



DASH-IF Implementation Guidelines: Content Protection Information Exchange Format (CPIX)

Commit Snapshot, 12 July 2023

Issue Tracking:

[GitHub](#)

Editor:

DASH Industry Forum

Table of Contents

1	Scope
2	Disclaimer
3	Normative Language
4	Introduction
5	Definition of Terms
6	Use Cases and Requirements
6.1	Introduction
6.2	Overview of the End to End Architecture
6.3	Use Cases for the Preparation of Content
6.3.1	Introduction
6.3.2	On-Demand Content
6.3.3	Live Content
6.3.4	Catch-up
6.4	Exchange over an Interface
6.4.1	Introduction
6.4.2	Content Key Delivery to One Entity
6.4.3	Secure Content Key Delivery to Several Entities
6.4.4	Content Key Delivery with Usage Rules
6.4.4.1	Introduction
6.4.4.2	Label Filter
6.4.4.3	Key Period Filter
6.4.4.4	Policy-based Filters
6.4.5	Content Key Delivery with DRM Signaling
6.4.6	Incremental Update and Extension of the Document
6.4.7	Content Key Hierarchy Delivery for Content Packaging
6.4.8	Root Key Delivery for License Server Operation
6.5	Workflow Examples
6.5.1	Encryptor Producer and Encryptor Consumer
6.5.1.1	Introduction
6.5.1.2	Encryptor Producer

- 6.5.1.3 Encryptor Consumer
- 6.5.2 Multiple Producers

7 XSD Schema Definition

- 7.1 Introduction
- 7.2 Requirements
- 7.3 Structure Overview
- 7.4 Hierarchical Data Model
 - 7.4.1 Introduction
 - 7.4.2 CPIX Element
 - 7.4.3 DeliveryDataList Element
 - 7.4.4 DeliveryData Element
 - 7.4.5 ContentKeyList Element
 - 7.4.6 ContentKey Element
 - 7.4.7 DRMSystemList Element
 - 7.4.8 DRMSystem Element
 - 7.4.9 HLSSignalingData Element
 - 7.4.10 ContentKeyPeriodList Element
 - 7.4.11 ContentKeyPeriod Element
 - 7.4.12 ContentKeyUsageRuleList Element
 - 7.4.13 ContentKeyUsageRule Element
 - 7.4.14 Usage Rules Filters
 - 7.4.14.1 Introduction
 - 7.4.14.2 KeyPeriodFilter Element
 - 7.4.14.3 LabelFilter Element
 - 7.4.14.4 VideoFilter Element
 - 7.4.14.5 AudioFilter Element
 - 7.4.14.6 BitrateFilter Element
 - 7.4.15 UpdateHistoryItemList Element
 - 7.4.16 UpdateHistoryItem Element

8 Key Management

- 8.1 Key Encryption in the CPIX Document
 - 8.1.1 Introduction
 - 8.1.2 Keys Used to Secure the CPIX Document
 - 8.1.3 Authenticated Encryption
 - 8.1.4 Digital Signature
 - 8.1.5 Mandatory Algorithms
- 8.2 Key Rotation Support
- 8.3 Content Keys with Several Protection Encryption Schemes

9 Examples

10 Abbreviations

References

Normative References

Candidate Technical Specification

DASH-IF CTS	Part XX	rev 0	Current version 2.3.1		
Status:	Draft	Internal Review	Community Review	IOP Review	IPR Review
Title:	DASH-IF Implementation Guidelines: Content Protection Information Exchange Format (CPIX)				
Source:	DASH-IF Interoperability Working Group,				

**Supporting
Companies:****Category:** Candidate Technical Specification**Date:** 2023-07-12

Abstract: This document defines a container allowing the exchange between entities of content protection information typically made of keys used for encrypting content and any associated DRM specific information. There may be one or several keys and these keys may be protected by one or several DRMs, hence there may be one or several DRM specific information. There is no assumption on the entities exchanging this information but it is not expected that a client device will use this exchange format. The goal is to allow entities involved in the content preparation workflow to get the content protection information so that, for example a DASH MPD can be generated with all content protection information. Because the defined container is not made for a specifically defined content preparation workflow but is generic, conformance is not considered to be a critical part of CPIX. As a consequence, no conformance is defined for this specification.

Disclaimer: This document is a candidate Technical Specification. DASH-IF is expecting to publish this initially, but to submit the specification to a formal specification organization. The primary choice is ETSI, for which DASH-IF has a PAS agreement.
This document is approved by DASH-IF IOP will be sent for IPR Review as well as to ETSI for publication.

**Expected
Publication:****Other
Comments:**

1. Scope§

The scope of this document is to define a Content Protection Information Exchange Format (CPIX). A CPIX document contains keys and DRM information used for encrypting and protecting content and can be used for exchanging this information among entities needing it in many possibly different workflows for preparing, for example, DASH or HLS content. The CPIX document itself can be encrypted, signed and authenticated so that its receivers can be sure that its confidentiality, source and integrity are also protected.

This specification describes version 2.3.1 of the CPIX document format. Detailed changes with respect to version 2.3 are tracked on [GitHub](#). Highlighted changes are:

- Clarified that key ids shall be UUID as described in [\[MPEGCENC\]](#) (added a constraint)
- Corrected a bug on @periodId that shall be a XS:REFID
- Clarification on the @explicitIV attribute under the [<ContentKey>](#) element encoding
- Taking into account the scenario described in Clause 9 of [\[DASHIF-IOPv5p6\]](#) for key rotation, clarification for the PSSH and ContentProtectionData content under DRMSystem. A clause is also added on this topic.
- Clean-up the Use Cases and Requirements clause (removed the electronic sell through use csae)
- Updated references
- Addition of the @version attribute under the [<CPIX>](#) element
- Addition of a clause on using the same content key with different encryption schemes

2. 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 <https://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.

3. Normative Language§

In the present document **shall**, **shall not**, **should**, **should not**, **may**, **need not**, **will**, **will not**, **can** and **cannot** are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](#) (Verbal forms for the expression of provisions). **must** and **must not** are NOT allowed in deliverables except when used in direct citation.

4. Introduction§

This document defines a container allowing the exchange between entities of content protection information typically made of keys used for encrypting content and any associated DRM specific information. There may be one or several keys and these keys may be protected by one or several DRMs, hence there may be one or several DRM specific information. There is no assumption on the entities exchanging this information but it is not expected that a client device will use this exchange format. The goal is to allow entities involved in the content preparation workflow to get the content protection information so that, for example a DASH MPD can be generated with all content protection information.

Because the defined container is not made for a specifically defined content preparation workflow but is generic, conformance is not considered to be a critical part of CPIX. As a consequence, no conformance is defined for this specification.

5. Definition of Terms§

Content

One or more audio-visual elementary streams and the associated MPD if in DASH format.

Content Key

A cryptographic key used for encrypting part of the Content.

Content Key Context

The portion of a media stream which is encrypted with a specific Content Key.

Content Protection

The mechanism ensuring that only authorized devices get access to Content.

Document Key

A cryptographic key used for encrypting the Content Key(s) in the CPIX document.

DRM Signaling

The DRM specific information to be added in Content for proper operation of the DRM system when authorizing a device for this Content. It is made of proprietary information for licensing and key retrieval.

PSSH

Protection System Specific Header box that is part of an ISOBMFF file. This box contains DRM Signaling.

6. Use Cases and Requirements

6.1. Introduction

Content Keys and DRM Signaling, a.k.a. content protection information need to be created and exchanged between some system entities when preparing Content. The flows of information are of very different nature depending on where Content Keys are created and also depending on the type of Content that can be either On-Demand or Live.

This clause presents different use cases where such exchanges are required. [§ 6.2 Overview of the End to End Architecture](#) is an overview of the general context in which exchange of content protection information is happening, [§ 6.3 Use Cases for the Preparation of Content](#) describes some workflows for content creation and [§ 6.4 Exchange over an Interface](#) goes in the details of how content protection information can be exchanged over an interface between two entities.

6.2. Overview of the End to End Architecture

This clause gives a general overview of the context in which content protection information need to be exchanged between entities in the backend. It completes clause 4 of [\[DASHIF-IOPv5p6\]](#) by putting more emphasis on the backend aspects.

This clause takes DASH content as an example for providing more specific and clear understanding, but this can be generalized to other streaming formats, such as HLS [\[HLS\]](#).

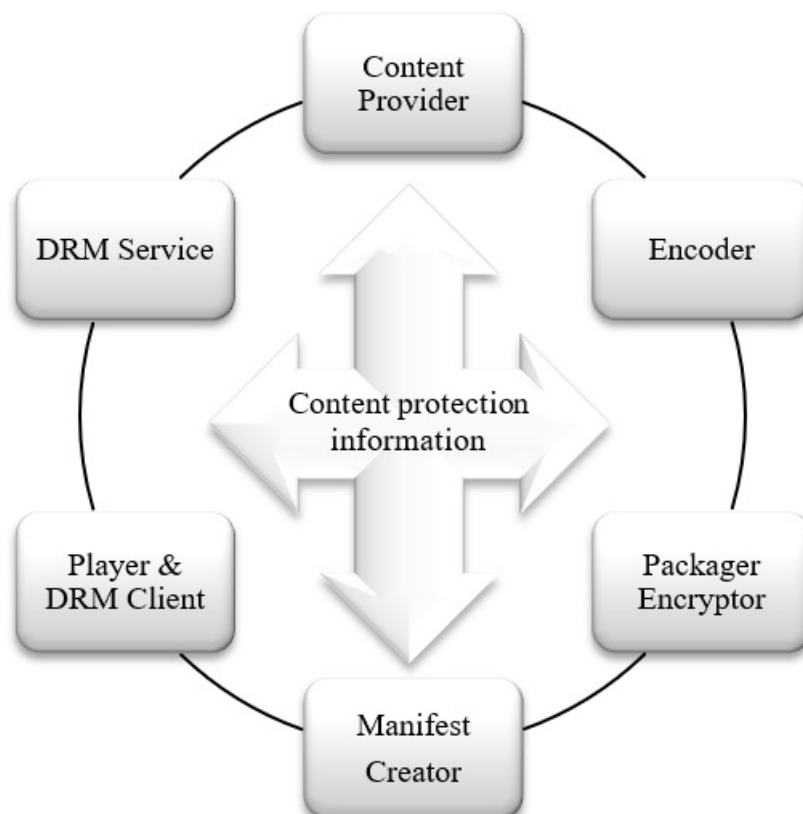


Figure 1 Logical roles that exchange DRM information and media.

The figure above shows logical entities that may send or receive content protection information such as content keys, asset identifiers, licenses, and DRM signaling (license acquisition information for example). A physical entity may combine multiple logical roles, and the point of origin for information, such as content keys and asset identifiers, can differ; so various information flows are possible. This is an example of how the roles are distributed to facilitate the description of workflow and use cases. Alternative roles and functions can be applied to create conformant content. The different roles are:

Content Provider

A publisher who provides the rights and rules for delivering protected media, also possibly source media (mezzanine format, for transcoding), asset identifiers, key identifiers (KID), content key values, encoding instructions, and content description metadata.

Encoder

A service provider who encodes media in a specified set of formats with different bitrates and resolutions etc., possibly determined by the publisher.

Packager / Encryptor

A service provider who encrypts and packages media, inserting DRM Signaling and metadata into the media files. In the case of DASH packaging, this consists of adding the default_KID in the file header tenc box, initialization vectors (IV) and subsample byte ranges in track fragments indexed by saio and saiz boxes, and possibly one or more pssh boxes containing license acquisition information (from the DRM Service). Tracks that are partially encrypted or encrypted with multiple keys require sample to group boxes and sample group description boxes in each track fragment to associate different KIDs to groups of samples. The Packager could originate values for KIDs, Content Keys, encryption layout, etc., then send that information to other entities that need it, including the DRM Service, and probably the Content Provider. However, the Packager could receive that information from a different point of origin, such as the Content Provider or DRM Service.

