DASH-IF implementation guidelines: content protection and security

Commit Snapshot, 15 July 2020

This version:
https://dashif-documents.azurewebsites.net/Guidelines-Security/master/Guidelines-Security.html

Issue Tracking:
GitHub
Inline In Spec

Editors:
DASH Industry Forum

Table of Contents

1. Purpose§

The guidelines defined in this document support the creation of protected interoperable services for high-quality video distribution based on MPEG-DASH and related standards. These guidelines are provided in order to address DASH-IF members' needs and industry best practices. The guidelines support the implementation of conforming service offerings as well as DASH client implementations.

While alternative interpretations may be equally valid in terms of standards conformance, services and clients created following the guidelines defined in this document can be expected to exhibit highly interoperable behavior between different implementations.

2. Scope§

This document is an update to the "Content Protection and Security" section of the DASH-IF IOP Guidelines version 4.3. The scope remains the same, giving guidelines for interoperable behaviors of clients in front of well formed encrypted content. This means:

- Updated encrypted content constraints for supporting CMAF. This includes the addition of the `cbcs` scheme support and recommendation for encrypting content when available using both `cbcs` and `cenc` protection schemes. Note that compared to DASH-IF IOP 4.3, there are no changes in the recommendations for using `default_KID` and `pssh` elements.
- Added discussions on compliance and robustness rules and their impact on the choices of the DRM client to instantiate.
- Clarified periodic reauthorization mechanisms, separating the topic of key hierarchy from periodic reauthentication - the two are now separate chapters.
- Clarified the client reference architecture which is an MSE/EME type of player, more precisely connecting between the DASH/DASH-IF/CMAF content format specifications and W3C EME.

In addition, this document:

- Introduces the **Interoperable license request model** that describes how players take content and consume it in ways that make sense on a platform that supports EME. From the Platform capabilities discovery and DRM selection to the license request protocol, this optional request model allows a player to obtain authorization tokens that can be used for retrieving licenses and content keys from a license server for rendering content. Any processing step in the proposed model can be redefined by the application logic.
- Introduces DASH-IF XML schema where two elements are defined for supporting the license request model. These elements are namely the `laurl` (license acquisition server URL) and `authzurl` (Authorization server URL).

3. Interpretation§

Requirements in this document describe service and client behaviors that DASH-IF considers interoperable.

If a **service provider** follows these requirements in a published DASH service, the published DASH service is likely to experience successful playback on a wide variety of clients and exhibit graceful degradation when a client does not support all features used by the service.

If a **client implementer** follows the client-oriented requirements described in this document, the DASH client will play content conforming to this document provided that the client device media platform supports all features used by a particular DASH service (e.g. the codecs and DRM systems).

This document uses statements of fact when describing normative requirements defined in referenced specifications such as [DASH] and [CMAF]. References are typically provided to indicate where the requirements are defined.

[RFC2119] statements (e.g. "SHALL", "SHOULD" and "MAY") are used when this document defines a new requirement or further constrains a requirement from a referenced document.

¶EXAMPLE 1
Statement of fact:

- A DASH presentation **is** a sequence of consecutive non-overlapping periods [DASH].

New or more constrained requirement:

- Segments **SHALL NOT** use the MPEG-2 TS container format.

There is no strict backward compatibility with previous versions - best practices change over time and what was once considered sensible may be replaced by a superior approach later on. Therefore, clients and services that were conforming to version N of this document are not guaranteed to conform to version N+1.

4. Disclaimer§

This is a document made available by DASH-IF. The technology embodied in this document may involve the use of intellectual property rights, including patents and patent applications owned or controlled by any of the authors or developers of this document. No patent license, either implied or express, is granted to you by this document. DASH-IF has made no search or investigation for such rights and DASH-IF disclaims any duty to do so. The rights and obligations which apply to DASH-IF documents, as such rights and obligations are set forth and defined in the DASH-IF Bylaws and IPR Policy including, but not limited to, patent and other intellectual property license rights and obligations. A copy of the DASH-IF Bylaws and IPR Policy can be obtained at http://dashif.org/.

The material contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS, and the authors and developers of this material and DASH-IF hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of workmanlike effort, and of lack of negligence.

In addition, this document may include references to documents and/or technologies controlled by third parties. Those third party documents and technologies may be subject to third party rules and licensing terms. No intellectual property license, either implied or express, to any third party material is granted to you by this document or DASH-IF. DASH-IF makes no any warranty whatsoever for such third party material.

Note that technologies included in this document and for which no test and conformance material is provided, are only published as a candidate technologies, and may be removed if no test material is provided before releasing a new version of this guidelines document. For the availability of test material, please check http://www.dashif.org.

5. Core concepts of content protection and security§

DASH-IF provides guidelines for using multiple DRM systems to access a DASH presentation by adding encryption signaling and DRM system configuration to DASH content encrypted in conformance to Common Encryption [CENC]. In addition to content authoring guidelines, DASH-IF specifies interoperable workflows for DASH client interactions with DRM systems, platform APIs and external services involved in content protection interactions.

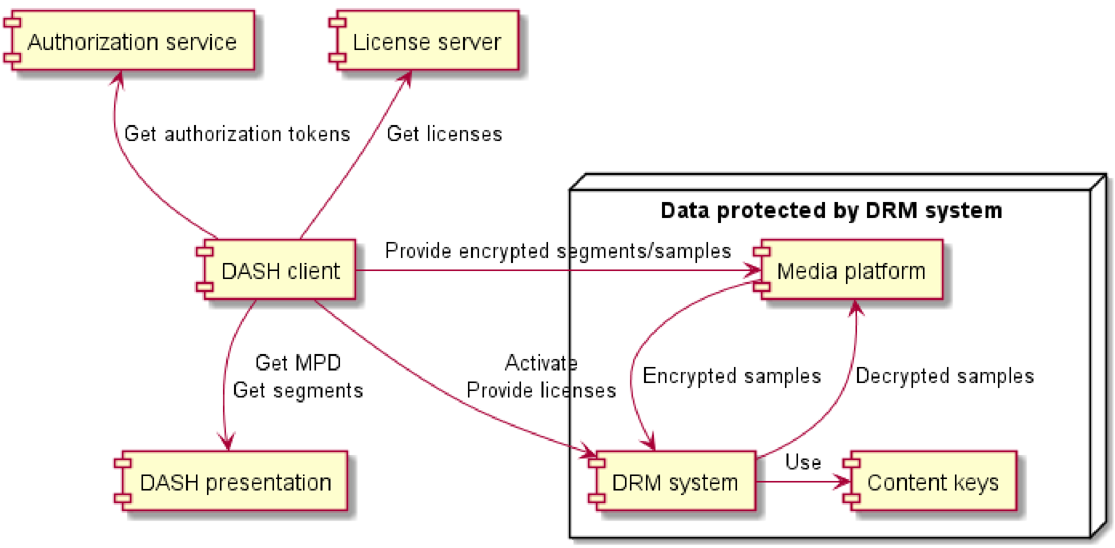


Figure 1 *A DRM system cooperates with the device's media platform to enable playback of encrypted content while protecting the decrypted samples and content keys against potential attacks. The DASH-IF implementation guidelines focus on the signaling in the DASH presentation and the interactions of the DASH client with other components.*

This document does not define any DRM system. DASH-IF maintains a registry of DRM system identifiers on dashif.org.

Common Encryption [CENC] specifies several protection schemes which can be applied by a scrambling system and used by different DRM systems. The same encrypted DASH presentation can be decrypted by different DRM systems if a DASH client is provided the DRM system configuration for each DRM system, either in the MPD or at runtime.

A **content key** is a 128-bit key used by a DRM system to make content available for playback. It is identified by a string called `default_KID` (or sometimes simply KID or "key ID"). The format constraints of the string are defined in [CENC].

> ¶ EXAMPLE 2
> Example `default_KID`: `72c3ed2c-7a5f-4aad-902f-cbef1efe89a9`

While the `default_KID` format visually resembles a UUID, it is not exactly the same. UUIDs have constraints on the byte values permitted at certain positions in the data structure, whereas [CENC] sets no constraints on the values in `default_KID`. [CENC] defines only the format of the string and merely recommends that the value in the string conform to UUID.

A content key and its identifier are shared between all DRM systems, whereas the mechanisms used for key acquisition and content protection are largely DRM system specific. Different DASH adaptation sets are often protected by different content keys.

A **license** is a data structure in a DRM system specific format that contains one or more content keys and associates them with a policy that governs the usage of the content keys (e.g. expiration time). The encapsulated content keys are typically encrypted and only readable by the DRM system.

6. Client reference architecture for encrypted content playback§

Different software architectural components are involved in playback of encrypted content. The exact nature depends on the specific implementation. A high-level reference architecture is described here.

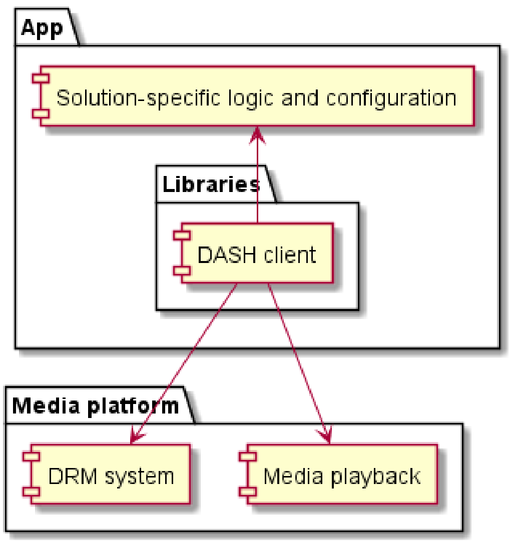


Figure 2 *Reference architecture for encrypted content playback.*

The **media platform** provides one or more APIs that allow the device's media playback and DRM capabilities to be used by a DASH client. The DASH client is typically a library included in an app. On some device types, the DASH client may be a part of the media platform.

This document assumes that the media platform exposes its encrypted content playback features via an API similar to W3C Encrypted Media Extensions (EME) [encrypted-media]. The technical nature of the API may be different but EME-equivalent functionality is expected.

The media platform often implements at least one DRM system. Additional DRM system implementations can be included as libraries in the app.

A **DRM system** is an implementation of content keys management. It is made of two main components: A license server for generating licenses and a DRM client for processing licenses and enforcing the associated policies. On some paltforms, the DRM client may handle the decryption of samples while on other platforms, decryption is handled by e.g. hardware elements the DRM client interacts with.

The guidelines in this document define recommended workflows and default behavior for a generic DASH client implementation that performs playback of encrypted content. In many scenarios, the default behavior is sufficient. When deviation from the default behavior is desired, **solution-specific logic and configuration** can be provided by the app. Extension points are explicitly defined in the workflows at points where solution-specific decisions are most appropriate.

7. Content encryption and DRM§

A DASH presentation MAY provide some or all adaptation sets in encrypted form, requiring the use of a DRM system to decrypt the content for playback. The duty of a DRM system is to prevent disclosure of the content key and misuse of the decrypted content (e.g. recording via screen capture software) and may be to decrypt content.

In a DASH presentation, every representation in an adaptation set SHALL be protected using the same content key (identified by the same `default_KID`).

This means that if representations use different content keys, they must be in different adaptation sets, even if they would otherwise (were they not encrypted) belong to the same adaptation set. A `urn:mpeg:dash:adaptation-set-switching:2016` supplemental property descriptor ([DASH] 5.3.3.5) SHALL be used to signal that such adaptation sets are suitable for switching.

Encrypted DASH content SHALL use either the `cenc` or the `cbcs` **protection scheme** defined in [CENC]. `cenc` and `cbcs` are two mutually exclusive protection schemes. DASH content encrypted according to the `cenc` protection scheme cannot be decrypted by a DRM system supporting only the `cbcs` protection scheme and vice versa.

Some DRM system implementations support both protection schemes. Even when this is the case, clients SHALL NOT concurrently consume encrypted content that uses different protection schemes.

Representations in the same adaptation set SHALL use the same protection scheme. Representations in different adaptation sets MAY use different protection schemes. If both protection schemes are used in the same period, all encrypted representations in that period SHALL be provided using both protection schemes. That is, the only permissible scenario for using both protection schemes together is to offer them as equal alternatives to target DASH clients with different capabilities.

Representations that contain the same media content using different protection schemes SHOULD use different content keys. This protects against some cryptographic attacks [MSPR-EncryptionModes].

7.1. Robustness§

DRM systems define rules that govern how they can be implemented. These rules can define different **robustness levels** which are typically used to differentiate implementations based on their resistance to attacks. The set of robustness levels, their names and the constraints that apply are all specific to each DRM system.

¶ EXAMPLE 3

A hypothetical DRM system might define the following robustness levels:

- High - All cryptographic operations are performed on a separate CPU not accessible to the device's primary operating system (often called a trusted execution environment). Decrypted data only exists in a memory region not accessible to the device's primary operating system (often called a secure media path).
- Medium - All cryptographic operations are performed on a separate CPU not accessible to the device's primary operating system. Decrypted data may be passed to the primary operating system's media platform APIs.
- Low - All operations are performed in software that can be inspected and modified by the user. Obfuscation must be used to protect against analysis.
- None - For development only. Implementation does not resist attacks.

Policy associated with content can require a DRM system implementation to conform to a certain robustness level, thereby ensuring that valuable content does not get presented on potentially vulnerable implementations. This policy can be enforced on different levels, depending on the DRM system:

1. A license server may refuse to provide content keys to implementations with unacceptable robustness levels.
2. The DRM system may refuse to use content keys whose license requires a higher robustness level than the implementation provides.

