WIDEVINE

License Duration and Renewal Updates

April 6th, 2019

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

Introduction and Audience 2

Control Fields 2

Soft or Hard Expiry 2

License Server SDK Fields 3

Use Cases 3

Streaming 3

Streaming (Quick Start) 4

Seven Day / Two Day -- Hard Rental Expiry 4

Hard Playback Expiry 4

Soft Playback Expiry 5

Seven Day / Two Day -- Soft Rental 6

Hard Playback Expiry 6

Soft Playback Expiry 7

License with Renewal 8

Limited Duration License 9

Protobuf Fields 10

Design Change 11

Goals 11

OEMCrypto Core License Fields 11

OEMCrypto and ODK Library Functions 12

Clocks and Timers 13

OEMCrypto implementers shall use the ODK library to implement these clocks and timers on top of a single system clock. This is described below. 13

Timer Implementation 13

Vendor Provided Timer 14
Using ODK Timer 15
ODK Time Functions 15
Setting TImer Limits 15
Setting and Updating Clock Values 15
Initializing Clock Values 15
Reloading Clock Values 16
First Playback -- Enabling Timer 16
During Playback -- Updating Timer 17
Resetting Timer 17
Usage Entry Updates 17
Backwards Compatibility **18**
Clear Lead **18**

Introduction and Audience

This document describes the time restrictions on a Widevine license. It is part of the OEMCrypto v16 design, but it also discusses the control fields that are accessible to content providers using the license server SDK. As such, the audience is content providers who wish to understand how durations are enforced; device makers who wish to understand how the ODK library works; and, of course, Widveine engineers who are implementing the ODK library. Content providers may wish to concentrate on the “Use Cases” section. Device makers may wish to concentrate on the section “OEMCrypto and ODK Library Functions”.

Control Fields

There are several fields in the license server SDK that are available to content providers for controlling time restrictions. These are used to fill out fields in the license protobuf message that is sent to the CDM.

Soft or Hard Expiry

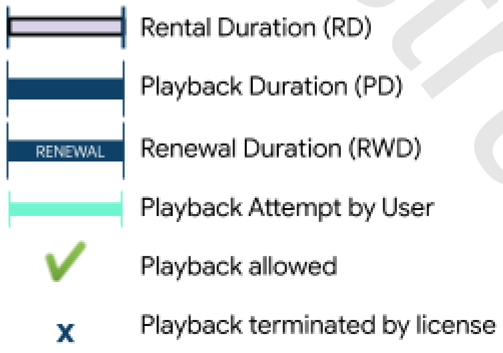
We discuss rental and playback durations below. Each of these can have either a soft or a hard expiry. A soft expiry means that playback will not be cutoff in the middle of an active session. A hard expiry means that durations are strictly enforced. The choice of whether to use soft or hard expiry is a business decision made by the content providers or the studios. Examples are given below in the section on Use Cases.

License Server SDK Fields

The following fields are available to content providers from the SDK for controlling a license's time restrictions. A license renewal only controls the renewal_delay_seconds value.
license_start_time Playback may not begin before this. This defaults to the time the server grants the request.
rental_duration_seconds - Time limit on starting playback. This is measured in seconds from license_start_time. Initial playback may not begin after this time. If the rental duration has hard expiry, then playback may not continue after this time.
playback_duration_seconds - Time limit on playback, starting from first playback. If playback expiry is hard, playback may not start or continue after this time. If a license is persistent, playback may not restart after this time.
renewal_delay_seconds - time to first renewal. (Also called the license renewal interval)
renewal_recovery_duration_seconds - time allowed for renewal in flight.
soft_enforce_playback_duration (default = false) -- If true, playback is not stopped when playback duration expires .
soft_enforce_rental_duration -- (default = true) -- If true, this only restricts the first playback. Playback duration may still cause a hard stop.
can_persist = determines if a license can be loaded multiple times. We frequently call this an offline license. However, this does not control if the content itself is stored on the device -- it only controls if the license is stored.

Use Cases

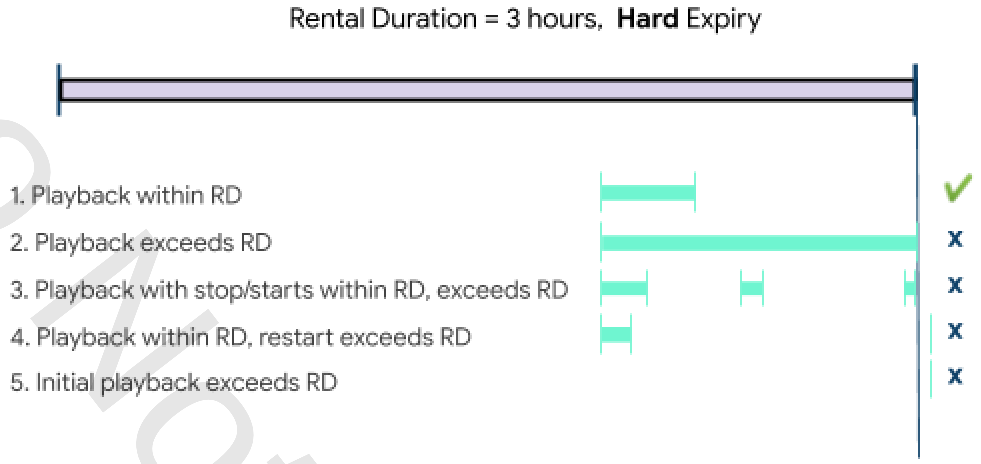
The most common use cases are as follows. Usually, license_start_time = now, and renewals are not used.
The following diagrams use the following legend.