Manifest Creator

A service provider which generates the media manifests which group the various media files into a coherent presentation. These manifest files may contain DRM signaling information. For DASH, the MPD Creator is assumed to create one or more types of DASH MPD files and provide indexing of segments and/or sidx indexes for download so that players can byte range index subsegments. The MPD shall include descriptors for Common Encryption and DRM systems and should include identification of the @default_KID for each AdaptationSet element, and sufficient information in ContentProtection elements to acquire a DRM license. The @default_KID is available from the Packager and any other role that created it, and the DRM signaling is available from the DRM Service

DRM Client

It gets information from different sources: media manifest files, media files, and DRM licenses.

DRM Service

The DRM Service creates licenses containing a protected Content Key that can only be decrypted by a trusted DRM Client. It needs to know the @default_KID and DRM systemID and possibly other information like asset ID in order to create and download one or more licenses required for a Presentation on a particular device. Each DRM system has different license acquisition information, a slightly different license acquisition protocol, and a different license format with different playback rules, output rules, revocation and renewal system, etc. For DASH, the DRM Service typically shall supply the Packager license acquisition information for each ContentProtection element or pssh box, respectively. The DRM Service may also provide logic to manage key rotation, DRM domain management, revocation and renewal and other Content Protection related features.

6.3. Use Cases for the Preparation of Content

6.3.1. Introduction

This clause describes some workflows for content preparation where content protection information is exchanged between or carried through some entities.

As for the previous clause, this clause takes DASH content as an example for providing more specific and clear understanding, but this can be generalized to other streaming formats, such as HLS.

6.3.2. On-Demand Content

The flow for preparing On-Demand Content requires that a media asset is available non-encrypted, ideally in the maximum resolution so that an adaptive streaming presentation can be prepared.

One possible flow is that a Content Management System (CMS) creates a workflow ensuring that DASH Content is prepared. The CMS makes the file available to a transcoder. The transcoder outputs the segmented files that can be encrypted. The encryption engine either generates the Content Keys or requests them from a DRM system. The DRM system also provides pssh boxes to be added to the media files, as well as ContentProtection elements to be added to the MPD file. When the encrypted DASH Content is ready, the MPD is generated by a MPD Generator. It asks the DRM system the required DRM Signaling to be added in the MPD. DASH content is then uploaded by the CMS on a CDN making it available to users. In parallel, editorial metadata is exported to the Portal, enabling access to users. DRM systems receive relevant metadata information that needs to be included in the license (output controls) when creating a license.

This flow is summarized in the figure below where arrows show the flow of information.

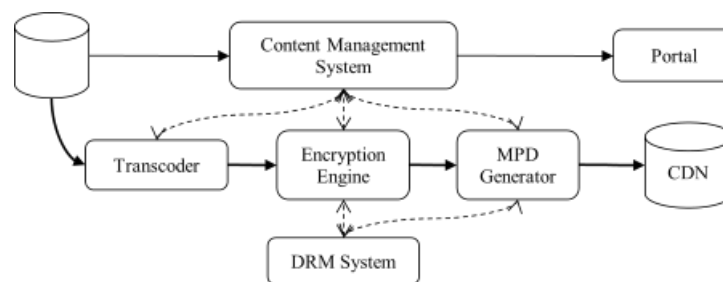


Figure 2 Example of workflow for On-Demand Content preparation.

6.3.3. Live Content

Metadata is regularly imported with new or updated information. Metadata can include different type of information on the EPG events such as the duration of the event, the list of actors, the output controls usage rules, a purchase window, etc.

Content is continuously received, transcoded in the desired format and encrypted if any type of entitlement is required.

One or many Content Keys can be used if key rotation is used or not. Such setting is static and configuration is hard-coded in the relevant equipment, hence a Content Management System is not required for this workflow to operate. As for Content on-Demand, keys are generated by the encryption engine or the DRM system and are available to all DRM systems and the encryption engine at the right moment depending on how these keys are used. The encoder requests to the DRM systems their specific signaling, if any, to be added in the MPD.

Encrypted segments and the media manifest are uploaded on a CDN making it available to users.

Metadata is exported to the Portal, enabling access to users. DRM systems receive relevant metadata information that needs to be included in the license (output controls).

This flow is summarized in the figure below where arrows show the flow of information.

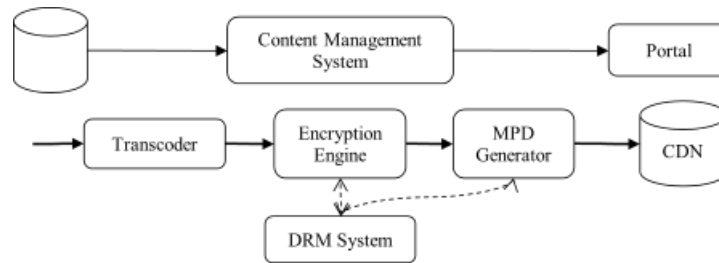


Figure 3 Example of workflow for Live Content preparation.

6.3.4. Catch-up

Live Content has already been encoded and encrypted (if required) for Live unicast. All DRM systems have access to the keys.

Additional metadata may be required for ensuring that events are effectively available in catch-up. These are made available to the Portal and some Live events are identified as being able to be replayed as On-demand. Optionally, the operator may choose to replace the advertising content with targeted ads.

6.4. Exchange over an Interface

6.4.1. Introduction

This clause gives details on how content protection information is exchanged or transferred over an interface between two or more entities.

6.4.2. Content Key Delivery to One Entity

In the simplest use case, content protection information is made of a Content Key. One entity sends a Content Key to the other entity.

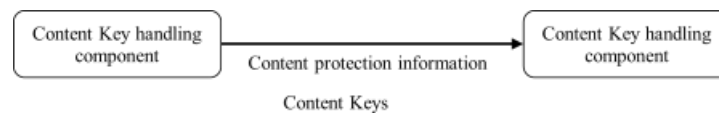


Figure 4 Content Key delivery to one entity.

The primary data model carried by content protection information document is made of one to many Content Keys with their associated KIDs. Any context or meaning is attributed externally. The document simply serves as a standard way to serialize Content Keys for delivery.

6.4.3. Secure Content Key Delivery to Several Entities

This use case is an extension of clause § 6.4.2 Content Key Delivery to One Entity and is compatible with the use cases presented in the following clauses.

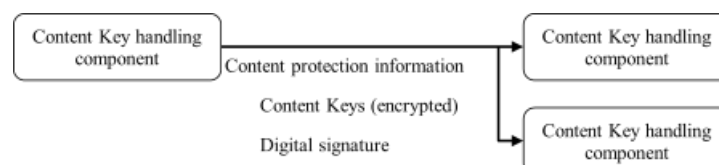


Figure 5 Secure Content Key delivery to several entities.

The entities exchanging Content Keys may want to rely upon a trust relationship that ensures authentication and privacy of communications. Such a mechanism can be provided by the communication protocol used to deliver the document but the document can also be self-protected. CPIX documents can deliver Content Keys in encrypted and digitally signed form, enabling confidentiality, authentication and nonrepudiation.

In situations with more than one recipient, the document allows each one to decrypt the Content Keys using its own private key.

6.4.4. Content Key Delivery with Usage Rules

6.4.4.1. Introduction

These use cases are extension of [§ 6.4.2 Content Key Delivery to One Entity](#) and present different rules that can be applied on a Content Key when delivered to an entity. Each usage rule defines a set of filters that are used to define a Content Key Context. If a rule match is found, the Content Key referenced by the usage rule is to be used to encrypt the Content Key Context defined by the rule.

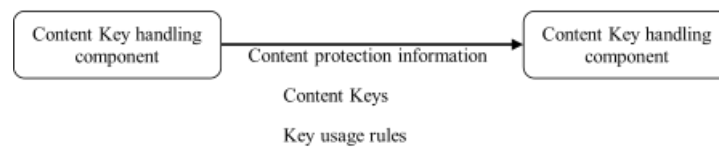


Figure 6 Content Key delivery with key usage rules.

6.4.4.2. Label Filter

This use case adds information to Content Keys that specifies how they are to be mapped to labelled Content Key Contexts, where the labeling system has been pre-agreed between the producer and consumer of the CPIX document.

For example, labels might be the IDs of DASH adaptation sets or, for more compatibility with formats other than DASH, names of media files/directories or input values for arbitrary custom logic.

The recipient will use the added information to map Content Keys to Content Key Contexts defined by labels.

6.4.4.3. Key Period Filter

This use case adds information to Content Keys that specifies how they are to be mapped to key periods, a.k.a. crypto-periods for Content Key rotation. The mapping is accomplished by defining key periods and mapping Content Keys to any number of key periods.

The recipient will use the added information to map Content Keys to time periods.

6.4.4.4. Policy-based Filters

This use case associates policy-based information with Content Keys, constraining how they define Content Key Contexts. Policy based filters are, for example, video or audio stream attributes and bitrate ranges.

The recipient will use the added information to map Content Keys to Content Key Contexts according to the defined policy.

Having no policy in some dimension means that the Content Key Context is not constrained in that dimension. For example, if the HDR policy is not specified, the Content Key Context may include both HDR and non-HDR media.

6.4.5. Content Key Delivery with DRM Signaling

This use case is an extension of [§ 6.4.2 Content Key Delivery to One Entity](#) and is compatible with [§ 6.4.4 Content Key Delivery with Usage Rules](#).

This use case adds DRM Signaling information to each Content Key. The recipient may embed this signaling into the data streams it generates.

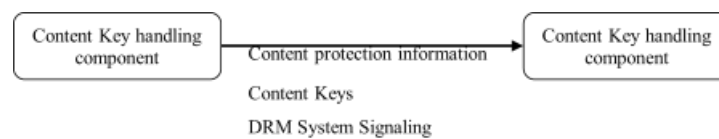


Figure 7 Content Key Delivery with DRM Signaling.

The primary data model carried by content protection information document needs then to include zero to many DRM system signaling elements, each element consisting of a DRM system ID, some signaling information such as for example signaling data for a DASH manifest or an HLS playlist or signaling data for an ISOBMFF file.

The use of 3rd party extensions enable the inclusion of DRM system signaling in forms suitable for other media delivery technologies.

The recipient may use the part of signaling data that it understands and knows how to embed into its output, ignoring signaling data that targets other media delivery technologies.

6.4.6. Incremental Update and Extension of the Document

This use case illustrates the usage of the content protection information document in a realistic workflow comprising multiple cooperating components that require a standardized data format for content protection information exchange.