Multiple implementations of a DRM system may be available to a DASH client, potentially at different robustness levels. The DASH client must choose at media load time which DRM system implementation to use. However, the required robustness level may be different for different device types and is not expressed in the MPD. This decision is a matter of policy and is impossible for a DASH client to determine on its own. Therefore, solution-specific logic and configuration must inform the DASH client of the correct choice.

A DASH client SHALL enable solution-specific logic and configuration to specify the robustness level of the DRM system implementation to be used. Depending on which DRM system is used, this can be implemented by:

1. Changing the mapping of DRM system to key system in EME-based implementations (see § 7.2 W3C Encrypted Media Extensions).
2. Specifying a minimum robustness level during capability detection (see § 11.1 Capability detection).

7.2. W3C Encrypted Media Extensions§

Whereas the DRM signaling in DASH deals with DRM systems, EME deals with **key systems**. While similar in concept, they are not always the same thing. A single DRM system may be implemented on a single device by multiple different key systems, with different codec compatibility and functionality, potentially at different robustness levels.

¶ EXAMPLE 4

A device may implement the "ExampleDRM" DRM system as a number of key systems:

- The key system "ExampleDRMvariant1" may support playback of encrypted H.264 and H.265 content at up to 1080p resolution with "low" robustness level.
- The key system "ExampleDRMvariant2" may support playback of encrypted H.264 content at up to 4K resolution with "high" robustness level.
- The key system "ExampleDRMvariant3" may support playback of encrypted H.265 content at up to 4K resolution with "high" robustness level.

Even if multiple variants are available, a DASH client SHOULD map each DRM system to a single key system. The default key system SHOULD be the one the DASH client expects to offer greatest compatibility with content (potentially at a low robustness level). The DASH client SHOULD allow solution-specific logic and configuration to override the key system chosen by default (e.g. to force the use of a high-robustness variant).

8. Content protection constraints for CMAF§

The structure of content protection related information in the CMAF containers used by DASH is largely specified by [CMAF] and [CENC] (in particular section 8). This chapter outlines some additional requirements to ensure interoperable behavior of DASH clients and services.

> Note: This document uses the `cenc:` prefix to reference the XML namespace `urn:mpeg:cenc:2013` [CENC].

Initialization segments SHOULD NOT contain any `moov/pssh` box ([CMAF], section 7.4.3) and DASH clients MAY ignore such boxes when encountered. Instead, `pssh` boxes required for DRM system initialization are part of the DRM system configuration and SHOULD be placed in the MPD as `cenc:pssh` elements in DRM system specific `ContentProtection` descriptors.

> Note: Placing the `pssh` boxes in the MPD has become common for purposes of operational agility - it is often easier to update MPD files than rewrite initialization segments when the default DRM system configuration needs to be updated or when a new DRM system needs to be supported. Furthermore, in some scenarios the appropriate set of `pssh` boxes is not known when the initialization segment is created.

Protected content MAY be published without any `pssh` boxes in both the MPD and media segments. All DRM system configuration can be provided at runtime, including the `pssh` box data. See also § 9.3 Providing default DRM system configuration.

Media segments MAY contain `moof/pssh` boxes ([CMAF] 7.4.3) to provide updates to DRM system internal state (e.g. to supply new leaf keys in a key hierarchy). See § 9.2.1 default_KID in hierarchical/derived/variant key scenarios for an example.

> Note: These state updates may be transparent to a DASH client on some media platforms that intercept the `moof/pssh` boxes and supply them directly to the active DRM system; on other media platforms, the DASH client may need to extract and forward the `moof/pssh` boxes to the DRM system.

8.1. Content protection data in CMAF containers§

This chapter describes the structure of content protection data in CMAF containers used to provide encrypted content in a DASH presentation, summarizing the requirements defined by [ISOBMFF], [DASH], [CENC], [CMAF] and other parts of DASH-IF implementation guidelines.

DASH initialization segments contain:

- Zero or more `moov/pssh` "Protection System Specific Header" boxes ([CENC] 8.1) which provide DRM system initialization data in DRM system specific format. This usage is deprecated in favor of providing this data in the MPD. If both are present, the value in the MPD is used. See § 9.3 Providing default DRM system configuration.
- Exactly one `moov/trak/mdia/minf/stbl/stsd/sinf/schm` "Scheme Type" box ([ISOBMFF] 8.12.5) identifying the protection scheme. See [CENC] section 4.
- Exactly one `moov/trak/mdia/minf/stbl/stsd/sinf/schi/tenc` "Track Encryption" box ([CENC] 8.2) which contains default encryption parameters for samples. These default parameters may be overridden in media segments (see below).

DASH media segments are composed of a single CMAF fragment that contains:

- Exactly one `moof/traf/senc` "Sample Encryption" box ([CENC] 7.2) which stores initialization vectors (IVs) and, optionally, subsample encryption ranges for samples in the same CMAF fragment.
- Zero or one `moof/traf/saiz` "Sample Auxiliary Information Size" boxes ([ISOBMFF] 8.7.8) which references the sizes of the per-sample data stored in the `moof/traf/senc` box ([CMAF] 8.2.2 and [CENC] section 7).
    - Omitted if the parameters provided by the `senc` box are identical for all samples in the CMAF fragment.
- Zero or one `moof/traf/saio` "Sample Auxiliary Information Offset" boxes ([ISOBMFF] 8.7.9) which references the sizes of the per-sample data stored in the `moof/traf/senc` box ([CMAF] 8.2.2 and [CENC] section 7).
    - Omitted if the parameters provided by the `senc` box are identical for all samples in the CMAF fragment.
- Zero or more `moof/pssh` "Protection System Specific Header" boxes ([CENC] 8.1) which provide transparent updates to DRM system internal state. See § 9.4 Delivering updates to DRM system internal state.
- For each sample group, exactly one `moof/traf/sgpd` "Sample Group Description" box ([ISOBMFF] 8.9.3 and [CENC] section 6) which contains overrides for encryption parameters defined in the `tenc` box.
    - Omitted if no parameters are overridden.
- For each sample grouping type (see [ISOBMFF], typically one), exactly one `moof/traf/sbgp` "Sample to Group" box ([ISOBMFF] 8.9.2 and [CENC] section 6) which associates samples with sample groups.
    - Omitted if no parameters are overridden.

A key hierarchy is implemented by listing the `default_KID` in the `tenc` box of the initialization segment (identifying the root key) and then overriding the key identifier in the `sgpd` boxes of media segments (identifying the leaf keys that apply to each media segment). The `moof/pssh` box is used to deliver/unlock new leaf keys and may provide the associated license policy.

When using CMAF chunks for delivery, each CMAF fragment may be split into multiple CMAF chunks. If the CMAF fragment contained any `moof/pssh` boxes, copies of these boxes SHALL be present in each CMAF chunk that starts with an independent media sample.

> Note: While DASH only requires the presence of `moof/pssh` in the first CMAF chunk, the requirement is more extensive in the interest of HLS interoperability [HLS-LowLatency].

9. Encryption and DRM signaling in the MPD§

A DASH client needs to recognize encrypted content and activate a suitable DRM system, configuring it to decrypt content. The MPD informs a DASH client of the protection scheme used to protect content, identifies the content keys that are used and optionally provides the default DRM system configuration for a set of DRM systems.

The **DRM system configuration** is the complete data set required for a DASH client to activate a single DRM system and configure it to decrypt content using a single content key. **It is supplied by a combination of XML elements in the MPD and/or solution-specific logic and configuration**. The DRM system configuration often contains:

- DRM system initialization data in the form of a DRM system specific `pssh` box (as defined in [CENC]).
- DRM system initialization data in some other DRM system specific form (e.g. `keyids` JSON structure used by W3C Clear Key)
- The used protection scheme (`cenc` or `cbcs`)
- `default_KID` that identifies the content key
- License server URL
- Authorization service URL

The exact set of values required for successful DRM workflow execution depends on the requirements of the selected DRM system (e.g. what kind of initialization data it can accept) and the mechanism used for content key acquisition (e.g. the DASH-IF interoperable license request model). By default, a DASH client SHOULD assume that a DRM system accepts initialization data in `pssh` format and that the DASH-IF interoperable license request model is used for content key acquisition.

When configuring a DRM system to decrypt content using multiple content keys, a distinct DRM system configuration is associated with each content key. Concurrent use of multiple DRM systems is not an interoperable scenario.

> Note: In theory, it is possible for the DRM system initialization data to be the same for different content keys. In practice, the `default_KID` is often included in the initialization data so this is unlikely. Nevertheless, DASH clients cannot assume that using equal initialization data implies anything about equality of the DRM system configuration or the content key - the `default_KID` is the factor identifying the scope in which a single content key is to be used. See § 9.2 default_KID defines the scope of DRM system interactions.

The DRM system configuration MAY change over time, both due to MPD updates in live services and due to runtime changes in the solution-specific logic and configuration. A typical example of runtime changse would be using a unique license server URL for each license request.

9.1. Signaling presence of encrypted content§

The presence of a `ContentProtection` descriptor with `schemeIdUri="urn:mpeg:dash:mp4protection:2011"` on an adaptation set informs a DASH client that all representations in the adaptation set are encrypted in conformance to Common Encryption ([DASH] sections 5.8.4.1 and 5.8.5.2 and [CENC] section 11) and require a DRM system to provide access.

This descriptor is present for all encrypted content ([DASH] section 5.8.4.1). It SHALL be defined on the adaptation set level. The `value` attribute indicates the used protection scheme ([DASH] section 5.8.5.2). The `cenc:default_KID` attribute SHALL be present and have a value matching the `default_KID` in the `tenc` box.

¶EXAMPLE 5

Signaling an adaptation set encrypted using the `cbcs` scheme and with a content key identified by `34e5db32-8625-47cd-ba06-68fca0655a72`.

```
<ContentProtection
    schemeIdUri="urn:mpeg:dash:mp4protection:2011"
    value="cbcs"
    cenc:default_KID="34e5db32-8625-47cd-ba06-68fca0655a72" />
```

The `tenc` box stores `default_KID` as a 16-byte array. The byte order SHALL be identical in the binary structure and the string-form `default_KID`.

¶EXAMPLE 6

The following are two equivalent forms of representing the same `default_KID`:

- In string form: `00010203-0405-0607-0809-0a0b0c0d0e0f`
- In binary form: `0x00`, `0x01`, `0x02`, `0x03`, `0x04`, `0x05`, `0x06`, `0x07`, `0x08`, `0x09`, `0x0a`, `0x0b`, `0x0c`, `0x0d`, `0x0e`, `0x0f`

Some Windows-targeting software libraries implement "Microsoft style" UUID serialization that changes the order of bytes when transforming between string form and binary form. This is not appropriate when serializing/deserializing `default_KID` values. Linux-based tooling typically does not change the byte order.

9.2. default_KID defines the scope of DRM system interactions§

A DASH client interacts with one or more DRM systems during playback in order to control the decryption of content. Some of the most important interactions are:

- Activating a DRM system to play back content protected with a specific set of content keys.
- Communicating with the DRM system to make content keys available for use, executing license requests as needed.

The scope of each of these interactions is defined by the `default_KID`. Each distinct `default_KID` identifies exactly one content key. The impact of this is further outlined in § 11 DRM workflows in DASH clients.

When activating a DRM system, a DASH client SHALL determine the required set of content keys based on the `default_KID` values of adaptation sets selected for playback. This set of content keys is used to activate the DRM system, after which zero or more of the content keys from this set are available for playback.

Clients SHALL provide all `default_KIDs` of the selected adaptation sets to the DRM system during activation and SHALL NOT assume that activating a DRM system with one content key will implicitly enable the use of any other content key.

> Note: An occasionally encountered anti-pattern is to activate a DRM system for only key X but to configure the license server to always provide both keys X and Y when key X is requested. This is not inteoperable behavior.

The DASH client and/or DRM system MAY batch license requests for different `default_KIDs` (and the respective responses) into a single transaction (for example, to reduce the chattiness of license acquisition traffic).

> Note: This optimization might require support from platform APIs and/or DRM system specific logic from the DASH client, as a batching mechanism is not yet a standard part of DRM related platform APIs.

9.2.1. default_KID in hierarchical/derived/variant key scenarios§

While it is common that `default_KID` identifies the actual content key used for encryption, a DRM system MAY make use of other keys in addition to the one signalled by the `default_KID` value but this SHALL be transparent to the client with only the `default_KID` being used in interactions between the DASH client and the DRM system. See § 13 Controlling access rights with a key hierarchy.

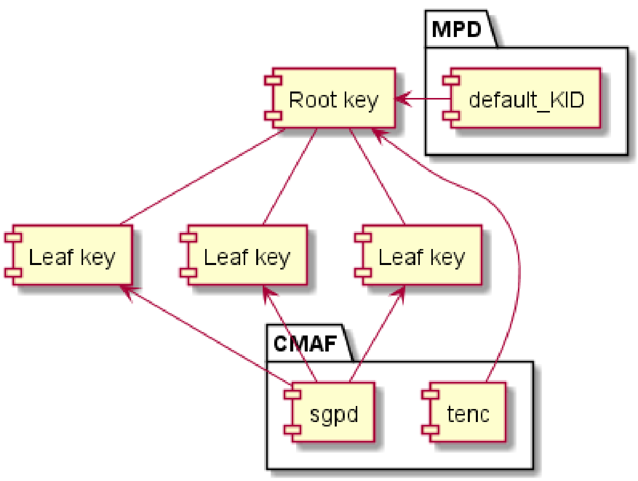


Figure 3 *In a hierarchical key scenario, `default_KID` references the root key and only the sample group descriptions reference the leaf keys.*

In a hierarchical key scenario, `default_KID` identifies the root key, not the leaf key used to encrypt media samples, and the handling of leaf keys is not exposed to a DASH client. As far as a DASH client knows, there is always only one content key identified by `default_KID`.