Streaming

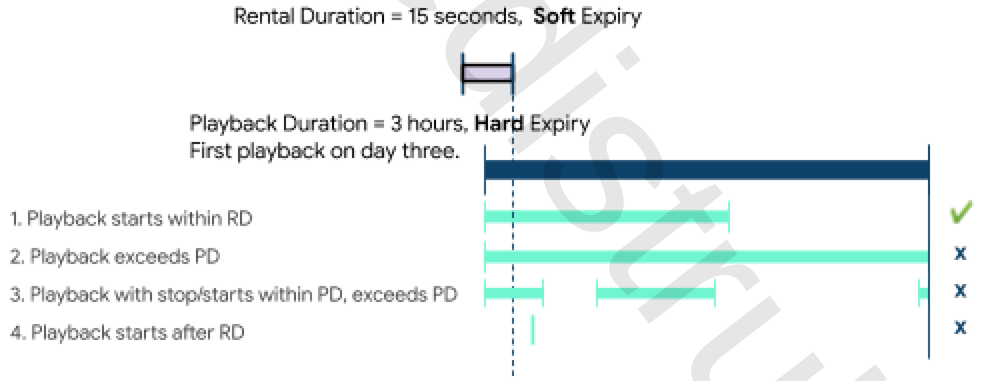
This is the simplest use case. The user has three hours to watch the movie from the time of rental.

rental_duration_seconds (RD) = 3 hours hard
playback_duration_seconds (PD) = 0 (unlimited)
soft_enforce_rental_duration = false


Rental Duration = 3 hours, **Hard** Expiry
1. Playback within RD
✓
2. Playback exceeds RD
x
3. Playback with stop/starts within RD, exceeds RD
x
4. Playback within RD, restart exceeds RD
x
5. Initial playback exceeds RD
x

Streaming (Quick Start)

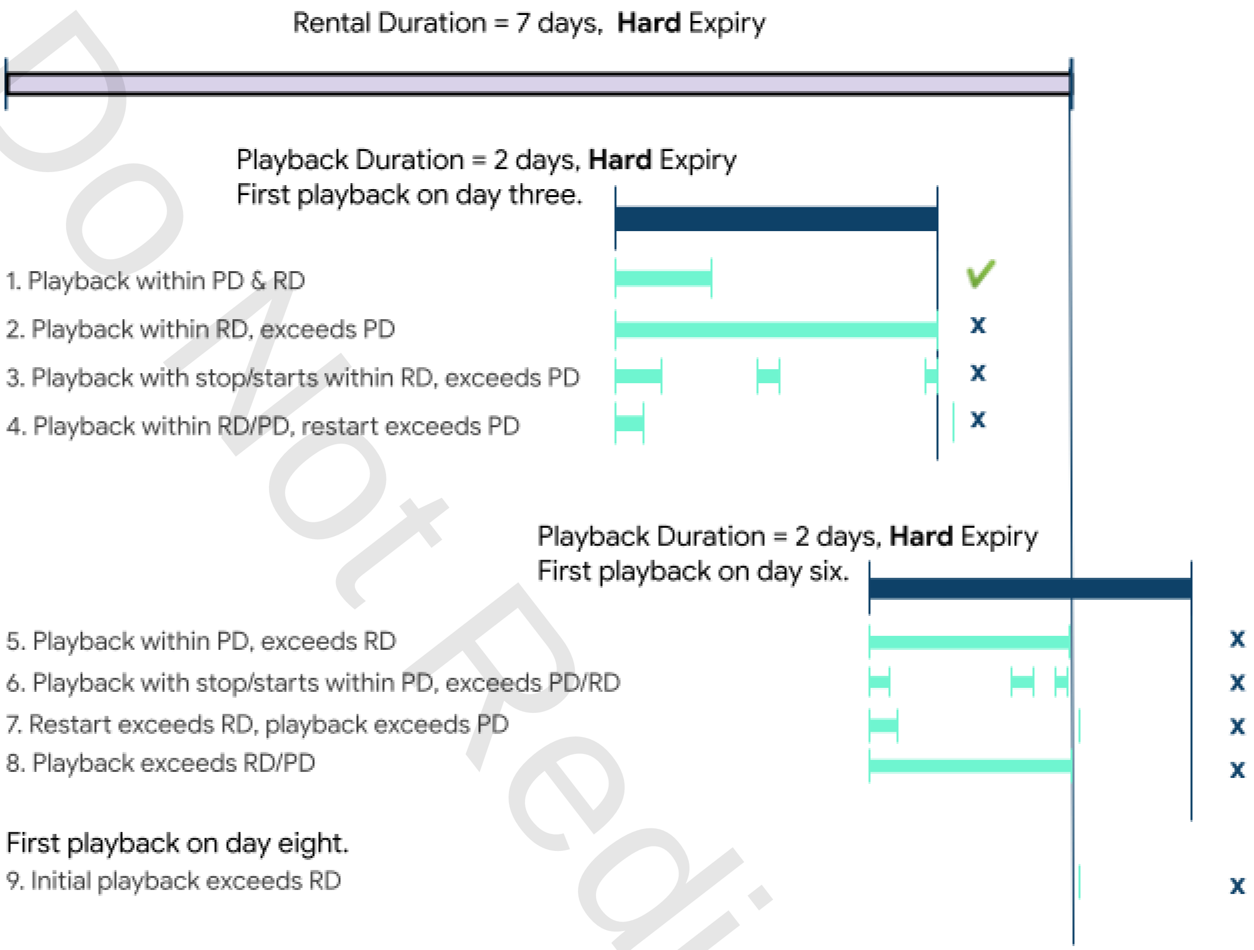
The user must start watching within 30 seconds, and then has three hours to finish.
rental_duration_seconds (RD) = 30 seconds, soft
playback_duration_seconds (PD) = 3 hours, hard
soft_enforce_rental_duration = true
soft_enforce_playback_duration = false