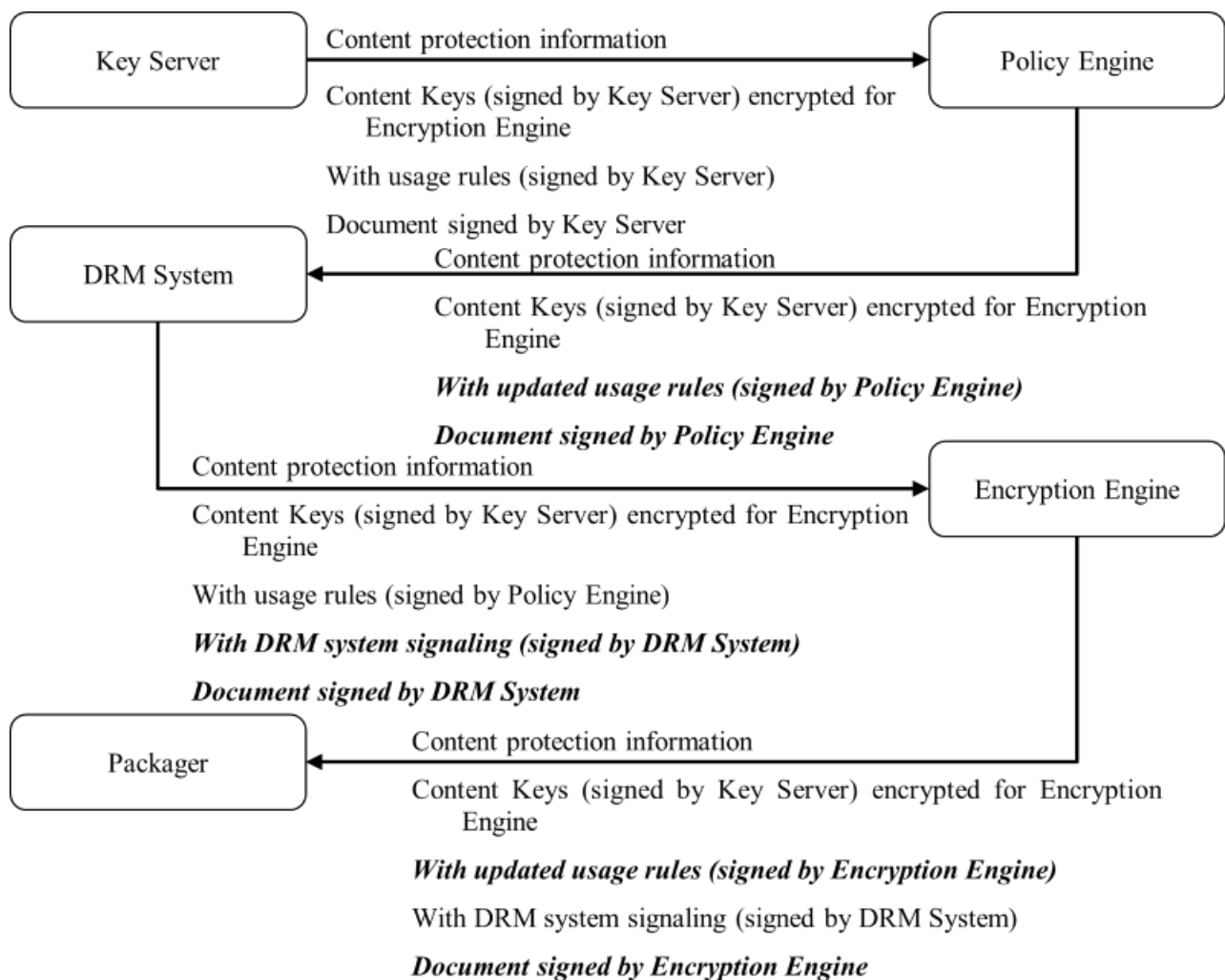


Figure 8 Incremental update and extension of the document.

Each component participating in such a workflow is the authority on a particular aspect. For example, the Key Server manages Content Keys and usage rules and may define the key periods, the DRM System knows how to define the correct DRM Signaling and the Encryption Engine might want to inform the Packager what representations the Content Keys actually got mapped to (the Packager might not have enough information to resolve usage rules based on detailed metadata, so the Encryption Engine could define a new set of usage rules that are simple enough for the Packager to understand, e.g. by making use of label filters).

As the document travels in the workflow, each component adds the elements containing the content protection items it generates (key periods, usage rules, Content Keys, DRM Signaling, etc), making it suitable for the next component that will make use of it. After each modification, the added elements may be signed to maintain a chain of trust on each set of elements individually. The document in its entirety may also be signed to authenticate the document as a whole.

Note that in the above example, the Content Key material itself is encrypted for the Encryption Engine. Despite the fact that many other components participate in the workflow, they do not have access to Content Keys.

6.4.7. Content Key Hierarchy Delivery for Content Packaging

Some DRM systems enable the use of hierarchy of keys, where the set of keys delivered to clients (root keys) within licenses differs from the set of keys used to encrypt Content (leaf keys). Doing so enables DRM systems to separate content encryption and commercial offer management.

Packaging content that uses a key hierarchy requires the Packager to know:

- The leaf keys.

- The KIDs of the root keys (but not the root keys themselves).
- DRM system signaling data for both root and leaf keys.

To fulfill this use case, CPIX enables the above data to be exchanged.

6.4.8. Root Key Delivery for License Server Operation

Some DRM systems enable the use of hierarchical keys, where the set of keys delivered to clients (root keys) differs from the set of keys used to encrypt Content (leaf keys).

When, for example, key creation is not a function of the license server, creating licenses in scenarios that use hierarchical keys requires the license server to know the root keys. CPIX enables root keys to be delivered to license servers.

The exchange of root keys is technically identical to the exchange of non-hierarchical Content Keys as described in [§ 6.4.2 Content Key Delivery to One Entity](#). It is expected that the recipient of a CPIX document in this use case is already aware of the hierarchical nature of the keys within, without any signaling in the CPIX document.

6.5. Workflow Examples

6.5.1. Encryptor Producer and Encryptor Consumer

6.5.1.1. Introduction

There are many workflows that are possible, depending on which entities provide information in the CPIX document, and which entities consume that information. Two simple single-producer, single-consumer examples are illustrated below:

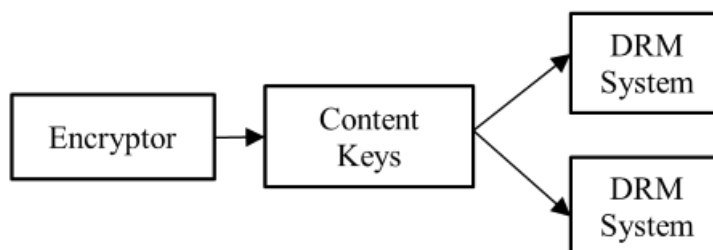


Figure 9 Encryptor Producer.

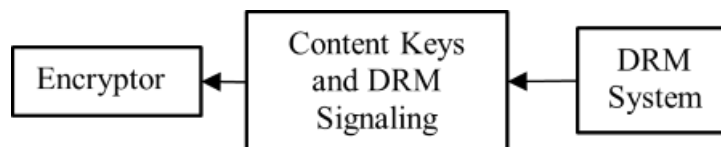


Figure 10 Encryptor Consumer.

All workflows require that content protection information and Content Keys be exchanged between two or more entities. In the examples above the entities are the Encryptor and DRM System:

- The Encryptor Producer example allows, in this case, the Encryptor to generate Content Keys and to push them to one or many DRM systems. The Encryptor could expect to receive from the DRM systems some DRM Signaling.
- The Encryptor Consumer example allows the Encryptor to pull Content Keys and DRM Signaling from a DRM system. In this case, Content Keys are generated by the DRM System.

The document allows supporting both workflows above in addition to other workflows not explicitly described here.

Before exchanging key information in a secure manner, the entities which exchange key material must know about each other and share public keys so that one entity could encrypt data and the other entity could decrypt it. This important step of Trust establishment is out of the scope of this document.

6.5.1.2. *Encryptor Producer*

This clause shows a possible workflow for securing the exchange of the key information between entities when the Encryptor generates the Content Keys. In this example, the Encryptor is the entity which is taking responsibility for generating the Content Keys, protecting them and pushing them to the DRM Systems.

- The first step is the Trust establishment. Public keys shall be exchanged between two or more entities (the Encryptors and the DRM Systems) prior exchanges.
- Once the Trust is established and the necessary associated key material is shared between entities, Content Keys can be exchanged. The Encryptor is encrypting these keys using DRM Systems public keys. The DRM Systems can decrypt using their own private key.
- The Encryptor provides crypto material required to uniquely identify the entity capable of decrypting the media.

All these steps are summarized in the figure below.

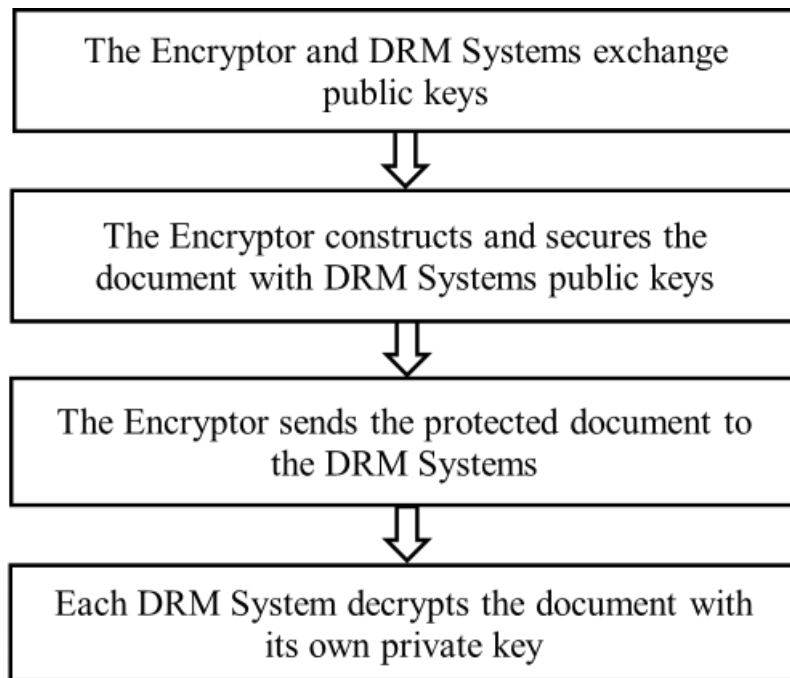


Figure 11 *Encryptor Producer example steps.*

6.5.1.3. *Encryptor Consumer*

This clause shows a possible workflow for securing the exchange of the key information between entities when the DRM System generates the Content Keys. In this model, the Encryptor can pull documents directly from a DRM System. In this case, the DRM System is generating Content Keys and is encrypting them for a secure delivery to the Encryptor.

- As in the case of the Encryptor Producer model, the first step is the Trust establishment. Public keys shall be exchanged between two or more entities (the Encryptors and the DRM System) prior exchanges.
- The DRM System will use the public key of the Encryptor to encrypt keys to be inserted in the document and will send it to Encryptor.
- The Encryptor can decrypt the Content Keys using its private key.

All these steps are summarized in the figure below.

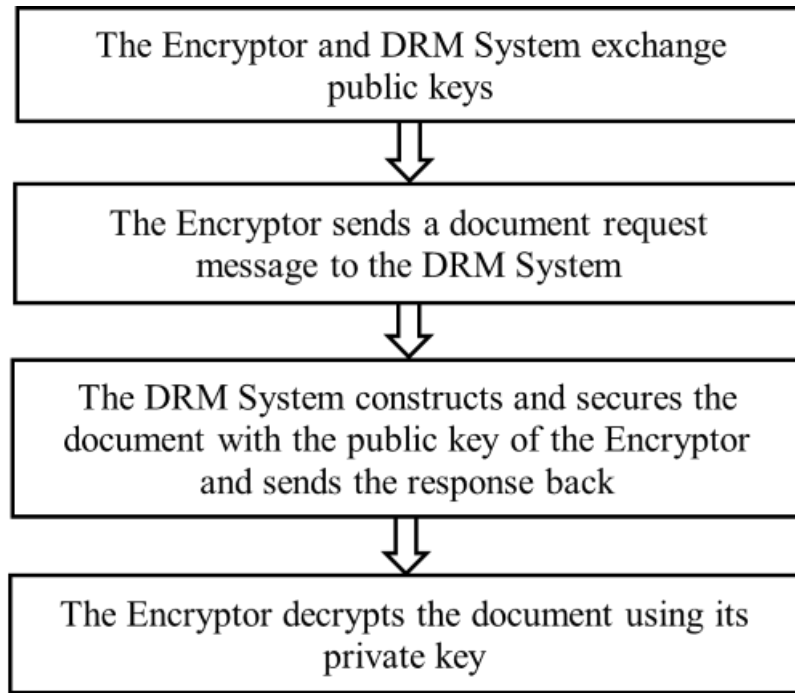


Figure 12 Encryptor Consumer example steps.