This logic applies to all scenarios that make use of additional keys, regardless of whether they are based on the key hierarchy, key derivation or variant key ([iso23001-12]) concepts.

9.3. Providing default DRM system configuration§

A DASH service SHOULD supply a default DRM system configuration in the MPD for all supported DRM systems in all encrypted adaptation sets. This enables playback without the need for DASH client customization or additional client-side configuration. DRM system configuration MAY also be supplied by solution-specific logic and configuration, replacing or enhancing the defaults provided in the MPD.

Any number of `ContentProtection` descriptors ([DASH] section 5.8.4.1) MAY be present in the MPD to provide DRM system configuration. These descriptors SHALL be defined on the adaptation set level. The contents MAY be ignored by the DASH client if overridden by solution-specific logic and configuration - the DRM system configuration in the MPD simply provides default values known at content authoring time.

A `ContentProtection` descriptor providing a default DRM system configuration SHALL use `schemeIdUri="urn:uuid:<systemid>"` to identify the DRM system, with the `<systemid>` matching a value in the DASH-IF system-specific identifier registry. The `value` attribute of the `ContentProtection` descriptor SHOULD contain the DRM system name and version number in a human readable form (for diagnostic purposes).

> Note: W3C defines the Clear Key mechanism ([encrypted-media] section 9.1), which is a "dummy" DRM system implementation intended for client and platform development/testing purposes. **Understand that Clear Key does not fulfill the content protection and content key protection duties ordinarily expected from a DRM system.** For more guidelines on Clear Key usage, see § 14 Use of W3C Clear Key with DASH.

Each DRM system specific `ContentProtection` descriptor can contain a mix of XML elements and attributes defined by [CENC], the DRM system author, DASH-IF or any other party.

For DRM systems initialized by supplying `pssh` boxes, the `cenc:pssh` element SHOULD be present under the `ContentProtection` descriptor if the value is known at MPD authoring time. The base64 encoded contents of the element shall be equivalent to a complete `pssh` box including its length and header fields ([CENC] section 11.3.3). See also § 8 Content protection constraints for CMAF.

DRM systems generally use the concept of license requests as the mechanism for obtaining content keys and associated usage constraints (see § 11.7 Performing license requests). For DRM systems that use this concept, one or more `dashif:laurl` elements SHOULD be present under the `ContentProtection` descriptor, with the value of the element being the URL to send license requests to. This URL MAY contain content identifiers.

Multiple mechanisms have historically been used to provide the license server URL in the MPD (e.g. embedding in the `cenc:pssh` data or passing by deprecated DRM system specific DASH-IF `Laurl` elements). A DASH client SHALL prefer `dashif:laurl` if multiple data sources for the URL are present in the MPD.

For DRM systems that require proof of authorization to be attached to the license request in a manner conforming to § 10 DASH-IF interoperable license request model, one or more `dashif:authzurl` elements SHOULD be present under the `ContentProtection` descriptor, containing the default URL to send authorization requests to (see § 11.7 Performing license requests).

Multiple `dashif:laurl` or `dashif:authzurl` elements under the same `ContentProtection` descriptor define sets of equivalent alternatives for the DASH client to choose from. A DASH client SHOULD select a random item from the set every time the value of such an element is used.

ISSUE 1 ¶The above paragraph on URL handling should be generalized to all sets of alternative URLs but there does not seem to be a suitable chapter in v4.3 If such a chapter is created in v5, we could replace the above paragraph with a reference to the general URL handling guidelines.

¶EXAMPLE 7

A `ContentProtection` descriptor that provides default DRM system configuration for a fictional DRM system.

```
<ContentProtection
  schemeIdUri="urn:uuid:d0ee2730-09b5-459f-8452-200e52b37567"
  value="FirstDRM 2.0">
  <cenc:pssh>YmFzZTY0IGVuY29kZWQgY29udGVudHMgb2YgkXBzc2iSIGJveCB3aXRoIHRoaXMgU3lzdGVtSUQ=</ce
nc:pssh>
  <dashif:authzurl>https://example.com/tenants/5341/authorize</dashif:authzurl>
  <dashif:laurl>https://example.com/AcquireLicense</dashif:laurl>
</ContentProtection>
```

The presence of a DRM system specific `ContentProtection` descriptor is not required in order to activate the DRM system; these descriptors are used merely to provide the default DRM system configuration. Empty `ContentProtection` descriptors SHOULD NOT be present in an MPD and MAY be ignored by DASH clients.

Because `default_KID` determines the scope of DRM system interactions, the contents of DRM system specific `ContentProtection` descriptors with the same `schemeIdUri` SHALL be identical in all adaptation sets with the same `default_KID`. This means that a DRM system will treat equally all adaptation sets that use the same content key.

Note: If you wish to change the default DRM system configuration associated with a content key, you must update all the instances where the data is present in the MPD. For live services, this can mean updating the data in multiple periods.

To maintain the `default_KID` association, a DASH client that exposes APIs/callbacks to business logic for the purpose of controlling DRM interactions and/or supplying data for DRM system configuration SHALL NOT allow these APIs to associate multiple DRM system configurations for the same DRM system with the same `default_KID`. Conversely, DASH client APIs SHOULD allow business logic to provide different DRM system configurations for the same DRM system for use with different `default_KIDs`.

9.4. Delivering updates to DRM system internal state§

Some DRM systems support live updates to DRM system internal state (e.g. to deliver new leaf keys in a key hierarchy). These updates SHALL NOT be present in the MPD and SHALL be delivered by `moof/pssh` boxes in media segments.

10. DASH-IF interoperable license request model§

The interactions involved in acquiring licenses and content keys in DRM workflows have historically been proprietary, requiring a DASH client to be customized in order to achieve compatibility with specific DRM systems or license server implementations. This chapter defines an interoperable model to encourage the creation of solutions that do not require custom code in the DASH client in order to play back encrypted content. Use of this model is optional but recommended.

Any conformance statements in this chapter apply to clients and services that opt in to using this model (e.g. a "SHALL" statement means "SHALL, if using this model," and has no effect on implementations that choose to use proprietary mechanisms for license acquisition). The authorization service and license server are considered part of the DASH service.

In performing license acquisition, a DASH client needs to:

1. Be able to prove that the user and device have the right to use the requested content keys.
2. Handle errors in a manner agnostic to the specific DRM system and license server being used.

This license request model defines a mechanism for achieving both goals. This results in the following interoperability benefits:

- DASH clients can execute DRM workflows without solution-specific logic and configuration.
- Custom code specific to a license server implementation is limited to backend business logic.

These benefits increase in value with the size of the solution, as they reduce the development cost required to offer playback of encrypted content on a wide range of DRM-capable client platforms using different DRM systems, with licenses potentially served by different license server implementations.

10.1. Proof of authorization§

An **authorization token** is a JSON Web Token used to prove to a license server that the caller has the right to use one or more content keys under certain conditions. Attaching this proof of authorization to a license request is optional, allowing for architectures where a "license proxy" performs authorization checks in a manner transparent to the DASH client.

The basic structural requirements for authorization tokens are defined in [jwt] and [jws]. This document adds some additional constraints to ensure interoperability. Beyond that, the license server implementation is what defines the contents of the authorization token (the set of claims it contains), as the data needs to express implementation-specific license server business logic parameters that cannot be generalized.

> Note: An authorization token is divided into a header and body. The distinction between the two is effectively irrelevant and merely an artifact of the JWT specification. License servers may use existing fields and define new fields in both the header and the body.

Implementations SHALL process claims listed in [jwt] section 4.1 "Registered Claim Names" when they are present (e.g. `exp` "Expiration Time" and `nbf` "Not Before"). The `typ` header parameter ([jwt] section 5.1) SHOULD NOT be present. The `alg` header parameter defined in [jws] SHALL be present.

¶ EXAMPLE 8

JWT headers, specifying digital signature algorithm and expiration time (general purpose fields):

```
{
    "alg": "HS256",
    "exp": "1516239022"
}
```

JWT body with list of authorized content key IDs (an example field that could be defined by a license server):

```
{
    "authorized_kids": [
        "1611f0c8-487c-44d4-9b19-82e5a6d55084",
        "db2dae97-6b41-4e99-8210-493503d5681b"
    ]
}
```

The above data sets are serialized and digitally signed to arrive at the final form of the authorization token:

eyJhbGciOiJIUzI1NiIsImV4cCI6IjE1MTYyMzkwMjIifQ.eyJhdXRob3JpemVkX2tpZHMiOlsiMTYxMWYwYzgtNDg3Yy00NGQ0LTliMTktODJlNWE2ZDU1MDg0IiwiZGIyZGFlOTctNmI0MS00ZTk5LTgyMTAtNDkzNTAzZDU2ODFiIl19.tBvW6XVPHBRp1JEwItsVnbHwIqoqnQAVQfTV9PGMkIU

Authorization tokens are issued by an authorization service, which is part of a solution’s business logic. The authorization service has access to the project-specific context that it needs to make its decisions (e.g. the active session, user identification and database of purchases/entitlements). A single authorization service can be used to issue authorization tokens for multiple license servers, simplifying architecture in solutions where multiple license server vendors are used.

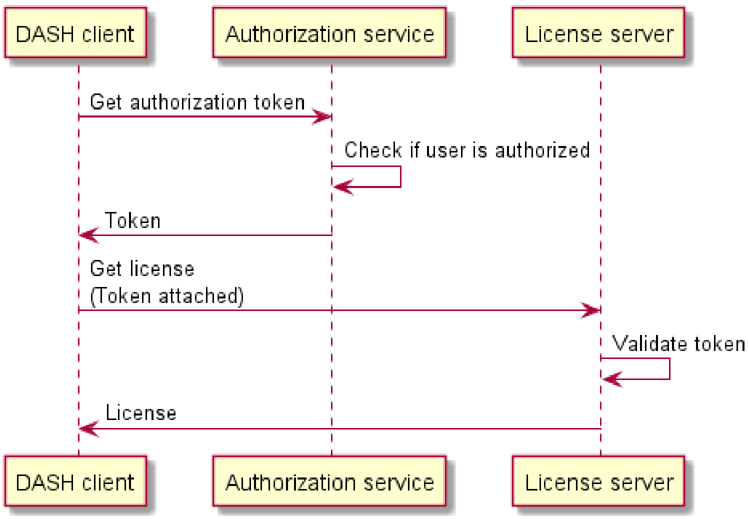


Figure 4 *Role of the authorization service in DRM workflow related communication.*

An authorization service SHALL digitally sign any issued authorization token with an algorithm from the "HMAC with SHA-2 Functions" or "Digital Signature with ECDSA" sets in [jwt]. The HS256 algorithm is recommended as a highly compatible default, as it is a required part of every JWT implementation. License server implementations SHALL validate the digital signature and reject tokens with invalid signatures or tokens using signature algorithms other than those referenced here. The license server MAY further constrain the set of allowed signature algorithms.

Successful signature verification requires that keys/certificates be distributed and trust relationships be established between the signing parties and the validating parties. The specific mechanisms for this are implementation-specific and out of scope of this document.

10.1.1. Obtaining authorization tokens§

To obtain an authorization token, a DASH client needs to know the URL of the authorization service. DASH services SHOULD specify the authorization service URL in the MPD using the `dashif:authzurl` element (see § 9.3 Providing default DRM system configuration).

If no authorization service URL is provided by the MPD nor made available at runtime, a DASH client SHALL NOT attach an authorization token to a license request. Absence of this URL implies that authorization operations are performed in a manner transparent to the DASH client (see § 10.3 Possible deployment architectures).

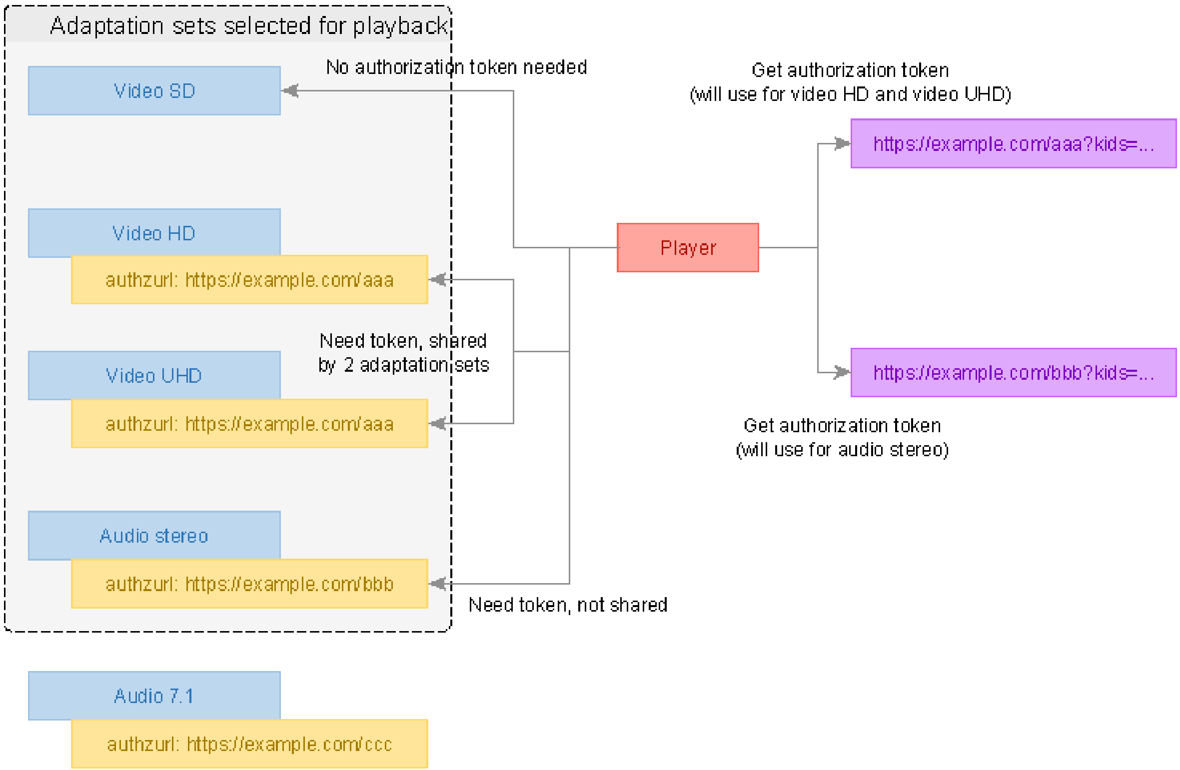


Figure 5 *Authorization tokens are requested from all authorization services referenced by the selected adaptation sets.*

DASH clients will use zero or more authorization tokens depending on the number of authorization service URLs defined for the set of content keys in use. One authorization token is requested from each distinct authorization service URL. The authorization service URL is specified individually for each DRM system and content key (i.e. it is part of the DRM system configuration). Services SHOULD use a single authorization token covering all content keys and DRM systems but MAY divide the scope of authorization tokens if appropriate (e.g. different DRM systems might use different license server vendors that use mutually incompatible authorization token formats).