Rental Duration = 15 seconds, **Soft** Expiry
Playback Duration = 3 hours, **Hard** Expiry
First playback on day three.
1. Playback starts within RD
✓
2. Playback exceeds PD
x
3. Playback with stop/starts within PD, exceeds PD
x
4. Playback starts after RD
x

Seven Day / Two Day -- Hard Rental Expiry

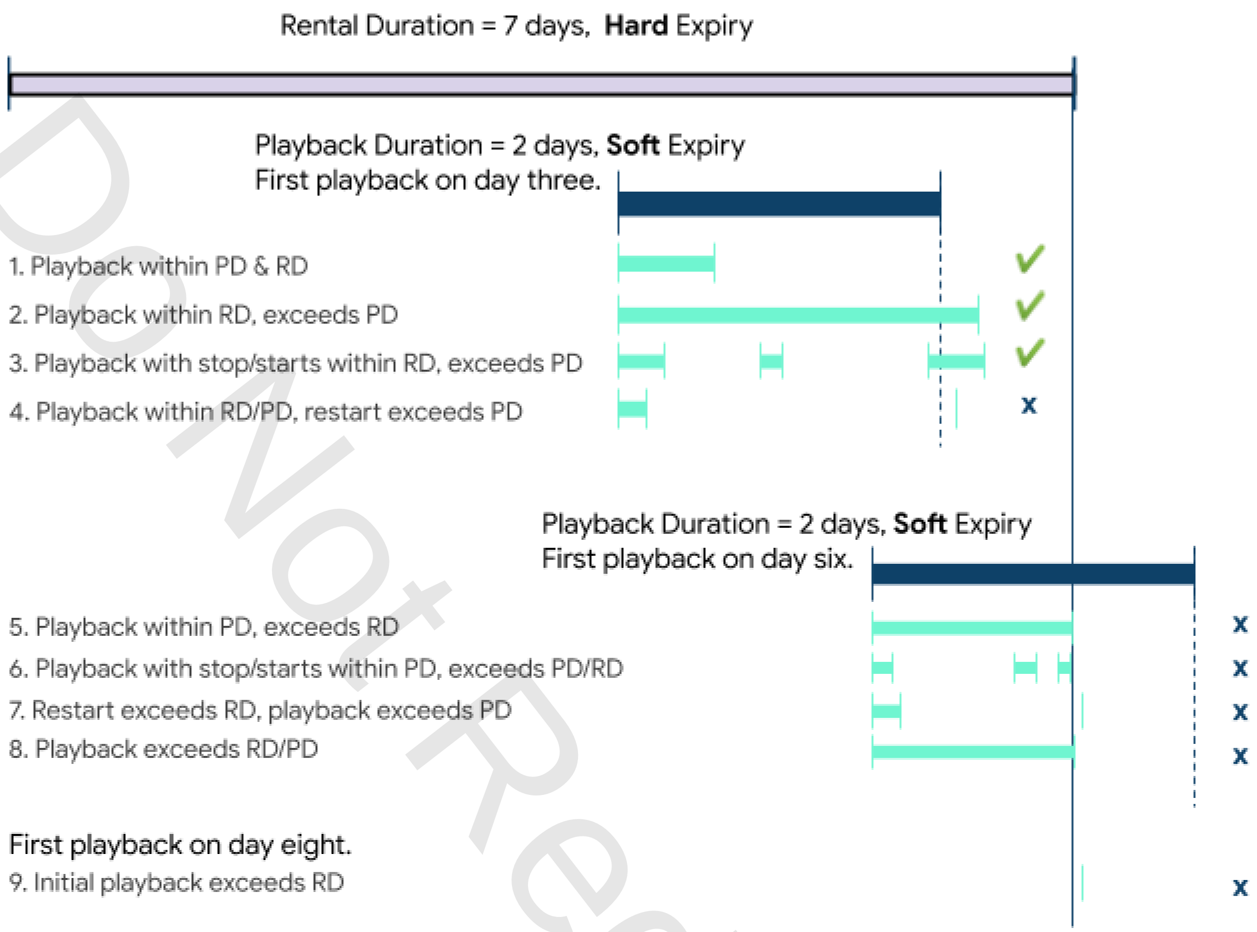
Hard Playback Expiry

When testing, these use cases will be grouped together as “SevenHardTwoHard”.
rental_duration_seconds (RD) = 7 days
playback_duration_seconds (PD) = 2 days
soft_enforce_rental_duration = false
soft_enforce_playback_duration = false