6.5.2. Multiple Producers

This clause illustrates that it is possible to have more complex workflows than those previously illustrated. In one such example, for DASH content, a media packager might define the types of streams in the presentation, an Encryptor might generate the Content Keys, a DRM System might generate other DRM Signaling, An Encryptor and an MPD Generator might be the consumers of the final document. In such workflows, the document gets passed from entity to entity in sequence, with each entity adding top-level elements, and recording the update.

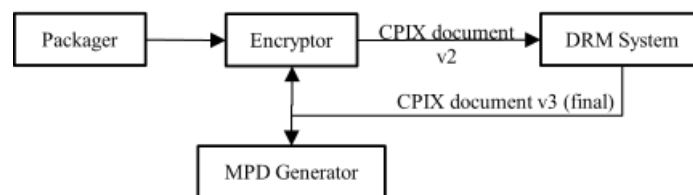


Figure 13 Multiple Producers example.

- The first step is the Trust establishment. Public keys shall be exchanged between two or more entities prior to exchanges.
- Once the Trust is established and the necessary associated key material is shared between entities, Content Keys can be exchanged.
- The Packager provides identification of the receivers and the various stream encoding criteria (usage rules) in version 1 of the document.
- The Encryptor adds key information in version 2 of the document. These elements only contain Keys and no DRM information.
- The DRM System imports the Content Keys stored in the document, and adds its own information in version 3 of the document, which is the finalized version.
- The Encryptor extracts content protection information from the document to be embedded in the media (e.g. pssh boxes).

- The MPD Generator also extracts content protection related information from the document to be embedded in the MPD document (e.g. pssh boxes, key IDs).

All these steps are summarized in the figure below.

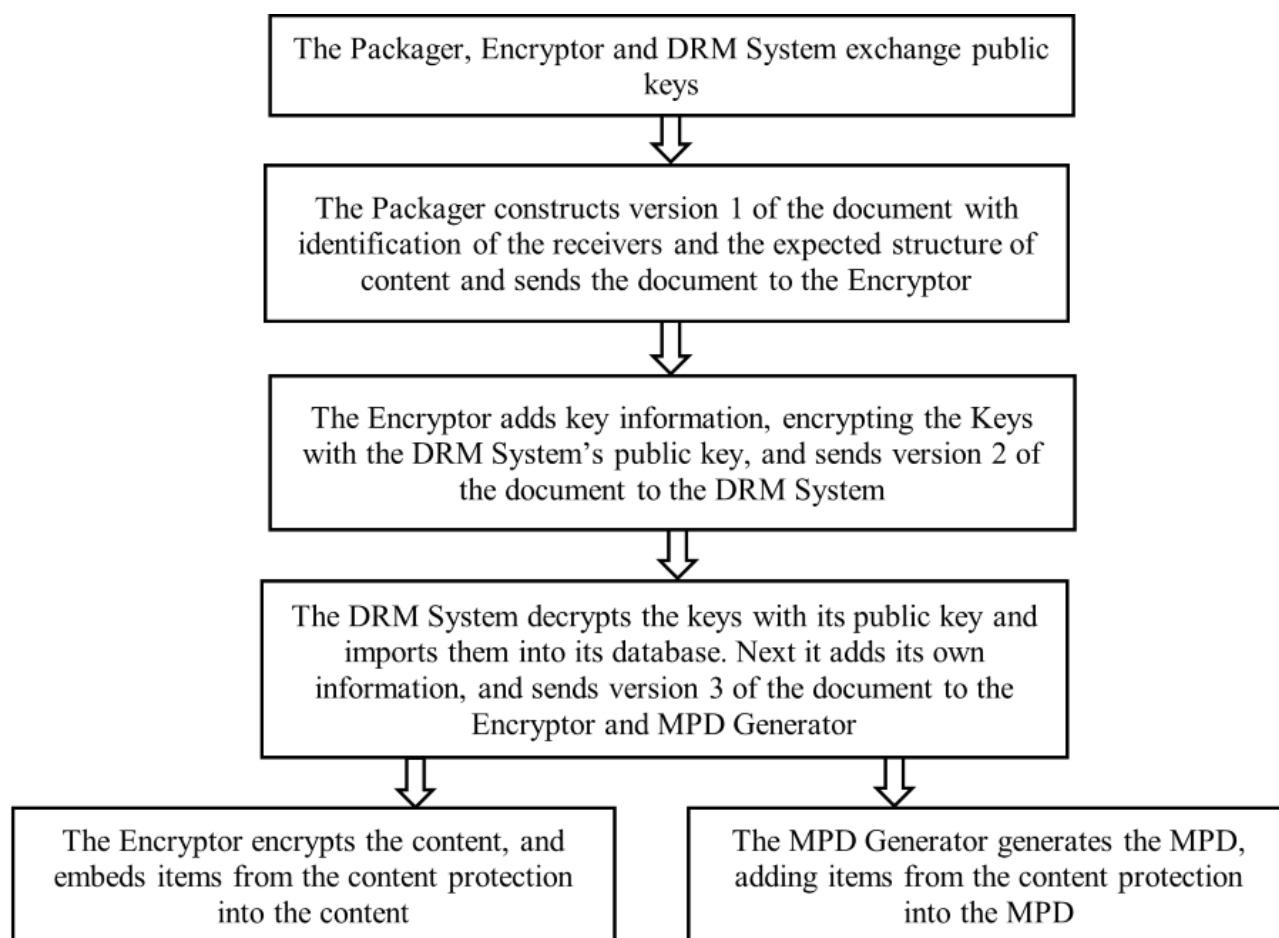


Figure 14 Multiple Producers example steps.

7. XSD Schema Definition§

7.1. Introduction§

This clause describes the Content Protection Information eXchange (CPIX) format to provide a framework to securely exchange Content Key(s) and DRM Signaling between different system entities (see clause [§6 Use Cases and Requirements](#)). This is an XML file that is described by the XSD available [here](#). This clause describes in details elements part of the schema.

7.2. Requirements§

It shall be possible to exchange Content Key(s) and DRM Signaling between entities involved in Content preparation workflows, an example of such interface where the exchange shall be possible is between a DRM system and the encryption engine.

It shall be possible that the manifest generator receives DRM Signaling for several DRM systems and/or content formats

Update of Content Key(s) shall be possible at periodic time or based on events. Some period of time could be in the clear (no encryption).

It shall allow generating MPD conformant to [\[DASHIF-IOPv5p6\]](#).

Content Key(s) shall be secured over the interface.

Entities exchanging content protection information should be authenticated.

7.3. Structure Overview

The structure is articulated around Content Keys and the accompanying material. The document contains all the information required for allowing any entitled entity to get access to or add in the Content Keys and either consume or add material, such as time constraint, DRM information to the CPIX document. The same XML file can be shared between several receiving entities. Hence, each one shall be able to decrypt keys and shall be properly identified.

Taking this into account, the CPIX document contains lists of elements:

- `DeliveryDataList`: This list contains instances of `DeliveryData`, each of which describes an entity entitled to decrypt Content Keys contained in the CPIX document.
- `ContentKeyList`: This list contains instances of `ContentKey`, each of which contains a Content Key used for encrypting media.
- `DRMSystemList`: This list contains instances of `DRMSystem`, each of which contains the signaling data to associate one DRM system with one Content Key.
- `ContentKeyPeriodList`: This list contains instances of `ContentKeyPeriod`, each of which defines a time period that may be referenced by the key period filters included in Content Key usage rules.
- `ContentKeyUsageRuleList`: This list contains instances of `ContentKeyUsageRule`, which maps a Content Key to one or more Content Key Contexts.
- `UpdateHistoryItemList`: This list contains instances of `UpdateHistoryItem`, each of which contains an update version number and an identifier of the entity which produced the update. Other elements in the document are linked to a specific update by update version number (via the `@updateVersion` attribute).
- `Signature`: Each instance of this element contains a digital signature [\[XMLDSIG-CORE\]](#) over either the entire document or a subset of XML elements.

The Content Keys can be encrypted inside the XML file using the public keys of the recipients, identified in the `DeliveryData` elements. The XML file also allows storing the Content Keys in the clear, in which case the security of the Content Keys is contingent on the security of the communication channel used to deliver the CPIX document to the recipients.

The figure below shows the first elements and a high-level view of the structure. Detailed description of the structure is given in the following clauses.

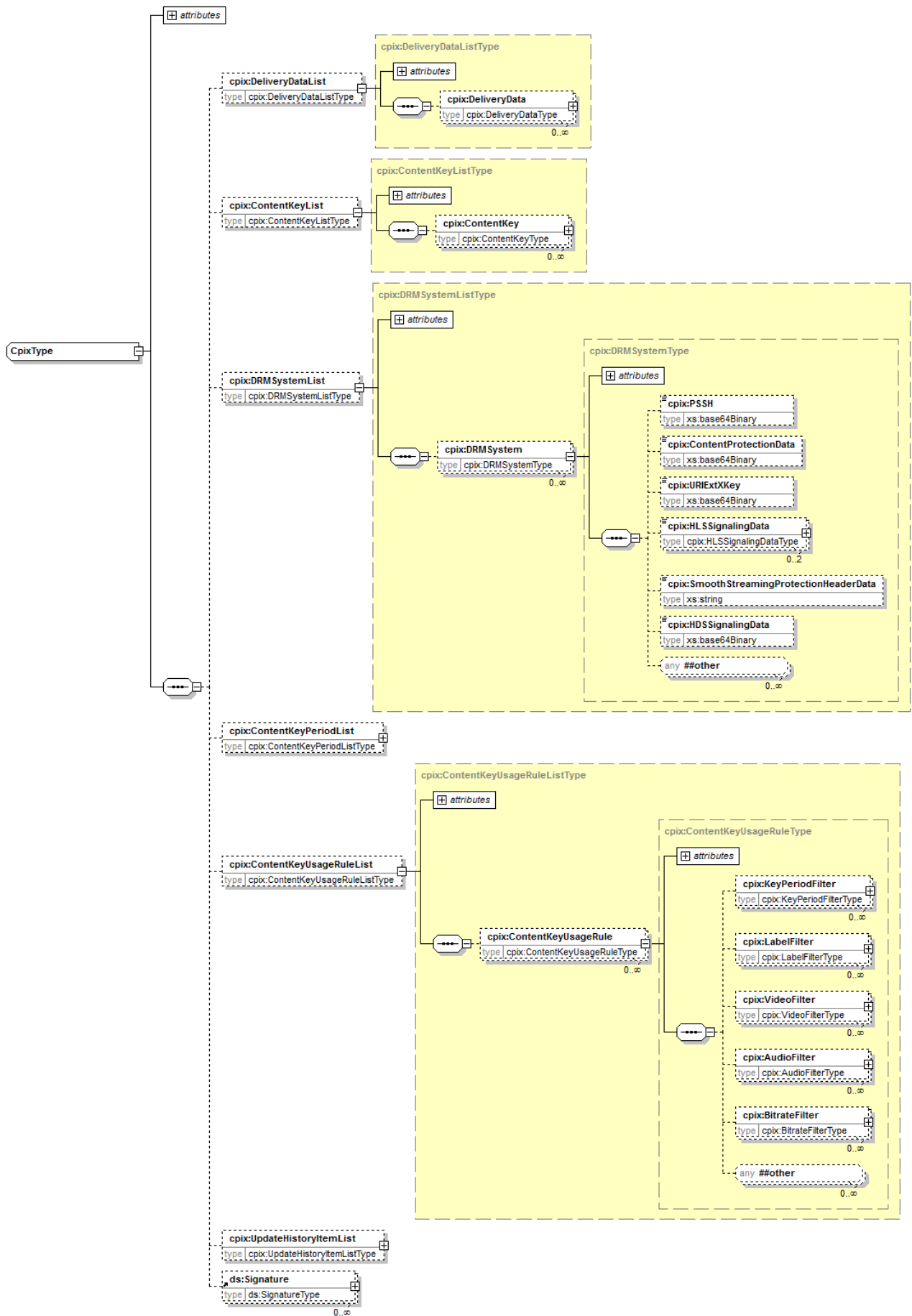


Figure 15 Content Protection Information Exchange Format high level view.