> Note: Path or query string parameters in the authorization service URL can be used to differentiate between license server implementations (and their respective authorization token formats).

DASH clients SHOULD cache and reuse authorization tokens up to the moment specified in the token's `exp` "Expiration Time" claim (defaulting to "never expires"). DASH clients SHALL discard the authorization token and request a new one if the license server indicates that the authorization token was rejected (for any reason), even if the "Expiration Time" claim is not present or the expiration time is in the future (see § 10.2 Problem signaling and handling).

Before requesting an authorization token, a DASH client SHALL take the authorization service URL and add or replace the `kids` query string parameter containing a comma-separated list in ascending alphanumeric order of `default_KID` values obtained from the MPD. This list SHALL contain every `default_KID` for which proof of authorization is requested from this authorization service (i.e. every distinct `default_KID` for which the same set of URLs was specified using `dashif:authzurl` elements).

To request an authorization token, a DASH client SHALL make an HTTP GET request to this modified URL, attaching to the request any standard contextual information used by the underlying platform and allowed by active security policy (e.g. HTTP cookies). This data can be used by the authorization service to identify the user and device and assess their access rights.

> Note: For DASH clients operating on the web platform, effective use of the authorization service may require the authorization service to exist on the same origin as the website hosting the DASH client in order to share the session cookies.

If the HTTP response status code indicates a successful result and `Content-Type: text/plain`, the HTTP response body is the authorization token.

¶ EXAMPLE 9

Consider an MPD that specifies the authorization service URL `https://example.com/Authorize` for the content keys with `default_KID` values `1611f0c8-487c-44d4-9b19-82e5a6d55084` and `db2dae97-6b41-4e99-8210-493503d5681b`.

The generated URL would then be `https://example.com/Authorize?kids=1611f0c8-487c-44d4-9b19-82e5a6d55084,db2dae97-6b41-4e99-8210-493503d5681b` to which a DASH client would make a GET request:

```
GET /Authorize?kids=1611f0c8-487c-44d4-9b19-82e5a6d55084,db2dae97-6b41-4e99-8210-493503d5681b
 HTTP/1.1
Host: example.com
```

Assuming authorization checks pass, the authorization service would return the authorization token in the HTTP response body:

```
HTTP/1.1 200 OK
Content-Type: text/plain

eyJhbGciOiJIUzI1NiIsImV4cCI6IjE1MTYyMzkwMjIifQ.eyJhdXRob3JpemVkX2tpZHMiOlsiMTYxMWYwYzgtNDg3Yy
00NGQ0LTliMTktODJlNWE2ZDU1MDg0IiwiZGIyZGFlOTctNmI0MS00ZTk5LTgyMTAtNDkzNTAzZDU2ODFiIl19.tBvW6X
VPHBRp1JEwItsVnbHwIqoqnQAVQfTV9PGMkIU
```

If the HTTP response status code indicates a failure, a DASH client needs to examine the response to determine the cause of the failure and handle it appropriately (see § 10.2 Problem signaling and handling). DASH clients SHOULD NOT treat every failed authorization token request as a fatal error - if multiple authorization tokens are used to authorize access to different content keys, it may be that some of them fail but others succeed, potentially still enabling a successful playback experience. The examination of whether playback can successfully proceed SHOULD be performed only once all license requests have been completed and the final set of available content keys is known. See also § 11.4 Handling unavailability of content keys.

DASH clients SHALL follow HTTP redirects signaled by the authorization service.

10.1.2. Issuing authorization tokens§

The mechanism of performing authorization checks is implementation-specific. Common approaches might be to identify the user from a session cookie, query the entitlements/purchases database to identify what rights are assigned to the user and then assemble a suitable authorization token, taking into account the license policy configuration that applies to the content keys being requested. The DRM system may be involved in order to ensure secure authentication of the device.

The structure of the authorization tokens is unconstrained beyond the basic requirements defined in § 10.1 Proof of authorization. Authorization services need to issue tokens that match the expectations of license servers that will be using these tokens. If multiple different license server implementations are served by the same authorization service, the path or query string parameters in the authorization service URL allow the service to identify which output format to use.

¶ EXAMPLE 10

Example authorization token matching the requirements of a hypothetical license server.

JWT headers, specifying digital signature algorithm and expiration time:

```
{
    "alg": "HS256",
    "exp": "1516239022"
}
```

JWT body with list of authorized content key IDs (an example field that could be defined by a license server):

```
{
    "authorized_kids": [
        "1611f0c8-487c-44d4-9b19-82e5a6d55084",
        "db2dae97-6b41-4e99-8210-493503d5681b"
    ]
}
```

Serialized and digitally signed:

`eyJhbGciOiJIUzI1NiIsImV4cCI6IjE1MTYyMzkwMjIifQ.eyJhdXRob3JpemVkX2tpZHMiOlsiMTYxMWYwYzgtNDg3Yy00NGQ0LTliMTktODJlNWE2ZDU1MDg0IiwiZGIyZGFlOTctNmI0MS00ZTk5LTgyMTAtNDkzNTAzZDU2ODFiIl19.tBvW6XVPHBRp1JEwItsVnbHwIqoqnQAVQfTV9PGMkIU`

An authorization service SHALL NOT issue authorization tokens that authorize the use of content keys that are not in the set of requested content keys (as defined in the request's `kids` query string parameter). An authorization service MAY issue authorization tokens that authorize the use of only a subset of the requested content keys, provided that at least one content key is authorized. If no content keys are authorized for use, an authorization service SHALL signal a failure.

Note: During license issuance, the license server may further constrain the set of available content keys (e.g. as a result of examining the robustness level of the DRM system implementation requesting the license). See § 11.4 Handling unavailability of content keys.

Authorization tokens SHALL be returned by an authorization service using JWS Compact Serialization [jws] (the `aaa.bbb.ccc` format). The serialized form of an authorization token SHOULD NOT exceed 5000 characters to ensure that a license server does not reject a license request carrying the token due to excessive HTTP header size.

10.1.3. Embedding secrets in authorization tokens§

Custom data fields on the authorization token MAY be encrypted to protect secrets within, with the data format, encryption method and key management scheme defined by the license server. These guidelines do not define any general recommendation for passing encrypted data through authorization tokens.

Note: The JSON Web Token [jwt] data format used for authorization tokens is based on JSON Web Signature [jws], which only supports signing without encryption. While JSON Web Encryption [jwe] does define an encrypted data format, this is a completely separate data format from JSON Web Token.

¶ EXAMPLE 11

Example JWT body containing a secret (client IP address, encrypted for privacy):

```
{
    "authorized_kids": [
        "1611f0c8-487c-44d4-9b19-82e5a6d55084",
        "db2dae97-6b41-4e99-8210-493503d5681b"
    ],
    "client_ip_encrypted": "460e39f04d204c6233757feba31e8c1828019179dd651c55b14ab6c78e745148"
}
```

This example uses a custom data format for the encrypted data.

10.1.4. Attaching authorization tokens to license requests§

Authorization tokens are attached to license requests using the `Authorization` HTTP request header, signaling the `Bearer` authorization type.

¶ EXAMPLE 12

HTTP request to a hypothetical license server, carrying an authorization token.

```
POST /AcquireLicense HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsImV4cCI6IjE1MTYyMzkwMjIifQ.eyJhdXRob3JpemVkX2tpZHMi
OlsiMTYxMWYwYzgtNDg3Yy00NGQ0LTliMTktODJlNWE2ZDU1MDg0IiwiZGIyZGFlOTctNmI0MS00ZTk5LTgyMTAtNDkzN
TAzZDU2ODFiIl19.tBvW6XVPHBRp1JEwItsVnbHwIqoqnQAVQfTV9PGMkIU

(opaque license request blob from DRM system goes here)
```

The same authorization token MAY be used with multiple license requests but one license request SHALL only carry one authorization token, even if the license request is for multiple content keys. A DASH client SHALL NOT use content key batching features offered by the platform APIs to combine requests for content keys that require the use of separate authorization tokens.

A DASH client SHALL NOT make license requests for content keys that are configured as requiring an authorization token but for which the DASH client has failed to acquire an authorization token.

> Note: A content key requires an authorization token if there is at least one `dashif:authzurl` in the MPD or if this element is added by solution-specific logic and configuration.

10.2. Problem signaling and handling§

Authorization services and license servers SHOULD indicate an inability to satisfy a request by returning an HTTP response that:

1. Signals a suitable status code (4xx or 5xx).
2. Has a `Content-Type` of `application/problem+json`.
3. Contains a HTTP response body conforming to [rfc7807].

¶EXAMPLE 13
HTTP response from an authorization service, indicating a rejected authorization token request because the requested content is not a part of the user's subscriptions.