Rental Duration = 7 days, **Hard** Expiry
Playback Duration = 2 days, **Hard** Expiry
First playback on day three.
1. Playback within PD & RD
✓
2. Playback within RD, exceeds PD
x
3. Playback with stop/starts within RD, exceeds PD
x
4. Playback within RD/PD, restart exceeds PD
x
Playback Duration = 2 days, **Hard** Expiry
First playback on day six.
5. Playback within PD, exceeds RD
x
6. Playback with stop/starts within PD, exceeds PD/RD
x
7. Restart exceeds RD, playback exceeds PD
x
8. Playback exceeds RD/PD
x
First playback on day eight.
9. Initial playback exceeds RD
x

Soft Playback Expiry

When testing, these use cases will be grouped together as “SevenHardTwoSoft”.
rental_duration_seconds (RD) = 7 days
playback_duration_seconds (PD) = 2 days
soft_enforce_rental_duration = false
soft_enforce_playback_duration = true

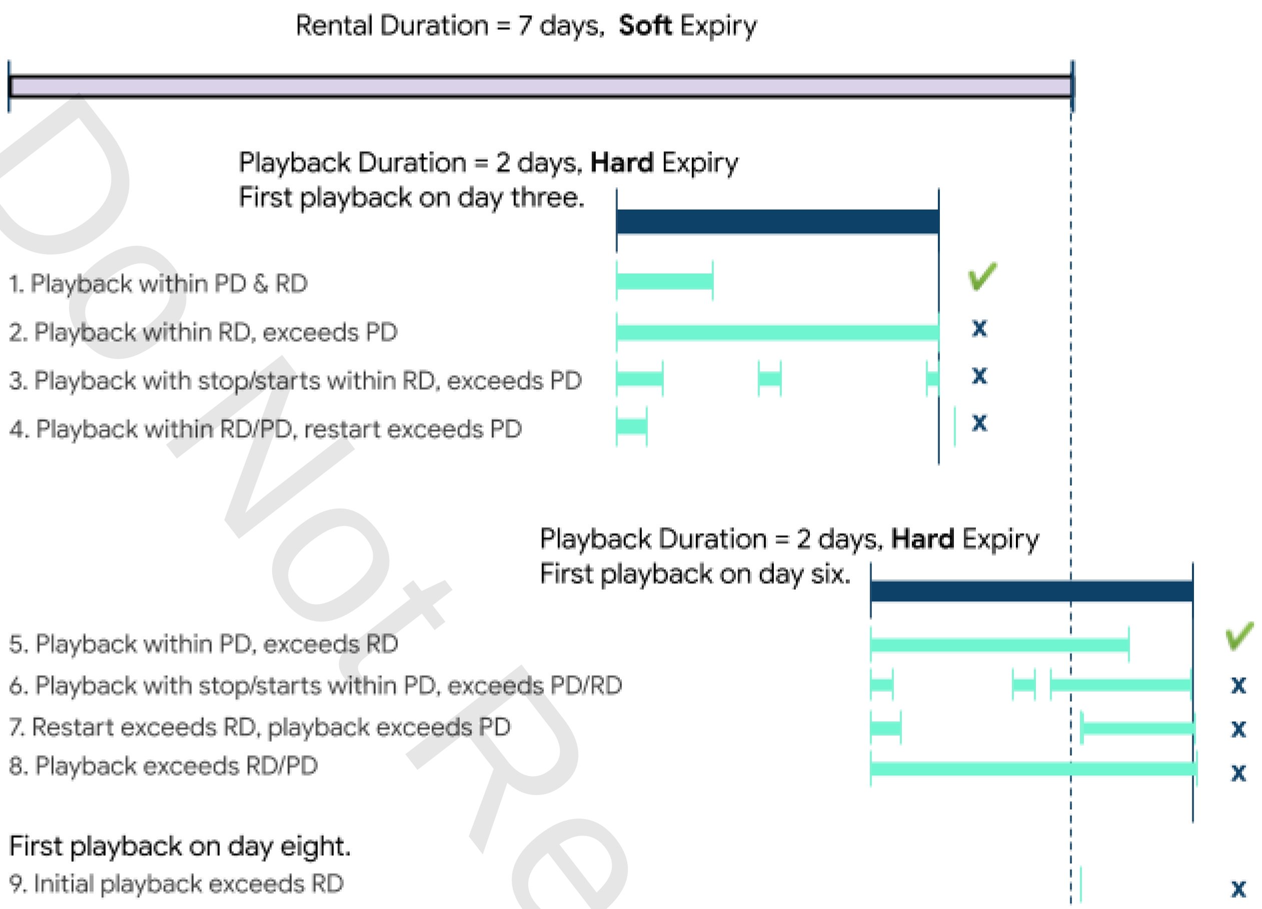

Rental Duration = 7 days, **Hard** Expiry
Playback Duration = 2 days, **Soft** Expiry
First playback on day three.
1. Playback within PD & RD
2. Playback within RD, exceeds PD
3. Playback with stop/starts within RD, exceeds PD
4. Playback within RD/PD, restart exceeds PD
✓
✓
✓
x
Playback Duration = 2 days, **Soft** Expiry
First playback on day six.
5. Playback within PD, exceeds RD
6. Playback with stop/starts within PD, exceeds PD/RD
7. Restart exceeds RD, playback exceeds PD
8. Playback exceeds RD/PD
x
x
x
x
First playback on day eight.
9. Initial playback exceeds RD
x