7.4. Hierarchical Data Model

7.4.1. Introduction

In this clause, the following conventions are used:

- Element names are in PascalCase and the number of allowed instances is defined by (min..max) where N for max means unbounded.
- Attribute names are in camelCase preceded with an @ and the use of an attribute is defined as: M=Mandatory, O=Optional, OD=Optional with Default Value, CM=Conditionally Mandatory. Attributes may be in any order.

The XSD schema for this model is provided [here](#). In addition to types defined in this document, the CPIX data model references types defined in [\[XMLSCHEMA11-2\]](#), [\[RFC6030\]](#), [\[XMLDSIG-CORE\]](#) and [\[XMLENC-CORE\]](#). External data types are prefixed with xs:, pskc:, ds: and xenc: respectively.

7.4.2. CPIX Element

The root element that carries the Content Protection Information for a set of media assets.

@id (O, xs:ID)

An identifier for the CPIX document. It is recommended to use an identifier that is unique within the scope in which this file is published.

@contentId (O, xs:string)

An identifier for the asset or content that is been protected by the keys carried in this CPIX document. It is recommended to use an identifier that is unique within the scope in which this file is published.

@name (O, xs:string)

A name for the presentation.

@version (O, xs:string)

A version for the CPIX document. The value shall reference a published version of the CPIX guidelines and be structured as majorVersion.minorVersion. This specification describes version 2.3.

If the CPIX client knows that it doesn't support all the features of a given CPIX version, it needs to behave according to the recommendations of the API used to exchange the CPIX document.

DeliveryDataList (0..1, <DeliveryDataList>)

A container for <DeliveryData> elements. If not present, Content Keys in the document are delivered in the clear, without encryption.

ContentKeyList (0..1, <ContentKeyList>)

A container for <ContentKey> elements.

DRMSystemList (0..1, <DRMSystemList>)

A container for <DRMSystem> elements. If not present, the document does not contain any DRM system signaling data.

ContentKeyPeriodList (0..1, <ContentKeyPeriodList>)

A container for <ContentKeyPeriod> elements.

ContentKeyUsageRuleList (0..1, <ContentKeyUsageRuleList>)

A container for <ContentKeyUsageRule> elements. If not present, the document does not define Content Key Contexts and an external mechanism is required for synchronizing the content creation workflow.

UpdateHistoryItemList (0..1, <UpdateHistoryItemList>)

A container for <UpdateHistoryItem> elements.

Signature (0..N, ds:Signature)

Digital signatures as defined in [\[XMLDSIG-CORE\]](#). Each signature signs either the full document or any set of elements within the CPIX document. Every digital signature shall contain an X.509 certificate identifying the signer and the associated public key.

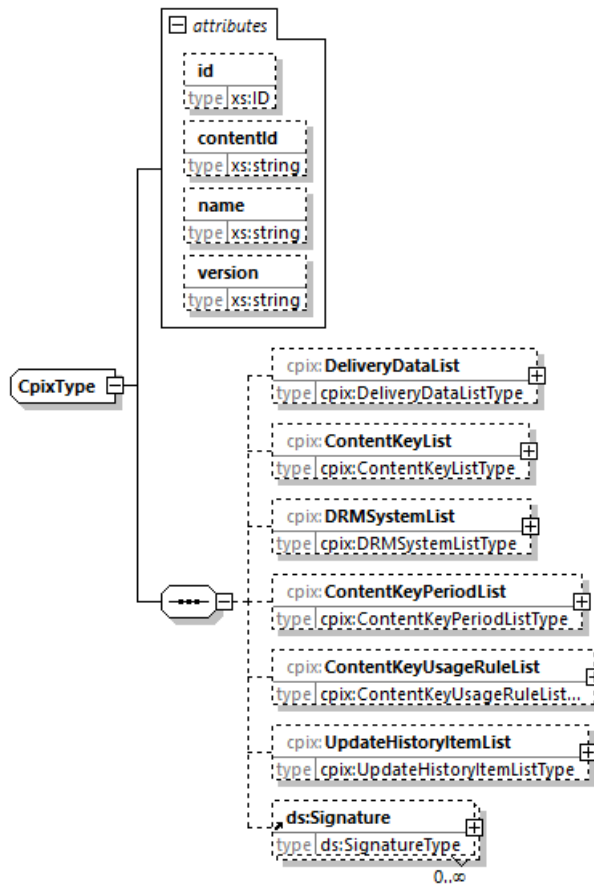


Figure 16 CPIX element.

7.4.3. DeliveryDataList Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the @updateVersion attribute of the <UpdateHistoryItem> element providing details on when this element was added or updated.

DeliveryData (0..N, <DeliveryData>)

Contains the required information allowing defining which entities can get access to the Content Keys delivered in this document.

There is one <DeliveryData> element per entity capable of accessing encrypted Content Keys stored in this document. If this element is not present, then the Content Keys are in the clear in the file.

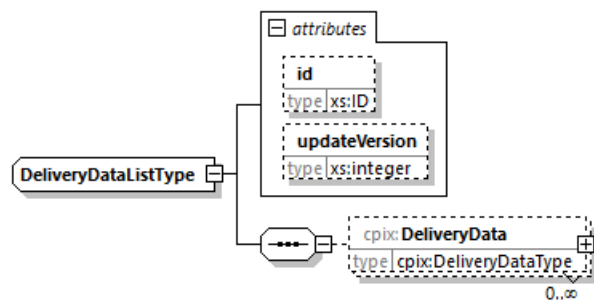


Figure 17 DeliveryDataList element.

7.4.4. DeliveryData Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the @updateVersion attribute of the [<UpdateHistoryItem>](#) element providing details on when this element was added or updated.

@name (O, xs:string)

Name of the Delivery Data.

DeliveryKey (1, ds:KeyInfoType)

Contains an X.509 certificate that identifies the intended recipient and the public key that was used to encrypt the Document Key.

Refer to [§ 8.1 Key Encryption in the CPIX Document](#) for a description of the key management within the CPIX document.

DocumentKey (1, cpix:KeyType)

Contains the key that was used for encrypting the Content Key stored in each [<ContentKey>](#) element. The Document Key is encrypted using the public key listed in the recipient's X.509 certificate.

This is an extension of KeyType defined in [\[RFC6030\]](#). The attributes @id and @Algorithm are optional in this extension.

Refer to [§ 8.1 Key Encryption in the CPIX Document](#) for a description of the key management within the CPIX document.

MACMethod (0..1, pskc:MACMethodType)

Identifies the MAC algorithm and contains the MAC key used to implement authenticated encryption of Content Keys. The MAC key is encrypted using the public key listed in the recipient's X.509 certificate.

Refer to [§ 8.1 Key Encryption in the CPIX Document](#) for a description of the key management within the CPIX document.

Description (0..1, xs:string)

A description of the element.

SendingEntity (0..1, xs:string)

The name of the entity generating this CPIX document.

SenderPointOfContact (0..1, xs:string)

The contact information, such as an email address, of the entity generating this CPIX document.

ReceivingEntity (0..1, xs:string)

The name of the entity capable of decrypting Content Keys in this CPIX document.

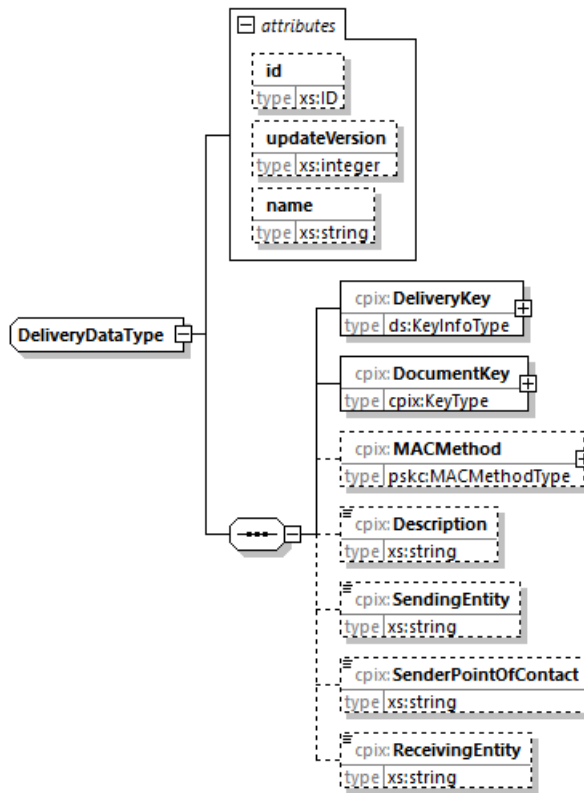


Figure 18 DeliveryData element.

7.4.5. ContentKeyList Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the @updateVersion attribute of the <UpdateHistoryItem> element providing details on when this element was added or updated.

ContentKey (0...N, <ContentKey>)

Contains all information on a Content Key used to encrypt one or more Content Key Contexts.

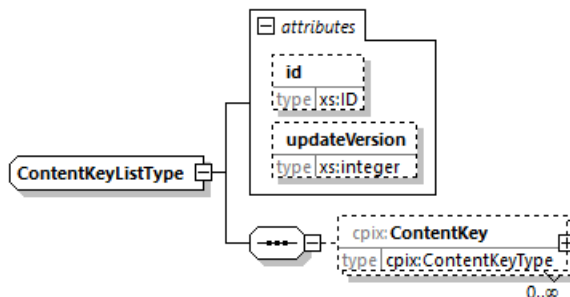


Figure 19 ContentKeyList element.

7.4.6. ContentKey Element

This is an extension of KeyType defined in [RFC6030]. The attributes @id and @Algorithm are optional in this extension.

The key this element contains can be encrypted. If it is encrypted, it is encrypted with the key that is under the DocumentKey element part of the <DeliveryData>. Refer to § 8.1 Key Encryption in the CPIX Document for a description of the key management within the CPIX document.

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@Algorithm (O, xs:ID)

This is an attribute inherited from [\[RFC6030\]](#) and made optional in this specification.

@kid (M, cpix:UUIDType)

The unique identifier of the Content Key. It shall be formatted as defined in [\[MPEGCENC\]](#), clause 11.2.

@explicitIV (O, xs:base64binary)

The IV associated with the Content Key. This is a 128-bit value in binary format, base64-encoded. This is the value of Constant IV defined in [\[MPEGCENC\]](#) or of the IV attribute under #EXT-X-KEY defined in [\[HLS\]](#).

Use of this attribute is not recommended except for compatibility with some DRM systems that explicitly need it.

@dependsOnKey (O, cpix:UUIDType)

This attribute signals that the Content Key is a leaf key in a key hierarchy. It references the @kid attribute of another [<ContentKey>](#) element describing the root key.

The referenced key shall not be a leaf key.

If this attribute is not specified, the Content Key is either a root key or does not participate in a key hierarchy.

The CPIX document format does not make a distinction between these two cases.

@commonEncryptionScheme (O, xs:string)

The encryption scheme that the content key is intended to be used with. When present, the value shall be a 4-character Common Encryption protection scheme name as defined by [\[MPEGCENC\]](#) or one of the encryption method defined in [\[HLS\]](#). If the attribute is omitted then content may be encrypted using any encryption scheme.

This attribute shall not be used if the @dependsOnKey attribute is present. In a key hierarchy, the root key defines the encryption scheme to use with all keys in the hierarchy.