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json

{
    "type": "https://dashif.org/drm-problems/not-authorized",
    "title": "Not authorized",
    "detail": "Your active service plan does not include the channel 'EurasiaSport'.",
    "href": "https://example.com/view-available-subscriptions?channel=EurasiaSport",
    "hrefTitle": "Available subscriptions"
}
```

A problem record SHALL contain a short human-readable description of the problem in the `title` field and SHOULD contain a human-readable description, designed to help the reader solve the problem, in the `detail` field.

> Note: The `detail` field is intended to be displayed to users of a DASH client, not to developers. The description should be helpful to the user whose device the DASH client is running on.

During DRM system activation, it is possible that multiple failures occur. DASH clients SHOULD be capable of displaying a list of error messages to the end-user and SHOULD deduplicate multiple records with the same `type` (e.g. if an authorization token expires, this expiration may cause failures when requesting five content keys but should result in at most one error message being displayed).

> Note: Merely the fact that a problem record was returned does not mean that it needs to be presented to the user or acted upon in other ways. The user may still experience successful playback in the presence of some failed requests. See § 11.4 Handling unavailability of content keys.

This chapter defines a set of standard problem types that SHOULD be used to indicate the nature of the failure. Implementations MAY extend this set with further problem types if the nature of the failure does not fit into the existing types.

> ISSUE 2 ¶Let's come up with a good set of useful problem types we can define here, to reduce the set of problem types that must be defined in solution-specific scope.

10.2.1. Problem type: not authorized to access content§

Type: `https://dashif.org/drm-problems/not-authorized`

Title: Not authorized

HTTP status code: 403

Used by: authorization service

This problem record SHOULD be returned by an authorization service if the user is not authorized to access the requested content keys. The `detail` field SHOULD explain why this is so (e.g. their subscription has expired, the requested content keys are for a movie not in their list of purchases, the content is not available in their geographic region).

The authorization service MAY supply a `href` (string) field on the problem record, containing a URL using which the user can correct the problem (e.g. purchase a missing subscription). If the `href` field is present, a `hrefTitle` (string) field SHALL also be present, containing a title suitable for a hyperlink or button (e.g. "Subscribe"). DASH clients MAY expose this URL and title in their user interface to enable the user to find a quick solution to the problem.

10.2.2. Problem type: insufficient proof of authorization§

Type: `https://dashif.org/drm-problems/insufficient-proof-of-authorization`

Title: Not authorized

HTTP status code: 403

Used by: license server

This problem record SHOULD be returned by a license server if the proof of authorization (if any) attached to a license request is not sufficient to authorize the use of any of the requested content keys. The `detail` field SHOULD explain what exactly was the expectation the caller failed to satisfy (e.g. no token provided, token has expired, token is for disabled tenant).

> Note: If the authorization token authorizes only a subset of requested keys, a license server does not signal a problem and simply returns only the authorized subset of content keys.

When encountering this problem, a DASH client SHOULD discard whatever authorization token was used, acquire a new authorization token and retry the license request. If no authorization service URL is available, this indicates a DASH service or client misconfiguration (as clearly, an authorization token was expected) and the problem SHOULD be escalated for operator attention.

10.3. Possible deployment architectures§

The interoperable license request model is designed to allow for the use of different deployment architectures in common use today, including those where authorization duties are offloaded to a "license proxy". This chapter outlines some of the possible architectures and how interoperable DASH clients support them.

The baseline architecture assumes that a separate authorization service exists, implementing the logic required to determine which users have the rights to access which content.

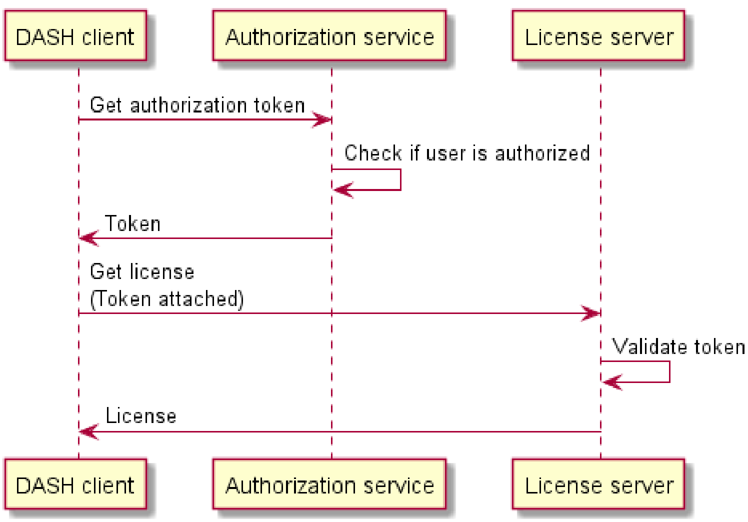


Figure 6 *The baseline architecture with an authorization service directly exposed to the DASH client.*

While the baseline architecture offers several advantages, in some cases it may be desirable to have the authorization checks be transparent to the DASH client. This need may be driven by license server implementation limitations or by other system architecture decisions.

A common implementation for transparent authorization is to use a "license proxy", which acts as a license server but instead forwards the license request after authorization checks have passed. Alternatively, the license server itself may perform the authorization checks.

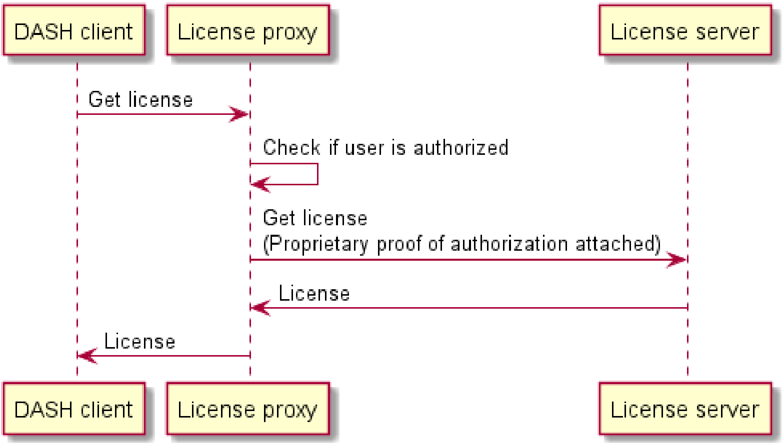


Figure 7 *A transparent authorization architecture performs the authorization checks at the license server, which is often hidden behind a proxy (indistinguishable from a license server to the DASH client).*

The two architectures can be mixed, with some DRM systems performing the authorization operations in the license server (or a "license proxy") and others using the authorization service directly. This may be relevant when integrating license servers from different vendors into the same solution.

A DASH client will attempt to contact an authorization service if an authorization service URL is provided either in the MPD or by solution-specific logic and configuration. If no such URL is provided, it will assume that all authorization checks (if any are required) are performed by the license server (in reality, often a license proxy) and will not attach any proof of authorization.

10.4. Passing a content ID to services§

The concept of a content ID is sometimes used to identify groups of content keys based on solution-specific associations. The DRM workflows described by this document do not require this concept to be used but do support it if the solution architecture demands it.

In order to make use of a content ID in DRM workflows, the content ID SHOULD be embedded into authorization service URLs and/or license server URLs (depending on which components are used and require the use of the content ID). This may be done either directly at MPD authoring time (if the URLs and content ID are known at such time) or by solution-specific logic and configuration at runtime.

Having embedded the content ID in the URL, all DRM workflows continue to operate the same as they normally would, except now they also include knowledge of the content ID in each request to the authorization service and/or license server. The content ID is an addition to the license request workflows and does not replace any existing data.

Embedding a content ID allows the service handling the request to use the content ID in its business logic. However, the presence of a content ID in the URL does not invalidate any requirements related to the processing of the `default_KID` values of content keys. For example, an authorization service must still constrain the set of authorized content keys to a subset of the keys listed in the `kids` parameter (§ 10.1.2 Issuing authorization tokens).

No generic URL template for embedding the content ID is defined, as the content ID is always a proprietary concept. Recommended options include:

- Query string parameters: `https://example.com/tenants/5341/authorize?contentId=movie865343651`

- Path segments: `https://example.com/moviecatalog-license-api/movie865343651/AcquireLicense`

EXAMPLE 14

DRM system configuration with the content ID embedded in the authorization service and license server URLs. Each service may use a different implementation-defined URL structure for carrying the content ID.

```
<ContentProtection
  schemeIdUri="urn:uuid:d0ee2730-09b5-459f-8452-200e52b37567"
  value="AcmeDRM 2.0">
  <cenc:pssh>YmFzZTY0IGVuY29kZWQgY29udGVudHMgb2YgkXBzc2iSIGJveCB3aXRoIHRoaXMgU3lzdGVtSUQ=</cenc:pssh>
  <dashif:authzurl>https://example.com/tenants/5341/authorize?contentId=movie865343651</dashif:authzurl>
  <dashif:laurl>https://example.com/moviecatalog-license-api/movie865343651/AcquireLicense</dashif:laurl>
</ContentProtection>
```

The content ID SHOULD NOT be embedded in DRM system specific data structures such as `pssh` boxes, as logic that depends on DRM system specific data structures is not interoperable and often leads to increased development and maintenance costs.

11. DRM workflows in DASH clients§

To present encrypted content a DASH client needs to:

1. Select a DRM system that is capable of decrypting the content.
    - During selection, the set of desired DRM system capabilities and the supported capabilities is examined to identify suitable candidate systems.
2. Activate the selected DRM system and configure it to decrypt content.
    - During activation, acquire any missing content keys and the licenses that govern their use.
3. Monitor for changes in the availability of content keys and in the content protection attributes of the media stream and take required action to ensure that playback can continue (e.g. live services may periodically change the content keys, requiring new licenses to be obtained, or existing licenses can simply expire and need renewal).

This chapter defines the recommended DASH client workflows for interacting with DRM systems in these aspects.

11.1. Capability detection§

A DRM system implemented by a client platform may only support playback of encrypted content that matches certain parameters (e.g. codec type and level). A DASH client needs to detect what capabilities each DRM system has in order to understand what adaptation sets can be presented and to make an informed choice when multiple DRM systems can be used.

¶ EXAMPLE 15

A typical DRM system might offer the following set of capabilities:

- Playback of H.264 High profile up to level 4.0 at "low" robustness
- Playback of H.264 High profile level 4.1 at "low" robustness
- Playback of H.265 Main 10 profile up to level 5.2 at "low" robustness
- Playback of AAC at "low" robustness
- Unique user identification
- Session persistence

A typical media platform API such as EME [encrypted-media] will require the DASH client to query the platform by supplying a desired capability set. The media platform will inspect the desired capabilities, possibly displaying a permissions prompt to the user (if sensitive capabilities such as unique user identification are requested), after which it will return a supported capability set that indicates which of the desired capabilities are available.

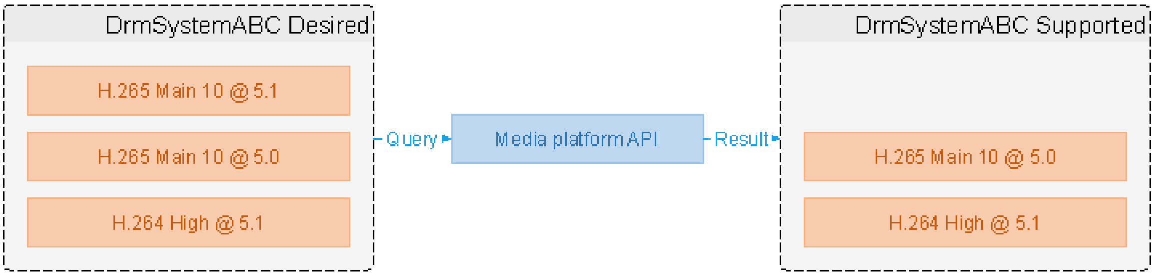


Figure 8 *The DASH client presents a set of desired capabilities for each DRM system and receives a response with the supported subset.*

The exact set of capabilities that can be used and the data format used to express them in capability detection APIs are defined by the media platform API. A DASH client is expected to have a full understanding of the potentially offered capabilities and how they map to parameters in the MPD. Some capabilities may have no relation to the MPD and whether they are required depends entirely on the DASH client or solution-specific logic and configuration.

To detect the set of supported capabilities, a DASH client must first determine the **required capability set** for each adaptation set. This is the set of capabilities required to present all the content in a single adaptation set and can be determined based on the following:

1. Content characteristics defined in the MPD (e.g. codecs strings of the representations and the used protection scheme).
2. Solution-specific logic and configuration (e.g. what robustness level is required).

Querying for the support of different protection schemes is currently not possible via the capability detection API of Encrypted Media Extensions [encrypted-media]. To determine the supported protection schemes, a DASH client must assume what the CDM supports. A bug is open on W3C EME and a pull request exists for the ISOBMFF file format bytestream. In future versions of EME, this may become possible.

Some of the capabilities (e.g. required robustness level) are DRM system specific. The required capability set contains the values for all DRM systems.

During DRM system selection, the required capability set of each adaptation set is compared with the supported capability set of a DRM system. As a result of this, each candidate DRM system is associated with zero or more adaptation sets that can be successfully presented using that DRM system.

It is possible that multiple DRM systems have the capabilities required to present some or all of the adaptation sets. When multiple candidates exist, the DASH client SHOULD enable solution-specific logic and configuration to make the final decision.

> Note: Some sensible default behavior can be implemented in a generic way (e.g. the DRM system should be able to enable playback of both audio and video if both media types are present in the MPD). Still, there exist scenarios where the choices seem equivalent to the DASH client and an arbitrary choice needs to be made.

The workflows defined in this document contain the necessary extension points to allow DASH clients to exhibit sensible default behavior and enable solution-specific logic and configuration to drive the choices in an optimal direction.

11.2. Selecting the DRM system§

The MPD describes the protection scheme used to encrypt content, with the `default_KID` values identifying the content keys required for playback, and optionally provides the default DRM system configuration for one or more DRM systems via `ContentProtection` descriptors. It also identifies the codecs used by each representation, enabling a DASH client to determine the set of required DRM system capabilities.

Neither an initialization segment nor a media segment is required to select a DRM system. The MPD is the only component of the presentation used for DRM system selection.

EXAMPLE 16

An adaptation set encrypted with a key identified by `34e5db32-8625-47cd-ba06-68fca0655a72` using the `cenc` protection scheme.