Notice that in case 4, soft expiry only affects continued playback -- the user cannot restart playback.

Seven Day / Two Day -- Soft Rental

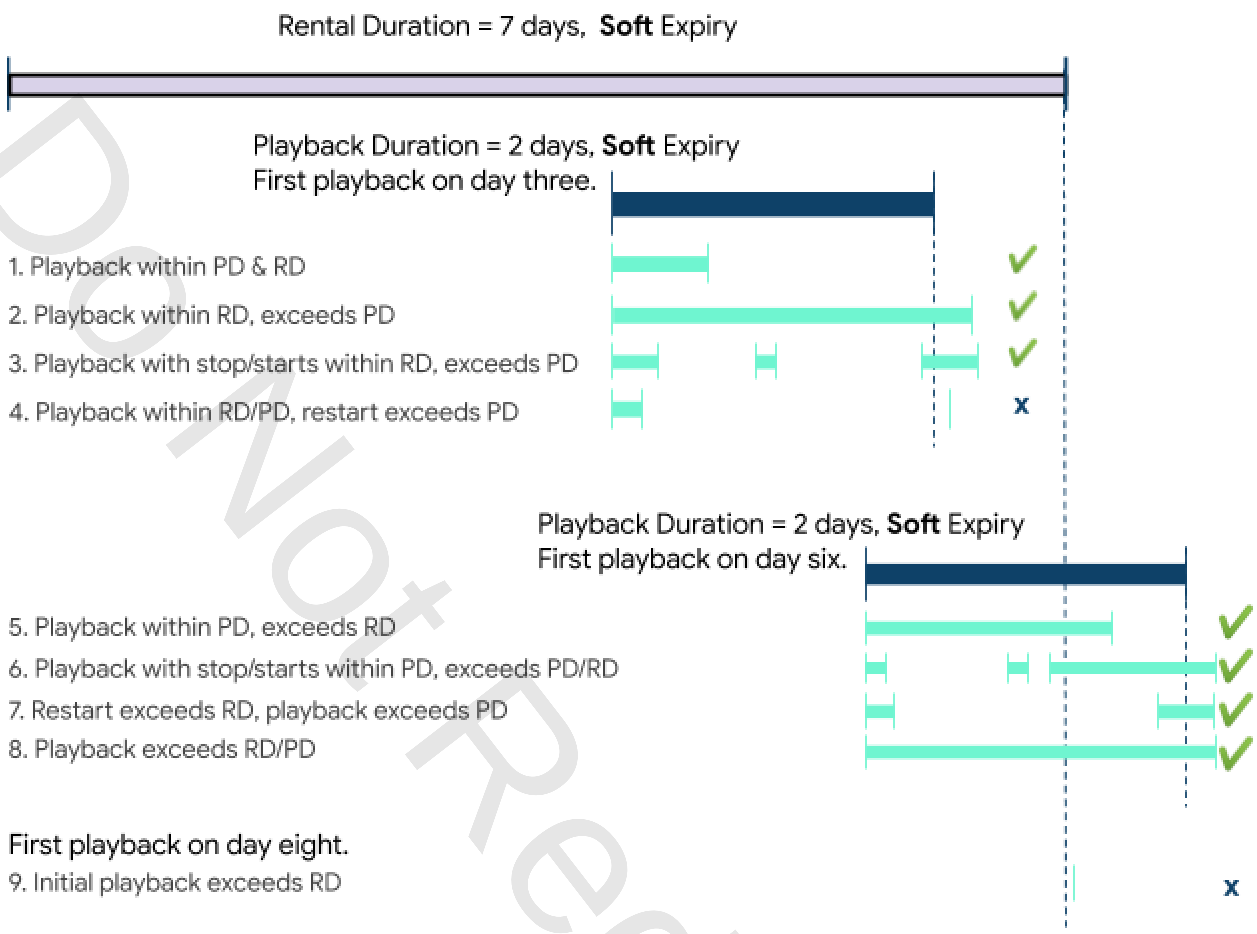
Hard Playback Expiry

When testing, these use cases will be grouped together as “SevenSoftTwoHard”.
rental_duration_seconds (RD) = 7 days
playback_duration_seconds (PD) = 2 days
soft_enforce_rental_duration = true
soft_enforce_playback_duration = false