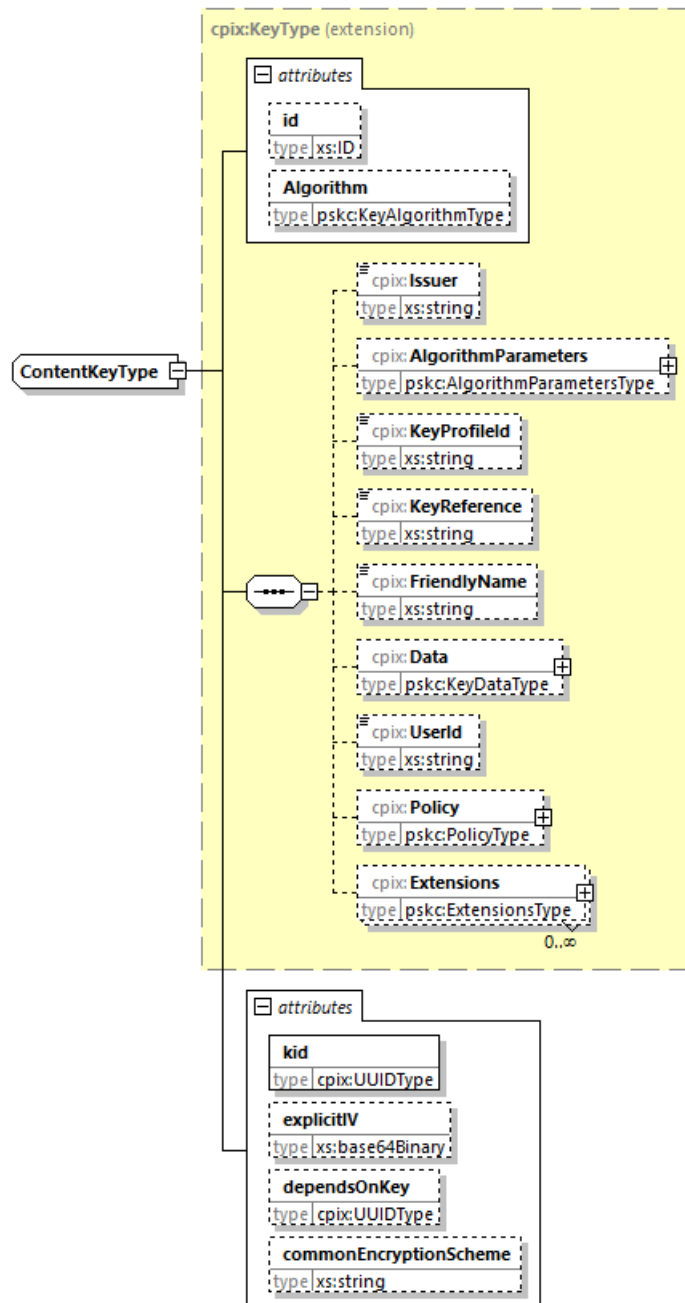


Figure 20 ContentKey element.

7.4.7. DRMSystemList Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the @updateVersion attribute of the [UpdateHistoryItem](#) element providing details on when this element was added or updated.

DRMSystem (0..N, <DRMSystem>)

DRM Signaling of a DRM system associated with a Content Key.

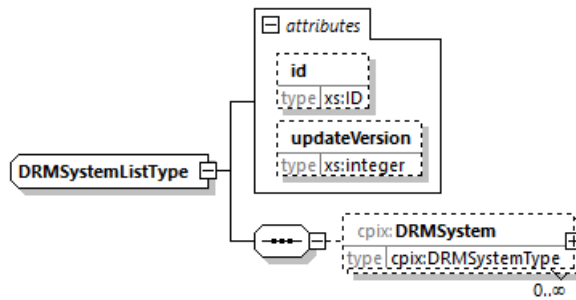


Figure 21 DRMSystemList element.

7.4.8. DRMSystem Element

This element contains all information on a DRM system that can be used for retrieving licenses for getting access to content. This specification defines elements for DRM system signaling in DASH, ISO BMFF, Smooth Streaming, HLS and HDS formats. Implementations may extend CPIX documents with additional elements to provide DRM system signaling information for other formats.

The DRM system signaling data in `<DRMSystem>` elements often contains the protection scheme identifier in a DRM or streaming protocol system specific format. Values in `<DRMSystem>` elements shall be aligned with the values in `@commonEncryptionScheme` attributes of the `<ContentKey>` elements.

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the `@updateVersion` attribute of the `<UpdateHistoryItem>` element providing details on when this element was added or updated.

@systemId (M, cpiX:UUIDType)

This is the unique identifier of the DRM system. Values are available on dashif.org.

@kid (M, cpiX:UUIDType)

Matches the `@kid` attribute of the `<ContentKey>` this element references.

@name (O, xs:string)

This is a human-readable name and version of the DRM system. This can be used in a MPD as the value for the `@value` attribute of the `ContentProtection` element.

PSSH (0..1, xs:base64binary)

This is the full pssh box that should be added to ISO BMFF files encrypted with the referenced Content Key.

If the referenced `<ContentKey>` element by `@kid` includes a `@dependsOnKey` attribute, the value shall be inserted under the `moof` box.

If the referenced `<ContentKey>` element by `@kid` does not include a `@dependsOnKey` attribute, the value may be inserted under the `moov` box or the `moof` box. In this case, see [\[DASHIF-IOPv5p6\]](#), clause 6, if this element is used.

ContentProtectionData (0..1, xs:base64binary)

This is the full well-formed standalone XML fragment to be added to the DASH manifest under the `ContentProtection` element for this DRM system. This is UTF-8 text without a byte order mark. An example of such data is the W3C signaling defined in [\[DASHIF-IOPv5p6\]](#), in this case, all `dashif:xxx` elements are children of the `ContentProtection` element and are therefore provided in this element.

If the referenced `<ContentKey>` element by `@kid` includes a `@dependsOnKey` attribute, this element shall not be used.

If the referenced `<ContentKey>` element by `@kid` does not include a `@dependsOnKey` attribute, the value may be added under the `ContentProtection` element for this DRM system.

This element has meaning only when a DASH manifest is created for the media content.

URIExtXKey (0..1, xs:base64binary)

This is the full data to be added in the URI parameter of the #EXT-X-KEY tag of a HLS playlist. This is UTF-8 text without a byte order mark.

This element is present only when the content is in the HLS format.

The use of this element is deprecated. Using [<HLSSignalingData>](#) is recommended.

HLSSignalingData (0..2, <HLSSignalingData>)

This is the full data including the #EXT-X-KEY or #EXT-X-SESSION-KEY tag of a HLS playlist [HLS] depending on the destination of the data (see § 7.4.9 HLSSignalingData Element). This may contain multiple lines allowing to add lines with proprietary tags and values. This is UTF-8 text without a byte order mark.

This element shall not be used if the referenced Content Key is a leaf key in a key hierarchy.

This element has meaning only when a HLS playlist is created for the media content.

SmoothStreamingProtectionHeaderData (0..1, xs:string)

This is the inner text of the ProtectionHeader XML element to be added to the Smooth Streaming manifest for this DRM system. This is UTF-8 text without a byte order mark.

This element shall not be used if the referenced Content Key is a leaf key in a key hierarchy.

This element has meaning only when a Smooth Streaming manifest is created for the media content.

HDSSignalingData (0..1, xs:base64binary)

This is the full drmAdditionalHeader element to be added to the HDS playlist (including the beginning and ending tags). This is UTF-8 text without a byte order mark.

This element shall not be used if the referenced Content Key is a leaf key in a key hierarchy.

This element has meaning only when a Flash media manifest is created for the media content.

Additional child elements not defined by DASH-IF may be present containing signaling data for other media formats. Such elements shall appear after any elements defined here.

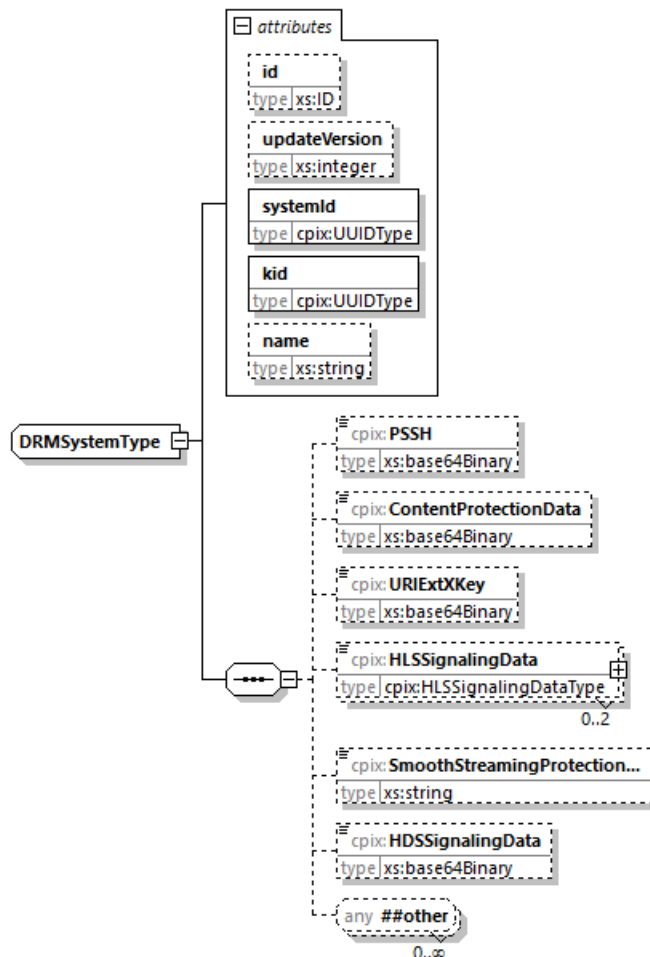


Figure 22 DRMSystem element.

7.4.9. HLSSignalingData Element

The HLSSignalingData shall be base64 encoded text. It has an optional attribute allowing to define where this data is to be placed, either in the master playlist or in the media playlist. It allows having different proprietary signaling in these locations. In a <DRMSystem> element, every HLSSignalingData shall have a different @playlist value if present. If @playlist is not present then the HLSSignalingData goes in the media playlist and there is no signaling in the master playlist (in this case, there is only one HLSSignalingData element in the <DRMSystem> element).

@playlist (O, restricted xs:string)

Specifies the destination of the data carried by this element. It can only have two values master and media.

There is a uniqueness rule for this attribute. If two elements are added under a <DRMSystem> element, they shall not have the same @playlist value.

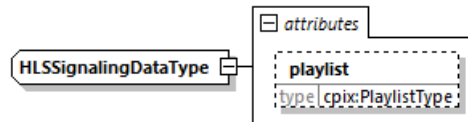


Figure 23 HLSSignalingData element.

7.4.10. ContentKeyPeriodList Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the @updateVersion attribute of the <UpdateHistoryItem> element providing details on when this element was added or updated.

ContentKeyPeriod (0..N, <ContentKeyPeriod>)

For every Content Key, <ContentKeyPeriod> elements cover non overlapping periods of time. The concatenation of all period of times may not fully cover the Content as some parts may be in the clear.

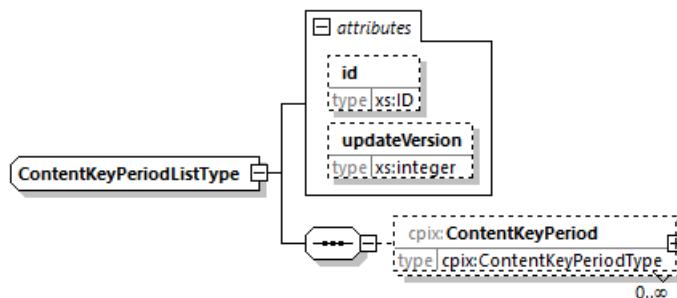


Figure 24 ContentKeyPeriodList element.

7.4.11. ContentKeyPeriod Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@index (O, xs:integer)

Numerical index for the key period. Mutually exclusive with @start and @end.

@start (O, xs:dateTime)

Wall clock (Live) or media time (VOD) for the start time for the period. Mutually exclusive with @index.

@end (O, xs:dateTime)

Wall clock (Live) or media time (VOD) for the end time for the period. Mutually exclusive with @index.