```
<AdaptationSet>
    <ContentProtection
        schemeIdUri="urn:mpeg:dash:mp4protection:2011"
        value="cenc"
        cenc:default_KID="34e5db32-8625-47cd-ba06-68fca0655a72" />
    <ContentProtection
        schemeIdUri="urn:uuid:d0ee2730-09b5-459f-8452-200e52b37567"
        value="FirstDrm 2.0">
        <cenc:pssh>YmFzZTY0IGVuY29kZWQgY29udGVudHMgb2YgkXBzc2iSIGJveCB3aXRoIHRoaXMgU3lzdGVtSU
Q=</cenc:pssh>
        <dashif:authzurl>https://example.com/tenants/5341/authorize?mode=firstDRM</dashif:aut
hzurl>
        <dashif:authzurl>https://alternative.example.com/tenants/5341/authorize?mode=firstDRM
</dashif:authzurl>
        <dashif:laurl>https://example.com/AcquireLicense</dashif:laurl>
        <dashif:laurl>https://alternative.example.com/AcquireLicense</dashif:laurl>
    </ContentProtection>
    <ContentProtection
        schemeIdUri="urn:uuid:eb3841cf-d7e4-4ec4-a3c5-a8b7f9f4f55b"
        value="SecondDrm 8.0">
        <cenc:pssh>ZXQgb2YgcGxheWFibGUgYWRhcHRhdGlvbiBzZXRzIG1heSBjaGFuZ2Ugb3ZlciB0aW1lIChlLm
cuIGR1ZSB0byBsaWNlbnNlIGV4cGlyYXRpb24gb3IgZHVl</cenc:pssh>
        <dashif:authzurl>https://example.com/tenants/5341/authorize?mode=secondDRM</dashif:au
thzurl>
    </ContentProtection>
    <Representation mimeType="video/mp4" codecs="avc1.64001f" width="640" height="360" />
    <Representation mimeType="video/mp4" codecs="avc1.640028" width="852" height="480" />
</AdaptationSet>
```

The MPD provides DRM system configuration for DRM systems:

- For `FirstDRM`, the MPD provides complete DRM system configuration, including the optional `dashif:authzurl`. Two equivalent alternative URLs are provided for accessing the associated services.
- For `SecondDRM`, the MPD does not provide the license server URL. It must be supplied at runtime.

There are two encrypted representations in the adaptation set, each with a different codecs string. Both codecs strings are included in the required capability set of this adaptation set. A DRM system must support playback of both representations in order to present this adaptation set.

In addition to the MPD, a DASH client can use solution-specific logic and configuration for controlling DRM selection and configuration decisions (e.g. loading license server URLs from configuration data instead of the MPD). This is often implemented in the form of callbacks exposed by the DASH client to an "app" layer in which the client is hosted. It is assumed that when executing any such callbacks, a DASH client makes available relevant contextual data, allowing the business logic to make fully informed decisions.

The purpose of the DRM system selection workflow is to select a single DRM system that is capable of decrypting a meaningful subset of the adaptation sets selected for playback. The selected DRM system will meet the following criteria:

1. It is actually implemented by the media platform.
2. It supports a set of capabilities sufficient to present an acceptable set of adaptation sets.
3. The necessary DRM system configuration for this DRM system is available.

It may be that the selected DRM system is only able to decrypt a subset of the encrypted adaptation sets selected for playback. See also § 11.4 Handling unavailability of content keys.

The set of adaptation sets considered during selection does not need to be constrained to a single period, potentially enabling seamless transitions to a new period with a different set of content keys.

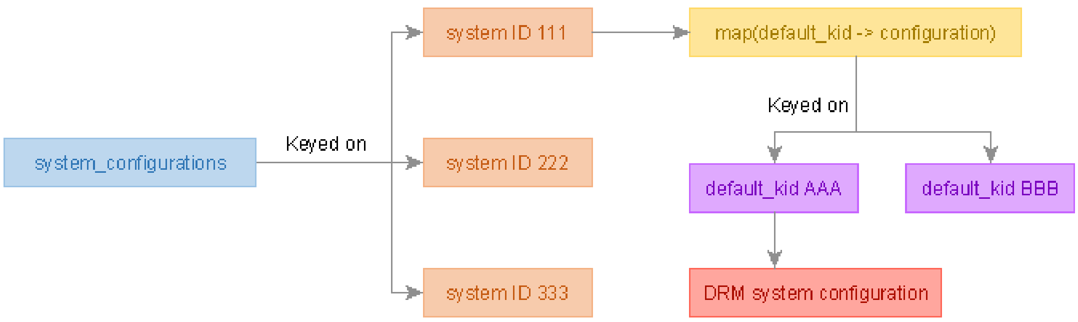
In live services new periods may be added over time, with potentially different DRM system configuration and required capability sets, making it necessary to re-execute the selection process.

> Note: If a new period has significantly different requirements in terms of DRM system configuration or the required capability sets, the media pipeline may need to be re-initialized to play the new period. This may result in a glitch/pause at the period boundary. The specifics are implementation-dependant.

The default DRM system configuration in the MPD of a live service can change over time. DASH clients are not expected to re-execute DRM workflows if the default DRM system configuration in the MPD changes for an adaptation set that has already been processed in the past. Such changes will only affect clients that are starting playback.

When encrypted adaptation sets are initially selected for playback or when the selected set of encrypted adaptation sets changes (e.g. because a new period was added to a live service), a DASH client SHOULD execute the following algorithm for DRM system selection:

1. Let *adaptation_sets* be the set of encrypted adaptation sets selected for playback.
2. Let *signaled_system_ids* be the set of DRM system IDs for which a `ContentProtection` descriptor is present in the MPD on any entries in *adaptation_sets*.
3. Let *candidate_system_ids* be an ordered list initialized with items of *signaled_system_ids* in any order.
4. Provide *candidate_system_ids* to solution-specific logic and configuration for inspection/modification.
   - This enables business logic to establish an order of preference where multiple DRM systems are present.
   - This enables business logic to filter out DRM systems known to be unsuitable.
   - This enables business logic to include DRM systems not signaled in the MPD.
5. Let *default_kids* be the set of all distinct `default_KID` values in *adaptation_sets*.
6. Let *system_configurations* be an empty map of `system ID -> map(default_kid -> configuration)`, representing the DRM system configuration of each `default_KID` for each DRM system.



7. For each *system_id* in *candidate_system_ids*:
   1. Let *configurations* be a map of `default_kid -> configuration` where the keys are *default_kids* and the values are the DRM system configurations initialized with data from `ContentProtection` descriptors in the MPD (matching on `default_KID` and *system_id*).
      - If there is no matching `ContentProtection` descriptors in the MPD, the map still contains a partially initialized DRM system configuration for the `default_KID`.
      - Enhance the MPD-provided default DRM system configuration with synthesized data where

appropriate (e.g. to generate W3C Clear Key initialization data in a format supported by the platform API).

2. Provide *configurations* to solution-specific logic and configuration for inspection and modification, passing *system_id* along as contextual information.

   - This enables business logic to override the default DRM system configuration provided by the MPD.
   - This enables business logic to inject values that were not embedded in the MPD.
   - This enables business logic to reject content keys that it knows cannot be used, by removing the DRM system configuration for them.

3. Remove any entries from *configurations* that do not contain all of the following pieces of data:

   - License server URL
   - DRM system initialization data in a format accepted by the particular DRM system; this is generally a `pssh` box [CENC], though some DRM systems also support other formats

4. Add *configurations* to *system_configurations* (keyed on *system_id*).

8. Remove from *candidate_system_ids* any entries for which the map of DRM system configurations in *system_configurations* is empty.
9. Let *required_capability_sets* be a map of `adaptation set -> capability set`, providing the required capability set of every item in *adaptation_sets*.
10. Match the capabilities of DRM systems with the required capability sets of adaptation sets:

    1. Let *supported_adaptation_sets* be an empty map of `system ID -> list of adaptation set`, incidating which adaptation sets are supported by which DRM systems.
    2. For each *system_id* in *candidate_system_ids*:

       1. Let *candidate_adaptation_sets* by the set of adaptation sets for which *system_configurations* contains DRM system configuration (keyed on *system_id* and then the `default_KID` of the adaptation set).

          - This excludes from further consideration any adaptation sets that could not be used due to lacking DRM system configuration, even if capabilities match.

       2. Let *maximum_capability_set* be the union of all values in *required_capability_sets* keyed on items of *candidate_adaptation_sets*.
       3. Query the DRM system identified by *system_id* with the capability set *maximum_capability_set*, assigning the output to *supported_capability_set*.

          - A DRM system that is not implemented is treated as having no capabilities.

       4. For each *adaptation_set* in *candidate_adaptation_sets*:

          1. If *supported_capability_set* contains all the capabilities in the corresponding entry in *required_capability_sets* (keyed on *adaptation_set*), add *adaptation_set* to the list in *supported_adaptation_sets* (keyed on *system_id*).

11. Remove from *supported_adaptation_sets* any entries for which the value (the set of adaptation sets) meets any of the following criteria:

    - The set is empty (the DRM system does not support playback of any adaptation set).
    - The set does not contain all encrypted media types present in the MPD (e.g. the DRM system can decrypt only the audio content but not the video content).

12. If *supported_adaptation_sets* is empty, playback of encrypted content is not possible and the workflow ends.

13. If *supported_adaptation_sets* contains multiple items, request solution-specific logic and configuration to select the preferred DRM system from among them.
    - This allows solution-specific logic and configuration to make an informed choice when different DRM systems can play different adaptation sets. Contrast this to the initial order of preference that was defined near the start of the algorithm, which does not consider capabilities.
14. If solution-specific logic and configuration does not make a decision, find the first entry in *candidate_system_ids* that is among the keys of *supported_adaptation_sets*. Remove items with any other key from *supported_adaptation_sets*.
    - This falls back to the "order of preference" logic and takes care of scenarios where business logic did not make an explicit choice.
15. Let *selected_system_id* be the single remaining key in *supported_adaptation_sets*.
16. Let *final_adaptation_sets* be the single remaining value in *supported_adaptation_sets*.
17. Let *final_configurations* (map of `default_KID -> DRM system configuration`) be the value from *system_configurations* keyed on *selected_system_id*.
18. Remove from *final_configurations* any entries keyed on `default_KID` values that are not used by any adaptation set in *final_adaptation_sets*.
    - These are the configurations of adaptation sets for which configuration was present but for which the required capabilities were not offered by the DRM system.
19. Prohibit playback of any encrypted adaptation sets that are not in *final_adaptation_sets*.
    - These are existing adaptation sets for which either no DRM system configuration exists or for which the required capabilities are not provided by the selected DRM system.
20. Execute the DRM system activation workflow, providing *selected_system_id* and *final_configurations* as inputs.

If a DRM system is successfully selected, activation and potentially one or more license requests will follow before playback can proceed. These related workflows are described in the next chapters.

11.3. Activating the DRM system§

Once a suitable DRM system has been selected, it must be activated by providing it a list of content keys that the DASH client requests to be made available for content decryption, together DRM system specific initialization data for each of the content keys. The result of activation is a DRM system that is ready to decrypt zero or more encrypted adaptation sets selected for playback.

During activation, it may be necessary to perform license requests in order to obtain some or all of the content keys and the usage policy that constrains their use. Some of the requested content keys may already be available to the DRM system, in which case no license request will be triggered.

> Note: The details of stored content key management and persistent DRM session management are out of scope of this document - workflows described here simply accept the fact that some content keys may already be available, regardless of why that is the case or what operations are required to establish content key persistence.

Once a suitable DRM system has been selected, a DASH client SHOULD execute the following algorithm to activate it:

1. Let *configurations* be the input to the algorithm; it is a map with the entry keys being `default_KID` values identifying the content keys and the entry values being the DRM system configuration to use with that particular content key.

2. Let *pending_license_requests* be an empty set.
3. For each *kid* and *config* pair in *configurations* invoke the platform API to activate the selected DRM system and signal it to make *kid* available for decryption, passing the DRM system the initialization data stored in *config*.
    - If the DRM system indicates that one or more license requests are needed, add any license request data provided by the DRM system and/or platform API to *pending_license_requests*, together with the associated *kid* and *config* values.
4. If *pending_license_requests* is not an empty set, execute the license request workflow and provide this set as input to the algorithm.
5. Inspect the set of content keys the DRM system indicates are now available and deselect from playback any adaptation sets for which the content key has not become available.
6. Inspect the set of remaining adaptation sets to determine if a sufficient data set remains for successful playback. Raise error if playback cannot continue.

The default format for initialization data supplied to a DRM system is a `pssh` box. However, if the DASH client has knowledge of any special initialization requirements of a particular DRM system, it MAY supply initialization data in other formats (e.g. the `keyids` JSON structure used by W3C Clear Key). Presence of initialization data in the expected format is considered during DRM system selection when determining whether a DRM system is a valid candidate.

For historical reasons, platform APIs often implement DRM system activation as a per-content-key operation. Some APIs and DRM system implementations may also support batching all the content keys into a single activation operation, for example by combining multiple "content key and DRM system configuration" data sets into a single data set in a single API call. DASH clients MAY make use of such batching where supported by the platform API. The workflow in this chapter describes the most basic scenario where activation must be performed separately for each content key.

> Note: The batching may, for example, be accomplished by concatenating all the `pssh` boxes for the different content keys. Support for this type of batching among DRM systems and platform APIs remains uncommon, despite the potential efficiency gains from reducing the number of license requests triggered.

11.4. Handling unavailability of content keys§

It is possible that not all of the encrypted adaptation sets selected for playback can actually be played back (e.g. because a content key for ultra-HD content is only authorized for use by implementations with a high robustness level). The unavailability of one or more content keys SHOULD NOT be considered a fatal error condition as long as at least one audio and at least one video adaptation set remains available for playback (assuming both content types are initially selected for playback). This logic MAY be overridden by solution specific business logic to better reflect end-user expectations.

A DASH client can request a DRM system to enable decryption using any set of content keys (if it has the necessary DRM system configuration). However, this is only a request and playback can be countermanded at multiple stages of processing by different involved entities.

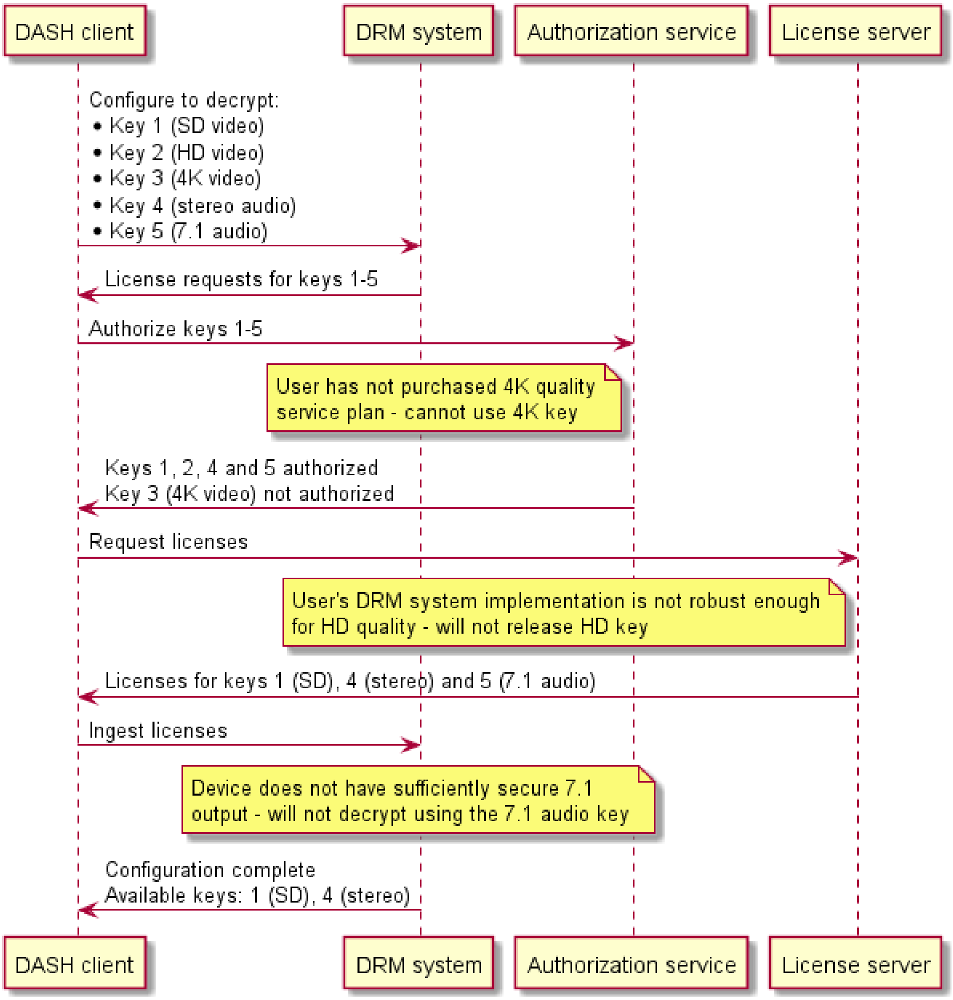


Figure 9 *The set of content keys made available for use can be far smaller than the set requested by a DASH client. Example workflow indicating potential instances of content keys being removed from scope.*

The set of available content keys is only known at the end of executing the activation workflow and may decrease over time (e.g. due to license expiration). The proper handling of unavailable keys depends on the limitations imposed by the platform APIs.

Media platform APIs often refuse to start or continue playback if the DRM system is not able to decrypt all the data already in media platform buffers.

It may be appropriate for a DASH client to avoid buffering data for encrypted adaptation sets until the required content key is known to be available. This allows the client to avoid potentially expensive buffer resets and rebuffering if unusable data needs to be removed from buffers.

Note: The DASH client should still download the data into intermediate buffers for faster startup and simply defer submitting it to the media platform API until key availability is confirmed.

11.5. Handling changes in required and available content keys§

The set of available content keys can change over time (e.g. due to license expiration or due to new periods in the presentation requiring different content keys).

If a content key expires during playback it is common for a media platform to pause playback until the content key can be refreshed with a new license or until data encrypted with the now-unusable content key is removed from buffers. DASH clients SHOULD acquire new licenses in advance of license expiration and SHOULD implement appropriate recovery/fallback behavior to ensure a minimally disrupted user experience in situations where some content keys remain available even after attempted license renewal.

A DASH client SHALL monitor the set of `default_KID` values that are required for playback and either request the DRM system to make these content keys available or deselect the affected adaptation sets when the content keys become unavailable. Conceptually, any such change can be handled by re-executing the DRM system selection and activation workflows, although platform APIs may also offer more fine-grained update capabilities.

> Note: Some CDM implementations emit license renewal signals using the EME `license-renewal` [encrypted-media] message. CDMs are not obligated to implement this mechanism and DASH clients cannot rely on this message as the only source of expiration information. In particular, the `MediaKeySession.expiration` property needs to be monitored to stay informed of upcoming license expiration.

A DASH client MAY enable solution-specific logic and configuration to disable proactive license acquisition, for example to enable scenarios where solution-specific logic and configuration explicitly triggers license requests at desired times and with desired parameters.

11.6. Content protection policies§

When content keys are acquired, the license that delivers them also supplies a policy for the DRM system, instructing it how to protect the content that is made accessible by the content keys.

¶EXAMPLE 17

Protection policy may define the following example requirements:

- All connected displays must support HDCP 2.2 or newer.
- The video display area must be no more than 1280x720 pixels.