Rental Duration = 7 days, Soft Expiry
Playback Duration = 2 days, Hard Expiry
First playback on day three.
1. Playback within PD & RD
2. Playback within RD, exceeds PD
3. Playback with stop/starts within RD, exceeds PD
4. Playback within RD/PD, restart exceeds PD
Playback Duration = 2 days, Hard Expiry
First playback on day six.
5. Playback within PD, exceeds RD
6. Playback with stop/starts within PD, exceeds PD/RD
7. Restart exceeds RD, playback exceeds PD
8. Playback exceeds RD/PD
First playback on day eight.
9. Initial playback exceeds RD
✓
x
x
x
✓
x
x
x
x

Soft Playback Expiry

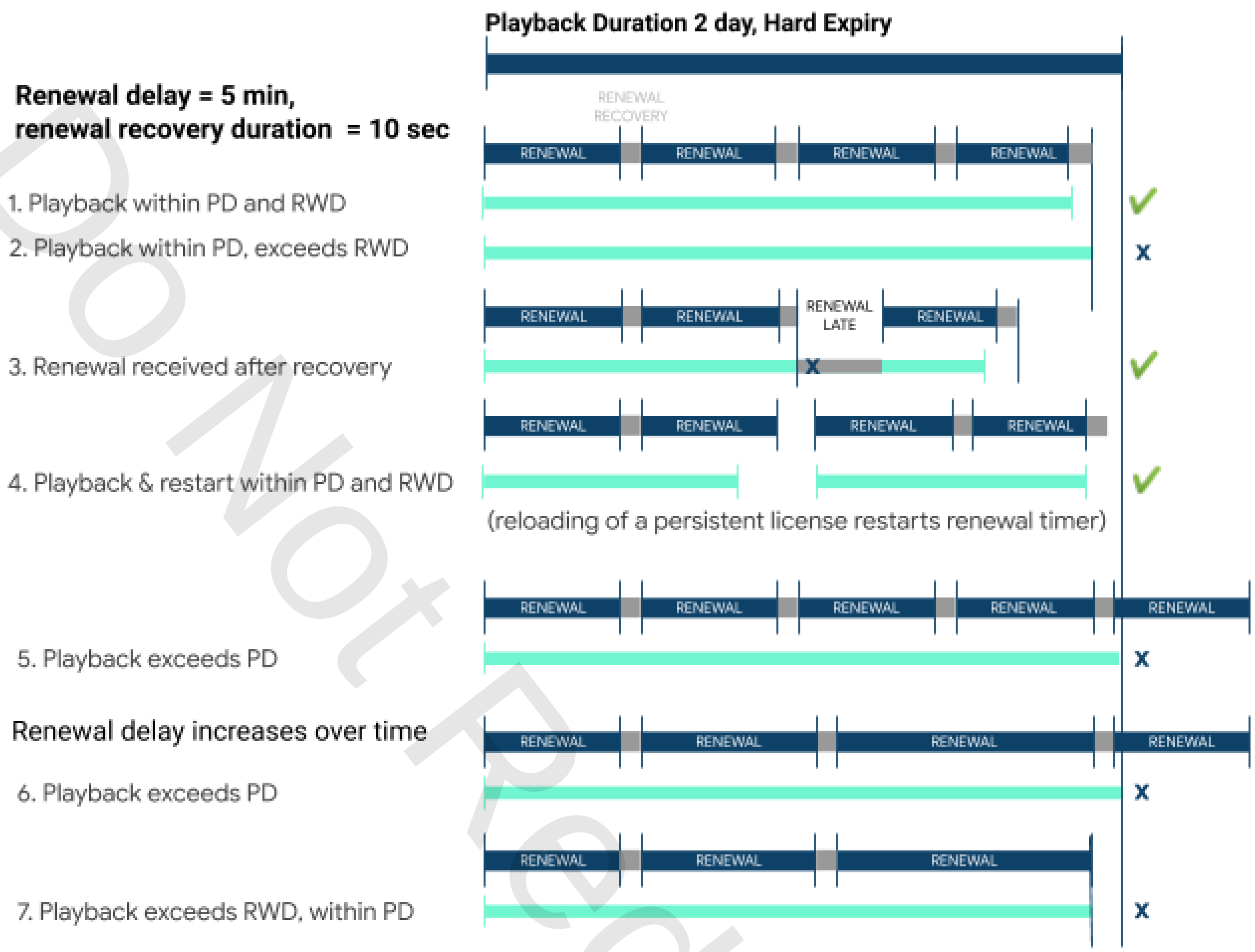
When testing, these use cases will be grouped together as “SevenSoftTwoSoft”.
rental_duration_seconds (RD) = 7 days
playback_duration_seconds (PD) = 2 days
soft_enforce_rental_duration = true
soft_enforce_playback_duration = true


Rental Duration = 7 days, Soft Expiry
Playback Duration = 2 days, Soft Expiry
First playback on day three.
1. Playback within PD & RD
2. Playback within RD, exceeds PD
3. Playback with stop/starts within RD, exceeds PD
4. Playback within RD/PD, restart exceeds PD
x
Playback Duration = 2 days, Soft Expiry
First playback on day six.
5. Playback within PD, exceeds RD
6. Playback with stop/starts within PD, exceeds PD/RD
7. Restart exceeds RD, playback exceeds PD
8. Playback exceeds RD/PD
First playback on day eight.
9. Initial playback exceeds RD
x

Notice that in case 4, soft expiry only affects continued playback -- the user cannot restart playback.

