

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

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

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

The function OEMCrypto_UpdateUsageEntry changes:

```
OEMCryptoResult OEMCrypto_UpdateUsageEntry(...)
```

This function should call the function ODK_UpdateLastPlaybackTime described above, and then copy the values from the clock_values to the usage entry.

Resetting Timer

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

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

Usage Entry Updates

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

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

This function updates the status in clock_values.