When @start and @end are present, the interval is defined by [@start, @end), meaning that the key is in use at the start time but not at the end time. If neither are specified, then it is assumed that the encryptor is determining the key period boundaries internally, and other components do not need to be aware of them. In this case, the key periods are referenced simply by a sequence number (@index). An example of this would be an encryptor which rotates the keys once an hour, and not necessarily at specific times.

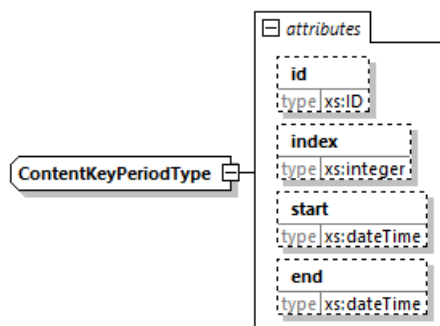


Figure 25 ContentKeyPeriod element.

7.4.12. ContentKeyUsageRuleList Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (O, xs:integer)

Matches the @updateVersion attribute of the <UpdateHistoryItem> element providing details on when this element was added or updated.

ContentKeyUsageRule (0..N, <ContentKeyUsageRule>)

A rule which defines a Content Key Context.

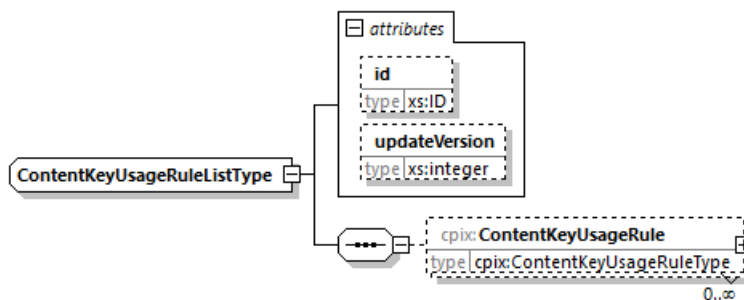


Figure 26 ContentKeyUsageRuleList element.

7.4.13. ContentKeyUsageRule Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@kid (M, cpix:UIDType)

Matches the @kid attribute of the <ContentKey> this element references.

In hierarchical key scenarios, this shall reference a leaf key, not a root key.

@intendedTrackType (O, xs:string)

Specifies the type of media track which corresponds to the streams which match the rules defined in this element.

Examples of types for the media track might be UHD, UHD+HFR. See § 7.4.14.3 LabelFilter Element for more details.

KeyPeriodFilter (0..N, <KeyPeriodFilter>)

Defines a period of time constraints for the Content Key Context.

This filter links <ContentKey> and <ContentKeyPeriod> elements.

LabelFilter (0..N, <LabelFilter>)

Defines a label association for the Content Key Context.

VideoFilter (0..N, <VideoFilter>)

Defines video constraints to be associated with the Content Key Context.

This filter can only be used on media content of type video.

AudioFilter (0..N, <AudioFilter>)

Defines audio constraints to be associated with the Content Key Context.

This filter can only be used on media content of type audio.

BitrateFilter (0..N, <BitrateFilter>)

Defines bitrate constraints to be associated with the Content Key Context.

Additional child elements not defined by DASH-IF may be present containing proprietary filters. Such elements shall appear after any elements defined here.

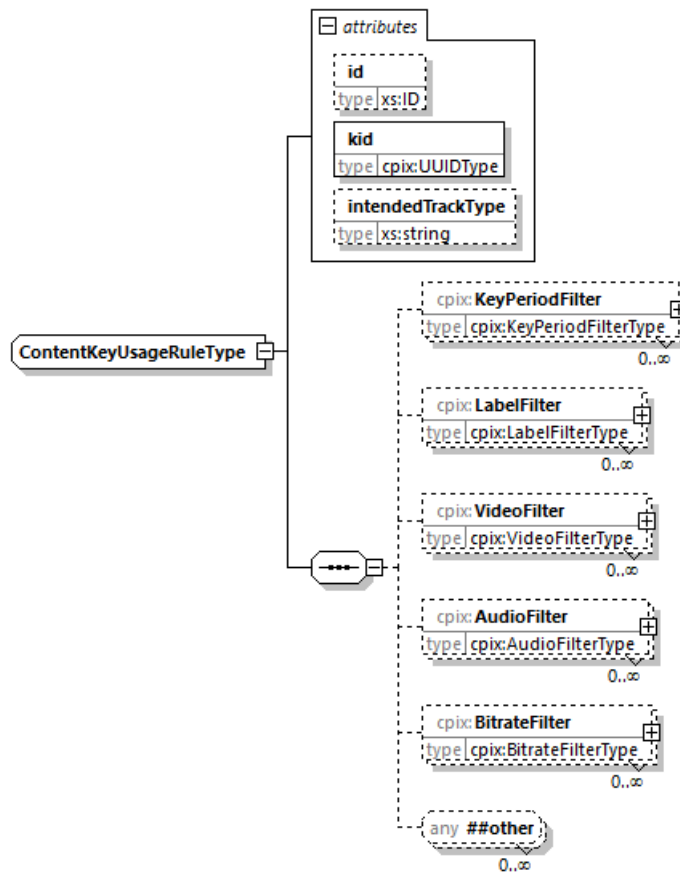


Figure 27 ContentKeyUsageRule element.

7.4.14. Usage Rules Filters

7.4.14.1. Introduction

There can be several filters defined within a single <ContentKeyUsageRule>. In this case, all rules apply identically, the entity generating the <ContentKeyUsageRule> element or adding a new rule is responsible for ensuring that they do not contradict each other. A set of rules that would match multiple Content Keys to a single Content Key Context is invalid.

If more than one of a particular type of filter (e.g. [<KeyPeriodFilter>](#)) is present within a [<ContentKeyUsageRule>](#), then they are first aggregated with a logical OR operator. After that, different types of filters are aggregated with a logical AND operator. For example, a rule that defines a label filter for stream-1, a label filter for steam-2 and a video filter would be matched as (stream-1 OR stream-2) AND video.

A scenario where multiple Content Keys can be mapped to a single Content Key Context shall be considered invalid. A CPIX document shall always match exactly zero or one Content Keys to any Content Key Context.

A usage rule shall be considered unusable if it contains a child element whose meaning is unknown (i.e. a filter of an unknown type) or which cannot be processed for any other reason (e.g. @minPixels in the [<VideoFilter>](#) element is defined but the implementation does not know the pixel count of the video samples). An entity interpreting the [<ContentKeyUsageRule>](#) element shall not perform Content Key(s) mapping to Content Key Contexts if any unusable usage rules exist. An entity that is not interpreting the [<ContentKeyUsageRule>](#) element (doing, for example, only storage of the CPIX document for latter distribution to another entity) can perform any processing on the document.

Processing of the Content Key(s) referenced by any unusable usage rules shall not be performed. The usable part of the document can be processed normally.

There can be many different sources for defining usage rules, for example, they can be the result of a right holder requirement or a decision to encrypt separately SD, HD and UHD tracks. The CPIX document does not keep track of the source of these rules, it only defines how to maps Content Keys to tracks.

7.4.14.2. KeyPeriodFilter Element

@periodId (M, xs:IDREF)

This references a [<ContentKeyPeriod>](#) element by @id. The filter will only match samples that belong to the referenced key period.

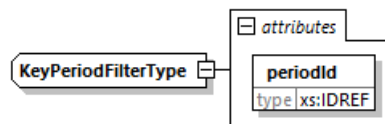


Figure 28 KeyPeriodFilter element.

7.4.14.3. LabelFilter Element

@label (M, xs:string)

The filter will only match samples that carry a matching label. The exact meaning of labels is implementation-defined and shall be agreed upon in advance by the producer and consumer of the CPIX document.

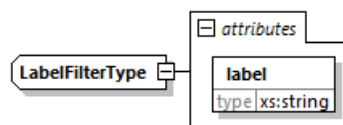


Figure 29 LabelFilter element.

The @label attribute is meant for triggering a particular [<ContentKeyUsageRule>](#) by using pre-agreed upon label strings. Its value may or may not correspond to media track types. One example is a label such as UHD that can be used to match the corresponding [<ContentKeyUsageRule>](#) element when used as an input or selector for a content encryptor, media packager, MPD generator or license service to select a specific Content Key, populate the ContentProtection element, or include the corresponding key in a content license. Another example is if there is a previous agreement defined outside of a CPIX document that "blue tracks" are encrypted with the Content Key 1234

and "green tracks" are encrypted with the Content Key 5678. The labels can be used in this case to identify the suitable tracks (without expressing the specifics of the agreement itself).

In contrast, the `@intendedTrackType` attribute of [<ContentKeyUsageRule>](#) is used to assign a track type to the media streams which match the filters. The value of the string may not be pre-agreed between the various entities making use of the CPIX document. Said differently, the `@intendedTrackType` attribute is a metadata that states business logic. For example, a rule can be that all low resolutions streams are encrypted with the same Content Key. The value `lowRes` matches this rule. It has no function in defining what Content Key are matched to what tracks, it simply acts as a label to allow business logic to say authorize the use of `lowRes` Content Key and then a CPIX processor can find the rules that matches the right Content Keys for `lowRes` and thereby associated with low resolution tracks.

If a specific key is to be used for more than one type of track (this is not recommended), then there ought to be multiple [<ContentKeyUsageRule>](#) elements, one for each track type, even if they all reference the same Content Key with the same `@kid`.

7.4.14.4. *VideoFilter* Element

If present, even without any attributes, the filter will only match video samples.

@minPixels (OD, xs:integer)

The filter will only match video samples that contain at least this number of pixels (encoded width x height before considering pixel/sample aspect ratio). The default value is 0 (zero).

@maxPixels (OD, xs:integer)

The filter will not match video samples that contain more than this number of pixels (encoded width x height before considering pixel/sample aspect ratio). The default value is `MAX_UINT32`.

@hdr (O, xs:boolean)

Boolean value indicating whether the matching video stream is encoded in HDR.

@wcg (O, xs:boolean)

Boolean value indicating whether the matching video stream is encoded in WCG.

@minFps (O, xs:integer)

Minimum nominal number of frames per second for the video stream. For interlaced video, this is half the number of fields per second.

@maxFps (O, xs:integer)

Maximum nominal number of frames per second for the video stream. For interlaced video, this is half the number of fields per second.

When `@minPixels` and `@maxPixels` are present, the interval is defined by `[@minPixels, @maxPixels]`, meaning that the filter is used for content with video samples that contain `@minPixels` pixels and is used for content with video samples that contain `@maxPixels` pixels.

When `@minFps` and `@maxFps` are present, the interval is defined by `(@minFps, @maxFps]`, meaning that the filter is not used for content with nominal FPS equal to `@minFps` but is used for content with nominal FPS equal to `@maxFps`.

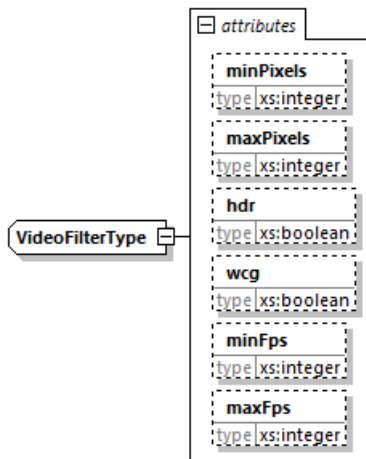


Figure 30 VideoFilter element.

7.4.14.5. AudioFilter Element

If present, even without any attributes, the filter will only match audio samples.

@minChannels (OD, xs:integer)

The filter will only match audio samples that contain at least this number of channels. The default value is 0 (zero).

@maxChannels (OD, xs:integer)

The filter will not match audio samples that contain more than this number of channels. The default value is MAX_UINT32.

When @minChannels and @maxChannels are present, the interval is defined by [@minChannels, @maxChannels], meaning that the filter is used for content with audio samples that have @minChannels audio channels and is used for content with audio samples that have @maxChannels audio channels.