License with Renewal

playback_duration_seconds (PD) = 2 days
soft_enforce_playback_duration = false
renewal_delay_seconds = 5 minutes


Playback Duration 2 day, Hard Expiry
Renewal delay = 5 min,
renewal recovery duration = 10 sec
RENEWAL RECOVERY
RENEWAL
1. Playback within PD and RWD
2. Playback within PD, exceeds RWD
RENEWAL LATE
3. Renewal received after recovery
4. Playback & restart within PD and RWD
(reloading of a persistent license restarts renewal timer)
5. Playback exceeds PD
Renewal delay increases over time
6. Playback exceeds PD
7. Playback exceeds RWD, within PD
✓
x

If rental duration is set to hard, cases 1-5 have the same outcome if you swap the playback duration threshold with rental duration threshold.
Note: in case 3, playback is not allowed if the renewal message is late. However, it may continue once the renewal is loaded. The license does not need to be reloaded.

Limited Duration License

A limited duration license restricts the playback time to a very short duration until a renewal is received. There will be only one renewal.

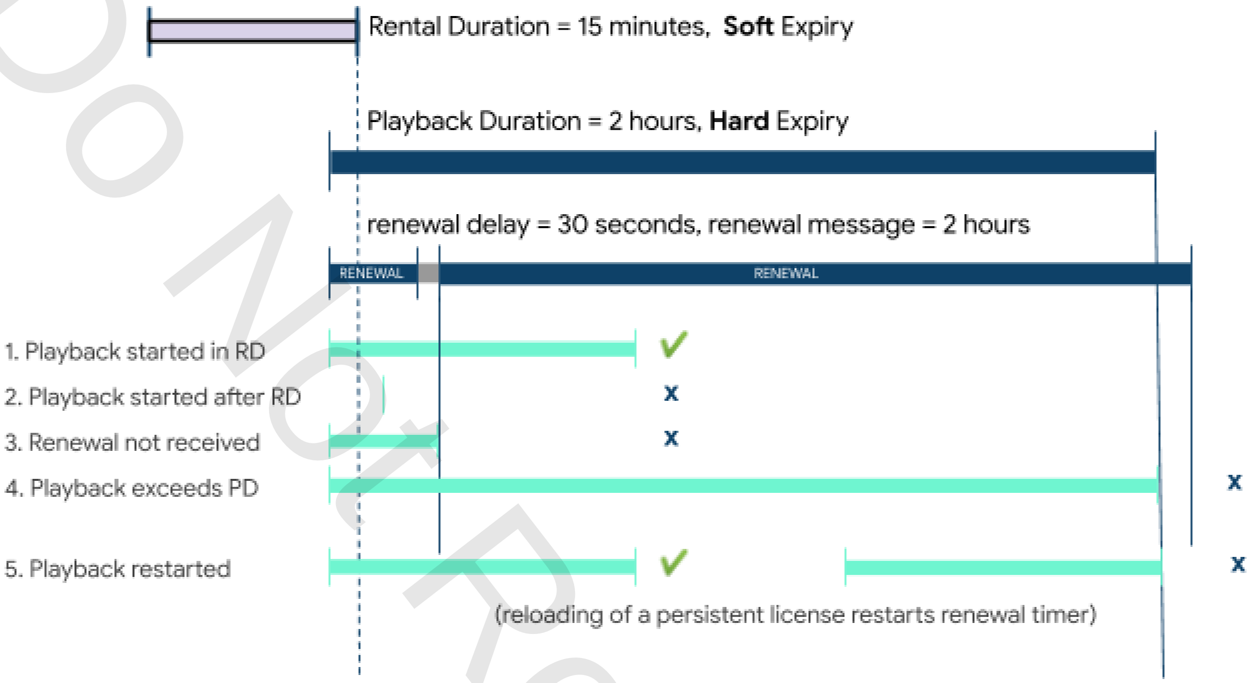
License Message Values:

rental_duration_seconds (RD) = 15 minutes. playback must start within the first 15 minutes.
playback_duration_seconds (PD) = 2 hours.
soft_enforce_rental_duration = true. playback may be restarted any time within playback duration.
soft_enforce_playback_duration = false. Latest playback is 2 hours and 15 minutes.
renewal_delay_seconds = 30 seconds.

Renewal Message Values:

renewal_delay_seconds (RWD) = 2 hours.

can_renew = false


Rental Duration = 15 minutes, **Soft** Expiry
Playback Duration = 2 hours, **Hard** Expiry
renewal delay = 30 seconds, renewal message = 2 hours
RENEWAL
RENEWAL
1. Playback started in RD
✓
2. Playback started after RD
x
3. Renewal not received
x
4. Playback exceeds PD
x
5. Playback restarted
✓
x
(reloading of a persistent license restarts renewal timer)

Protobuf Fields

The license message itself is a protobuf. It has several fields that are derived from those discussed above in the License Server SDK. These fields are enforced by the CDM.

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.

Design Change

This section describes how the design has changed from OEMCrypto v15 to v16.

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 Widedvine use cases.