- Minimum DRM system robustness level is "800".

Typical DRM systems will enforce the most restrictive protection policy from among all active content keys and will refuse to start playback if any of the constraints cannot be satisfied! As a result, it can be the case that even though only the constraints for a UHD video stream cannot be satisfied, playback of even the lower quality levels is blocked.

In many cases, it might be more desirable to instead exclude the UHD quality level from the set of adaptation sets selected for playback and DRM system activation. Alternatively, there may be a different DRM system implementation available on the device that is capable of satisfying the constraints. It is not possible for a DASH client to resolve these constraints as it has no knowledge of what policy applies nor of the capabilities of the different DRM system implementations.

Solution-specific logic and configuration SHOULD be used to select the most suitable DRM system, taking into consideration the protection policy, and to preemptively exclude adaptation sets from playback if it can be foreseen that the protection policy for their content keys cannot be satisfied. Likewise, license servers SHOULD NOT provide content keys if it can be foreseen that the recipient will be unable to satisfy their protection policy.

11.7. Performing license requests§

DASH clients performing license requests SHOULD follow the DASH-IF interoperable license request model. The remainder of this chapter only applies to DASH clients that follow this model. Alternative implementations are possible and in common use but are not interoperable and are not described in this document.

DRM systems generally do not perform license requests on their own. Rather, when they determine that a license is required, they generate a document that serves as the license request body and expect the DASH client to deliver it to a license server for processing. The latter returns a suitable response that, if a license is granted, encapsulates the content keys in an encrypted form only readable to the DRM system.

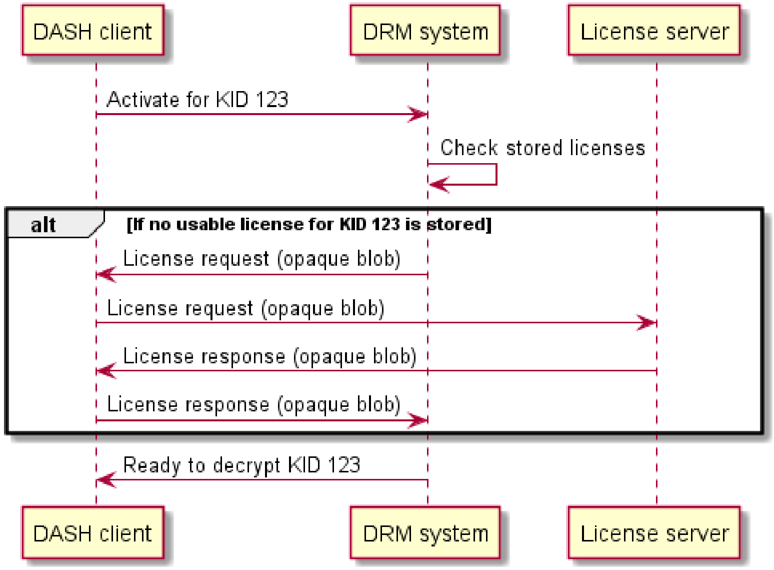


Figure 10 *Simplified conceptual model of license request processing. Many details omitted.*

The request and response body are in DRM system specific formats and considered opaque to the DASH client. A DASH client SHALL NOT modify the request body or the response body.

The license request workflow defined here exists to enable the following goals to be achieved without the need to customize the DASH client with logic specific to a DRM system or license server implementation:

1. Provide proof of authorization if the license server requires the DASH client to prove that the user being served has the rights to use the requested content keys.
2. Execute the license request workflow driven purely by the MPD, without any need for solution-specific logic and configuration.
3. Detect common error scenarios and present an understandable message to the user.

The proof of authorization is optional and the need to attach it to a license request is indicated by the presence of at least one `dashif:authzurl` in the DRM system configuration. The proof of authorization is a JSON Web Token in compact encoding (the `aaa.bbb.ccc` form) returned as the HTTP response body when the DASH client performs a GET request to this URL. The token is attached to a license request in the HTTP `Authorization` header with the `Bearer` type. For details, see § 10 DASH-IF interoperable license request model.

Error responses from both the authorization service and the license server SHOULD be returned as [rfc7807] compatible responses with a 4xx or 5xx status code and `Content-Type: application/problem+json`.

DASH clients SHOULD implement retry behavior to recover from transient failures and expiration of authorization tokens.

To process license requests queued during execution of the DRM system activation workflow, the client SHOULD execute the following algorithm:

1. Let *pending_license_requests* be the set of license requests that the DRM system has requested to be performed, with at least the following data present in each entry:
   - The license request body provided by the DRM system.
   - The DRM system configuration.
2. Let *retry_requests* be an empty set. It will contain the set of license requests that are to be retried due to transient failure.
3. Let *pending_authz_requests* be a map of `URL -> GUID[]`, with the keys being authorization service URLs and the values being lists of `default_KIDs`. The map is initially empty.
4. For each *request* in *pending_license_requests*:

    1. If the DRM system configuration does not contain at least one value for `dashif:authzurl`, skip to the next loop iteration. This means that no authorization token is to be attached to this license request.
    2. Create/update the entry in *pending_authz_requests* with the key being the set of `dashif:authzurl` values; add the `default_KID` to the list in the map entry value.
5. Let *authz_tokens* be a map of `GUID -> string`, with the keys being `default_KIDs` and the values being the associated authorization tokens. The map is initially empty.
6. For each *authz_url_set* and *kids* pair in *pending_authz_requests*:
    1. If the DASH client has a cached authorization token previously acquired for the same *authz_url_set* and *kids* combination that still remains valid according to its `exp` "Expiration Time" claim:
        1. Let *authz_token* be the cached authorization token.
    2. Else:
        1. Create a comma-separated list from *kids* in ascending alphanumeric (ASCII) order.
        2. Let *authz_url* be a random item from *authz_url_set*.
        3. Let *authz_url_with_kids* be *authz_url* with an additional query string parameter named `kids` with the value from *kids*.
            - *authz_url* may already include query string parameters, which should be preserved!
        4. Perform an HTTP GET request to *authz_url_with_kids* (following redirects).
            - Include any relevant HTTP cookies.
            - Allow solution-specific logic and configuration to intercept the request and inspect/modify it as needed (e.g. provide additional HTTP request headers to enable user identification).
        5. If the response status code indicates failure, make a note of any error information for later processing and skip to the next *authz_url*.
        6. Let *authz_token* be the HTTP response body.
        7. Submit *authz_token* into the DASH client cache, with the cache key being a combination of *authz_url_set* and *kids*, and the cache entry expiration being defined by the `exp` "Expiration Time" claim in the authorization token (defaulting to never expires).
    3. For each *kid* in *kids*, add an entry to *authz_tokens* with the key *kid* and the value being *authz_token*.
7. For each *request* in *pending_license_requests*:
    1. If the DRM system configuration from *request* contains an authorization service URL but there is no entry in *authz_tokens* keyed on the `default_KID` from *request*, skip to the next loop iteration.
        - This occurs when an authorization token is required but cannot be obtained for this license request.
    2. Execute an HTTP POST request with the following parameters:
        - Request body is the license request body from *request*.
        - Request URL is defined by DRM system configuration. If multiple license server URLs are defined, select a random URL from the set.
        - If *authz_tokens* contains an entry with the key being the `default_KID` from *request*, add the `Authorization` header with the value being the string `Bearer` concatenated with a space and the authorization token from *authz_tokens* (e.g. `Bearer aaa.bbb.ccc`).
    3. If the response status code indicates failure:
        1. Expel the used authorization token (if any) from the DASH client cache to force a new token to be used for any future license requests.

2. If the DASH client believes that retrying the license request might succeed (e.g. because the response indicates that the error might be transient or due to an expired authorization token that can be renewed), add *request* to *retry_requests*.
3. Make a note of any error information for later processing and presentation to the user.
4. Skip to the next loop iteration.

4. Submit the HTTP response body to the DRM system for processing.
   - This may cause the DRM system to trigger additional license requests. Append any triggered request to *pending_license_requests* and copy the DRM system configuration from the current entry, processing the additional entry in a future iteration of the same loop.
   - If the DRM system indicates a failure to process the data, make a note of any error information for later processing and skip to the next loop iteration.

8. If *retry_requests* is not empty, re-execute this workflow with *retry_requests* as the input.

While the above algorithm is presented sequentially, authorization requests and license requests may be performed in a parallelized manner to minimize processing time.

At the end of this algorithm, all pending license requests have been performed. However, it is not necessary that all license requests or authorization requests succeed! For example, even if one of the requests needed to obtain an HD quality level content key fails, other requests may still make SD quality level content keys available, leading to a successful playback if the HD quality level is deselected by the DASH client. Individual failing requests therefore do not indicate a fatal error. Rather, such error information should be collected and provided to the top-level error handler of the DRM system activation workflow, which can make use of this data to present user-friendly messages if it decides that meaningful playback cannot take place with the final set of available content keys. See also § 11.4 Handling unavailability of content keys.

11.7.1. Efficient license acquisition§

In some situations a DASH client can foresee the need to make new content keys available for use or to renew the licenses that enable content keys to be used. For example:

- Live DASH services can at any time introduce new periods that use different content keys. They can also alternmate between encrypted and clear content in different periods.
- The license that enables a content key to be used can have an expiration time, after which a new license is required.

DASH clients SHOULD perform license acquisition ahead of time, activating a DRM system before it is needed or renewing licenses before they expire. This provides the following benefits:

- Playback can continue seamlessly when licenses are renewed, without pausing for license acquisition.
- New content keys are already available when content needs them, again avoiding a pause for license acquisition.

To avoid a huge number of concurrent license requests causing license server overload, a DASH client SHOULD perform a license request at a randomly selected time between the moment when it became aware of the need for the license request and the time when the license must be provided to a DRM system (minus some safety margin).

Multiple license requests to the same license server with the same authorization token SHOULD be batched into a single request if the media platform API supports this. See § 11.3 Activating the DRM system for details.

The possibility for ahead-of-time DRM system activation, seamless license renewal and license request batching depends on the specific DRM system and media platform implementations. Some implementations might not support optimal behavior.

12. Periodic re-authorization§

In a live DASH presentation the rights of the user can be different for different programs included in the presentation. This chapter describes recommended mechanisms for forcing rights to be re-evaluated at program boundaries.

The user's level of access to content is governed by the issuance (or not) of licenses with content keys and the policy configuration carried by the licenses. The authorization server is the authority on what rights are assigned to the user and this is enforced by the license server. To force re-evaluation of rights, a service must force a new license request to be made. This can be accomplished by:

1. Defining an expiration time on the license.
2. Changing the content key to one that is not yet available to DASH clients, thereby triggering DRM system activation for the new content key.

Not every DRM system supports real-time license expiration - some widely used implementations only check license validity at activation time. Therefore the latter option is a more universally applicable method to force re-evaluation of access rights. As changing the content key is only possible on DASH period boundaries as the initialisation segment is updated, live DASH presentations SHOULD create a new period in which content is encrypted with new content keys to force re-evaluation of user's access rights.

> Note: Changing the content keys does not increase the cryptographic security of content protection. The term *periodic re-authorization* is therefore used here instead of *key rotation*, to maintain focus on the goal and not the mechanism.

13. Controlling access rights with a key hierarchy§

Using a key hierarchy allows a single content key to selectively unlock only a subset of a DASH presentation and apply license policy updates without the need to perform license requests at every program boundary. This mechanism is a specialization of periodic re-authorization for scenarios where license requests at program boundaries are not always desirable or possible.

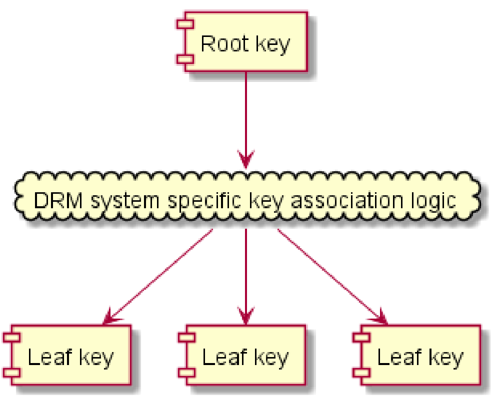


Figure 11 *A key hierarchy establishes a DRM system specific relationship between a root key and a set of leaf keys.*

A key hierarchy defines a multi-level structure of cryptographic keys, instead of a single content key:

- **Root keys** take the place of content keys in DASH client workflows.
- **Leaf keys** are used to encrypt the media samples.

A root key might not be an actual cryptographic key. Rather, it acts as a reference to identify the set of leaf keys that protect content. A DASH client requesting a license for a specific root key will be interpreted as requesting a license that makes available all the leaf keys associated with that root key.

Note: Intermediate layers of cryptographic keys may also exist between root keys and leaf keys but such layers are DRM system specific and only processed by the DRM system, being transparent to the DASH client and the media platform. To a DASH client, only the root keys have meaning. To the media platform, only the leaf keys have meaning.

This layering enables the user's rights to content to be evaluated in two ways:

1. Changing the root key invokes the full re-evaluation workflow as a new license request must be made by the DASH client.
2. Changing the leaf key invokes an evaluation of the rights granted by the license for the root key and processing of any additional policy attached to the leaf key. If result of this evaluation indicates the leaf key cannot be used, the DRM system will signal playback failure to the DASH client.

Changing the root key is equivalent to changing the content key in terms of content and MPD signaling, requiring a new period to be started. The leaf key can be changed in any media segment and does not require modification of the MPD. Leaf keys SHOULD NOT be changed within the same program. Changing leaf keys on a regular basis does not increase cryptographic security.

Note: A DASH service with a key hierarchy is sometimes referred to as using "internal key rotation".

The mechanism by which a set of leaf keys is made available based on a request for a root key is DRM system specific. Nevertheless, different DRM systems may be interoperable as long as they can each make available the required set of leaf keys using their system-specific mechanisms, using the same root key as the identifier for the same set of leaf keys.

When using a key hierarchy, the leaf keys are typically delivered in-band in the media segments, using `moof/pssh` boxes, together with additional/updated license policy constraints. The exact implementation is DRM system specific and transparent to a DASH client.

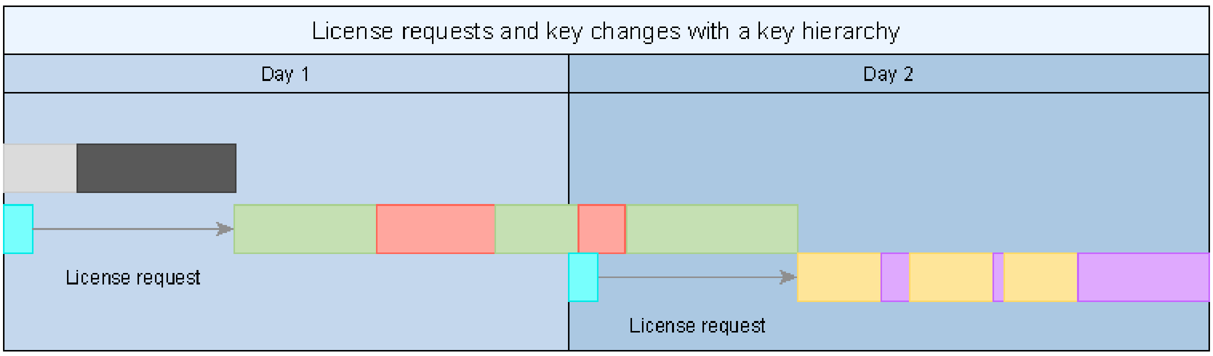


Figure 12 *Different rows indicate root key changes. Color alternations indicate leaf key changes. A key hierarchy enables per-program access control even in scenarios where a license request is only performed once per day. The single license request makes available all the leaf keys that the user is authorized to use during the next epoch.*

A key hierarchy is useful for broadcast scenarios where license requests are not possible at arbitrary times (e.g. when the system operates by performing nightly license updates). In such a scenario, this mechanism enables user access rights to be cryptographically enforced at program boundaries, defined on the fly by the service provider, while re-evaluating the access rights during moments when license requests are possible. At the same time, it enables the service provider to supply in-band updates to license policy (when supported by the DRM system).

Similar functionality could be implemented without a key hierarchy by using a separate content key for each program and acquiring all relevant licenses in advance. The advantages of a key hierarchy are:

- Greatly reduced license acquisition traffic and required license storage size, as DRM systems are optimized for efficient handling of large numbers of leaf keys.

- Ability for the service provider to adjust license policy at any time, not only during license request processing (if in-band policy updates are supported by the DRM system).

14. Use of W3C Clear Key with DASH§

Clear Key is a DRM system defined by W3C in [encrypted-media]. It is intended primarily for client and media platform development/test purposes and does not perform the content protection and content key protection duties ordinarily expected from a DRM system. Nevertheless, in DASH client DRM workflows, it is equivalent to a real DRM system.

A DRM system specific `ContentProtection` descriptor for Clear Key SHALL use the system ID `e2719d58-a985-b3c9-781a-b030af78d30e` and `value="ClearKey1.0"`.

The `dashif:laurl` element SHOULD be used to indicate the license server URL. Legacy content MAY also use an equivalent `Laurl` element from the `http://dashif.org/guidelines/clearKey` namespace, as this was defined in previous versions of this document (the definition is now expanded to also cover non-clearkey scenarios). Clients SHOULD process the legacy element if it exists and `dashif:laurl` does not.

The license request and response format is defined in [encrypted-media].

W3C describes the use of the system ID `1077efec-c0b2-4d02-ace3-3c1e52e2fb4b` in [eme-initdata-cenc] section 4 to indicate that tracks are encrypted with Common Encryption. However, the presence of this "common" `pssh` box does not imply that Clear Key is to be used for decryption. DASH clients SHALL NOT interpret a `pssh` box with the system ID `1077efec-c0b2-4d02-ace3-3c1e52e2fb4b` as an indication that the Clear Key mechanism is to be used (nor as an indication of anything else beyond the use of Common Encryption).

EXAMPLE 18

An example of a Clear Key `ContentProtection` descriptor using `laurl` is as follows.