OEMCrypto Core License Fields

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 had not been passed down to OEMCrypto for enforcement in the TEE before v16. 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.

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.

- **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. The server computes this to be the difference between license_start_time and the time the license is generated.
- **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.

- **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_rental_duration is false, then this restricts any playback. The server sets this to the playback_duration_seconds.
- **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. The server sets this to the sum of renewal_delay_seconds and renewal_recovery_duration_seconds.

The server bases the earliest_playback_start_seconds value on the time the license was generated, and the client bases the value on the time the license was requested. These times are normally a few seconds off due to message transit times. This is only exploitable by an attacker if the start time is in the future. It can hurt the user if the license request is very slow in transit, because the device considers that time as part of the rental duration and the server does not.

OEMCrypto and ODK Library Functions

OEMCrypto shall use the ODK library functions to maintain timer and clock values, and to compute cutoff times for the playback timer.

There are two new data structures related to duration and time that shall be part of an OEMCrypto Session.

```
ODK_TimerLimits timer_limits;
```

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

```
ODK_ClockValues clock_values;
```

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

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

Clocks and Timers

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. Conceptually, there are two clocks and one timer used by the ODK library to restrict playback:

- **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 duration -- 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 used to enforce the playback duration. 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 and renewal duration. On the first decrypt call in a session, it is initialized based on the various rental and playback duration restrictions, as shown in the use cases above. When a renewal is loaded, the playback timer is reset to the value specified by the renewal. The playback timer is not stored in the usage table entry.

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

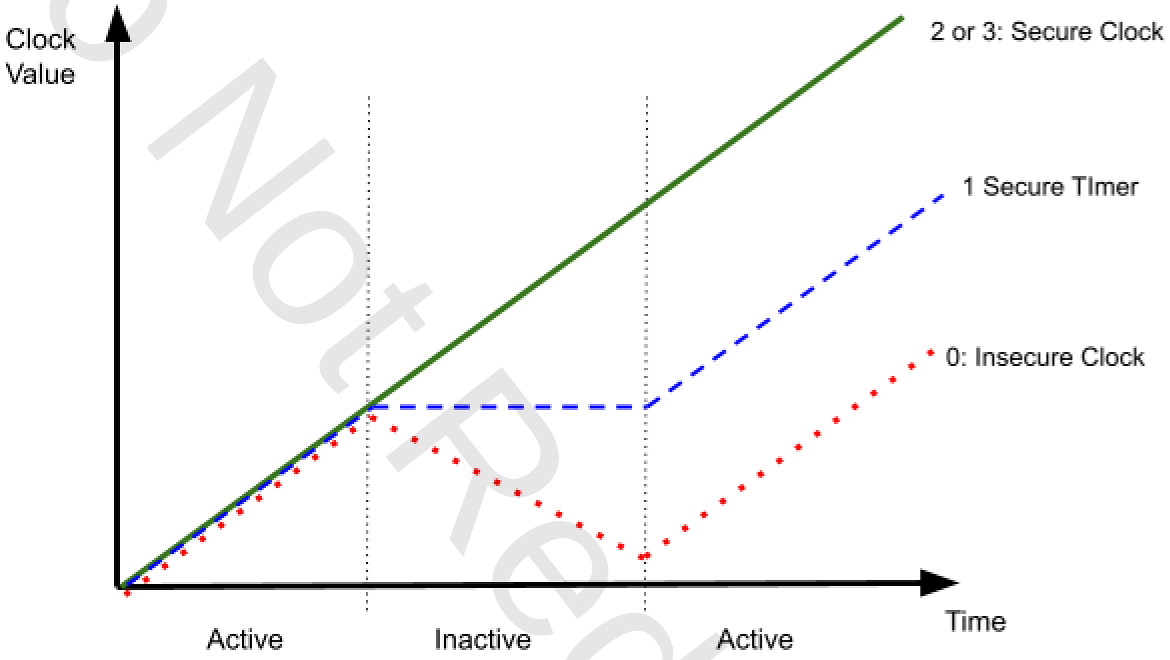
Timer Implementation

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 video 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 set the session's timer limits.

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

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

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

Setting and Updating Clock Values

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

Initializing Clock Values

The clock values are initialized in the OEMCrypto function

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

It does this by calling the ODK function

```
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.

An implementation of OEMCrypto may choose to call ODK_AttemptFirstPlayback from the first call to OEMCrypto_SelectKey instead of the first decrypt call. This may be done in order to optimize the decrypt calls. It is important that an implementation that does this will not allow keys to be used before they are selected.

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". They determine if playback is allowed, and compute the new value for the playback timer.
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.

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 key duration in the key control block. It will use
soft_enforce_rental_duration = false
soft_enforce_playback_duration = false
earliest_playback_start_seconds = 0
rental_duration_seconds = 0
total_playback_duration_seconds = 0
initial_renewal_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.

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.
For performance reasons, implementations may use the first call to OEMCrypto_SelectKey instead of a decrypt call. Under normal situations, any call to SelectKey will be followed immediately by a decrypt call. It is the OEMCrypto implementer's responsibility to verify that

decryption cannot occur until after a call to SelectKey. In particular, loading a license **shall not** install a default key.