```
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:dashif="https://dashif.org/">
 <Period>
  <AdaptationSet>
   <ContentProtection schemeIdUri="urn:uuid:e2719d58-a985-b3c9-781a-b030af78d30e" value="ClearKey1.0">
     <dashif:laurl>https://clearKeyServer.foocompany.com</dashif:laurl>
     <dashif:laurl>file://cache/licenseInfo.txt</dashif:laurl>
   </ContentProtection>
  </AdaptationSet>
 </Period>
</MPD>
```

Parts of the MPD structure that are not relevant for this chapter have been omitted - this is not a fully functional MPD file.

15. XML Schema for DASH-IF MPD extensions§

The namespace for the DASH-IF MPD extensions is `https://dashif.org/`. This document refers to this namespace using the `dashif` prefix. The XML schema of the extensions is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:dashif="https://dashif.org/"
    targetNamespace="https://dashif.org/">

    <xs:element name="laurl" type="xs:anyURI"/>
    <xs:element name="authzurl" type="xs:anyURI"/>
</xs:schema>
```

16. HTTPS and DASH§

Transport security in HTTP-based delivery may be achieved by using HTTP over TLS (HTTPS) as specified in [RFC 8446]. HTTPS is a protocol for secure communication which is widely used on the Internet and also increasingly used for content streaming, mainly for protecting:

- The privacy of the exchanged data from eavesdropping by providing encryption of bidirectional communications between a client and a server, and
- The integrity of the exchanged data against forgery and tampering.

As an MPD carries links to media resources, web browsers follow the W3C recommendation [mixed-content]. To ensure that HTTPS benefits are maintained once the MPD is delivered, it is recommended that if the MPD is delivered with HTTPS, then the media also be delivered with HTTPS.

DASH also explicitly permits the use of HTTPS as a URI scheme and hence, HTTP over TLS as a transport protocol. When using HTTPS in an MPD, one can for instance specify that all media segments are delivered over HTTPS, by declaring that all the `BaseURL`'s are HTTPS based, as follow:

```
<BaseURL>https://cdn1.example.com/</BaseURL>
<BaseURL>https://cdn2.example.com/</BaseURL>
```

One can also use HTTPS for retrieving other types of data carried with a MPD that are HTTP-URL based, such as, for example, DRM licenses specified within the `ContentProtection` descriptor:

```
<ContentProtection
    schemeIdUri="urn:uuid:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    value="DRMNAME version"
    <dashif:laurl>https://MoviesSP.example.com/protect?license=kljklsdfiowek</dashif:laurl>
</ContentProtection>
```

It is recommended that HTTPS be adopted for delivering DASH content. It should be noted nevertheless, that HTTPS does interfere with proxies that attempt to intercept, cache and/or modify content between the client and the TLS termination point within the CDN. Since the HTTPS traffic is opaque to these intermediate nodes, they can lose much of their intended functionality when faced with HTTPS traffic.

While using HTTPS in DASH provides good protection for data exchanged between DASH servers and clients, HTTPS only protects the transport link, but does not by itself provide an enforcement mechanism for access control and usage policies on the streamed content. HTTPS itself does not imply user authentication and content authorization (or access control). This is especially the case that HTTPS provides no protection to any streamed content cached in a local buffer at a client for playback. HTTPS does not replace a DRM.

Index§

Terms defined by this specification§

References§

Normative References§

[CENC]
Information technology — MPEG systems technologies — Part 7: Common encryption in ISO base media file format files. February 2016. Published. URL: https://www.iso.org/standard/68042.html

[CMAF]
Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media. March 2020. Published. URL: https://www.iso.org/standard/79106.html

[DASH]
Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats. December 2019. Published. URL: https://www.iso.org/standard/79329.html

[EME-INITDATA-CENC]
David Dorwin; et al. "cenc" Initialization Data Format. 15 September 2016. NOTE. URL: https://www.w3.org/TR/eme-initdata-cenc/

[ENCRYPTED-MEDIA]
David Dorwin; et al. Encrypted Media Extensions. 18 September 2017. REC. URL: https://www.w3.org/TR/encrypted-media/

[ISOBMFF]
Information technology — Coding of audio-visual objects — Part 12: ISO Base Media File Format. December 2015. International Standard. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c068960_ISO_IEC_14496-12_2015.zip

[JWS]
M. Jones; J. Bradley; N. Sakimura. JSON Web Signature (JWS). May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7515

[JWT]
M. Jones; J. Bradley; N. Sakimura. JSON Web Token (JWT). 6 July 2012. Internet Draft. URL: https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-01

[MIXED-CONTENT]
Mike West. Mixed Content. 2 August 2016. CR. URL: https://www.w3.org/TR/mixed-content/

[RFC2119]
S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC7807]
M. Nottingham; E. Wilde. Problem Details for HTTP APIs. March 2016. Proposed Standard. URL: https://tools.ietf.org/html/rfc7807

[RFC8446]
E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. August 2018. Proposed Standard. URL: https://tools.ietf.org/html/rfc8446

Informative References§

[HLS-LowLatency]
Protocol Extension for Low-Latency HLS (Preliminary Specification). URL: https://developer.apple.com/documentation/http_live_streaming/protocol_extension_for_low-latency_hls_preliminary_specification

[ISO23001-12]
Information technology — MPEG systems technologies — Part 12: Sample variants. December 2018. Published. URL: https://www.iso.org/standard/74431.html

[JWE]
M. Jones; J. Hildebrand. JSON Web Encryption (JWE). May 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7516

[MSPR-EncryptionModes]
PlayReady Content Encryption Modes. URL: https://docs.microsoft.com/en-us/playready/packaging/content-encryption-modes

Issues Index§

ISSUE 1 The above paragraph on URL handling should be generalized to all sets of alternative URLs but there does not seem to be a suitable chapter in v4.3 If such a chapter is created in v5, we could replace the above paragraph with a reference to the general URL handling guidelines. ↵

ISSUE 2 Let’s come up with a good set of useful problem types we can define here, to reduce the set of problem types that must be defined in solution-specific scope. ↵

↑