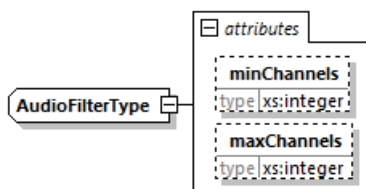


Figure 31 AudioFilter element.

7.4.14.6. BitrateFilter Element

@minBitrate (OD, xs:integer)

The filter will only match samples from streams with a nominal bitrate in b/s of at least this value. The default value is 0 (zero).

@maxBitrate (OD, xs:integer)

The filter will not match samples from streams with a nominal bitrate in b/s that exceeds this value. The default value is MAX_UINT32.

At least one of @minBitrate and @maxBitrate shall be specified.

When @minBitrate and @maxBitrate are present, the interval is defined by [@minBitrate, @maxBitrate], meaning that the filter is used for content with bitrate of @minBitrate and is used for content with bitrate of @maxBitrate.

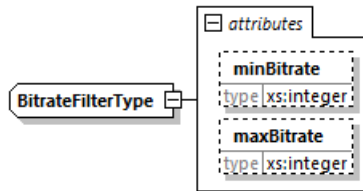


Figure 32 BitrateFilter element.

7.4.15. UpdateHistoryItemList Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

UpdateHistoryItem (0..N, <UpdateHistoryItem>)

It contains metadata about an update made to the CPIX document. There should be one entry for each instance in which an entity updated the document.

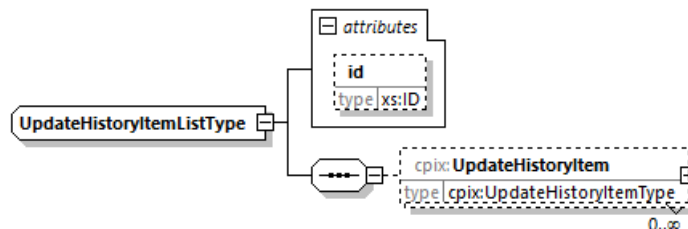


Figure 33 UpdateHistoryItemList element.

7.4.16. UpdateHistoryItem Element

@id (O, xs:ID)

An identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@updateVersion (M, xs:integer)

This is the ID referenced by other elements in the document. It is strongly recommended to use an identifier that is unique within the scope in which this CPIX document is published.

@index (M, xs:string)

This is the version number for the document update. Each <UpdateHistoryItem> element contains a unique value for this attribute. It is a monotonically increasing number, starting at value 1.

@source (M, xs:string)

This is the identifier for the entity which performed the document update.

@date (M, xs:dateTime)

This is the date and time when the document update was performed.

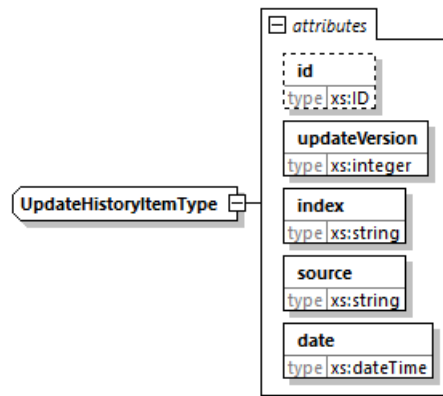


Figure 34 UpdateHistoryItem element.

8. Key Management

8.1. Key Encryption in the CPIX Document

8.1.1. Introduction

The CPIX document allows exchanging Content Keys in the clear but this is not a recommended method as it relies on the security of the communication mechanism used to deliver the CPIX document to the recipients, which may not be sufficient to adequately protect the Content Keys.

Content Keys can be delivered encrypted within the document itself and in this case, a multi-level structure of encryption keys is used for an efficient encryption avoiding duplication of encrypted content and expensive encryption methods. This clause describes the mechanism that shall be used when encryption of the Content Keys in the document is used.

8.1.2. Keys Used to Secure the CPIX Document

The document contains the following keys:

Content Keys

Each [ContentKey](#) element contains one Content Key that is used for encrypting an asset or crypto period of an asset or that acts as a dependency for the use of other Content Keys (when a key hierarchy is used). Typically, for Common Encryption as supported in [DASHIF-IOPv5p6](#), these keys are 128-bit keys used with the AES cipher.

Document Key

For every CPIX document, a Document Key is created. It is used for encrypting every Content Key. The Document Key is a 256-bit key and the encryption algorithm used for encrypting every Content Key is AES. The Document Key is part of each [DeliveryData](#) element. It is itself encrypted in the document, using the public key of each recipient.

Delivery Keys

Each [DeliveryData](#) element identifies a Delivery Key, which is a public key from a key pair owned by the intended recipient. The Delivery Key is identified in the [DeliveryData](#) element by including the X.509 certificate of the intended recipient. The Delivery Key is used for encrypting the Document Key using an algorithm that is described within the CPIX document, according to [XMLENC-CORE](#).

The below figure gives the schema of encryption of the different keys when there are several [DeliveryData](#) elements and several [ContentKey](#) elements. The Document Key allows reducing the numbers of [ContentKey](#) elements as the Content Key they contain are all encrypted by the same Document Key.

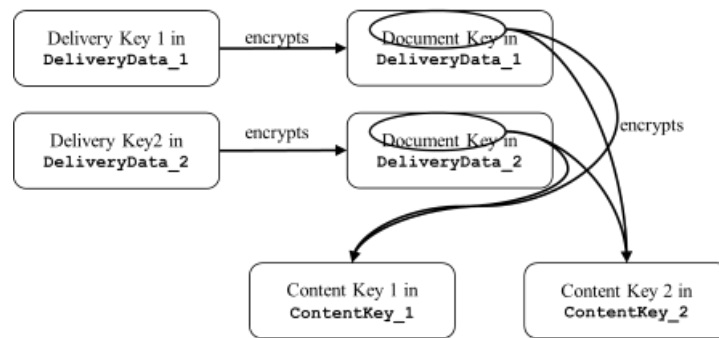


Figure 35 Encryption relationships within the CPIX document. Blue indicates encrypted data.

8.1.3. Authenticated Encryption

MAC Key

For every CPIX document, a MAC Key is created. It is used to calculate the MAC of every encrypted Content Key. The `<DeliveryData>` element identifies the MAC algorithm and provides the MAC Key, encrypted with the Delivery Key, for each recipient.

Authenticated Encryption of Content Keys

Implementations shall provide a MAC for every encrypted Content Key and shall verify the MAC before attempting to decrypt any encrypted Content Key. The purpose of the MAC is to protect against cryptographic vulnerabilities in the receiving application; it is not used as a general purpose authentication mechanism.

The MAC is calculated over the data in the `CipherValue` element (the concatenated IV and encrypted Content Key) and stored in the `ValueMac` element under the `Secret` element for each encrypted Content Key.

8.1.4. Digital Signatures

Every element in the document that has an `@id` attribute can be signed according to [\[XMLDSIG-CORE\]](#). Furthermore, the document (including any other signatures) can be signed as a whole.

Upon loading a CPIX document, implementations should verify that signatures are present on entities that are expected to be signed and verify all digital signatures that are present. Implementations should refuse to process a document if expected signatures are missing or if the signatures cannot be verified or if the signers are not trusted as authoritative sources for the signed data.

Implementations should sign any elements that recipients wish to authenticate. Note that modifying any signed data will require any signatures on the data to be removed and/or re-applied. This requires the appropriate consideration and trust model design in content processing workflow creation (out of scope of this specification).

8.1.5. Mandatory Algorithms

The following table gives the identification of the algorithms that shall be used for encryption, signature, MAC creation.

Usage	Algorithm
Content Key wrapping	AES256-CBC, PKCS #7 padding
Encrypted key MAC	HMAC-SHA512
Document Key wrapping	RSA-OAEP-MGF1-SHA1
Digital signature	RSASSA-PKCS1-v1_5

Digital signature digest	SHA-512
Digital signature canonicalization	Canonical XML 1.0 (omits comments)

For RSA, the recommended minimum key size is 3072 bits and is it not recommended to use certificates that are signed using SHA-1.

8.2. Key Rotation Support

A CPIX document can contain content protection information for multiple crypto-periods, or period of time for content encrypted using key rotation.

When content is protected with key rotation, a CPIX document shall contain one or more [<ContentKey>](#) elements and one or more [<ContentKeyPeriod>](#) elements, one of each per crypto-period which the document covers. Each [<ContentKey>](#) element contains the key material for a single crypto-period. The crypto-period itself is identified by the [<ContentKeyPeriod>](#) element, that includes `@start`, `@end` or `@index` attributes.

Key rotation may be supported in complex workflows, with one entity requesting DRM Signaling for multiple crypto periods, and another entity providing the requested information (keys, DRM system-specific information for the crypto period, etc). Clause 9 of [\[DASHIF-IOPv5p6\]](#) defines three scenarios for key rotation. The decision for encrypting content following one of these scenarios is made, most of the time, by the entity requesting DRM signaling, the entity providing this information is not aware of it. As a consequence, in some cases, the response will include some information that will not be inserted in the MPD (if requested for DASH content). In more details:

Manifest based key rotation signaling

The entity providing the DRM signaling shall insert the `PSSH` element and the `ContentProtectionData` element under the [<DRMSystem>](#) element associated to all content keys. The `ContentProtectionData` element shall be inserted in the MPD under the `ContentProtection` element. The `PSSH` element may be inserted under the `moov` box.

In-band key rotation signaling

The entity providing the DRM signaling shall insert the `PSSH` element and the `ContentProtectionData` element under the [<DRMSystem>](#) element associated to all content keys. The `ContentProtectionData` element shall be ignored. The `PSSH` element shall be inserted under the `moov` box and `moof` boxes of the segments that are encrypted by this key.

In-band key hierarchy

CPIX supports expressing two-level key hierarchies, where each leaf key has exactly one root key that is required in order to use the leaf key. Both root keys and leaf keys are represented using [<ContentKey>](#) elements, with leaf keys indicated by the presence of a `@dependsOnKey` attribute that references the root key as described in clause [§ 7.4.6 ContentKey Element](#).

If a CPIX document includes at least one [<ContentKey>](#) element that has a `@dependsOnKey` attribute, the content referenced by the `@contentId` attribute is fully protected with key hierarchy. A CPIX document may include [<ContentKey>](#) elements for leaf keys only, the referenced root key is then provided in a different document.

When using hierarchical keys, only the leaf keys shall be used to encrypt media content. Therefore, root keys shall not be referenced by any [<ContentKeyUsageRule>](#) elements.

The `PSSH` element under the [<DRMSystem>](#) element associated to the root key shall be inserted under the `moov` box, while, for the leaf keys, it shall be inserted under the `moof` boxes of the segments that are encrypted by this leaf key. The `ContentProtectionData` element of the [<DRMSystem>](#) element associated to the root key shall be inserted in the MPD. The entity providing the signaling uses the presence or not of the `@dependsOnKey` attribute in the [<ContentKey>](#) element for knowing what type of signaling it needs to generate for every key.

Note that not all DRM systems support key hierarchy.

8.3. Content Keys with Several Protection Encryption Schemes

In [MPEGCENC], several protection schemes that are not interoperable are defined. This means that several encrypted versions for the same content in the clear are created if the targeted devices support one or another protection scheme. While it may not be recommended, it is possible to use the same Content Keys when encrypting these different versions. In term of CPIX document, this means that several documents need to be created, these documents will differ on the @commonEncryptionScheme attribute under the <ContentKey> element which will take a different value depending on the protection schemes. Note that depending on the DRM, some elements under the <DRMSystem> element may also be different.

9. Examples§

Example CPIX documents are available on [GitHub](#). The examples contain valid data unless explicitly noted otherwise. Their contents are described on the referenced page.

10. Abbreviations§

AES

Advanced Encryption Standard

BMFF

Base Media File Format

CBC

Cypher Block Chaining

CDN

Content Delivery Network

CMS

Content Management System

CPIX

Content Protection Information eXchange

DASH

Dynamic Adaptive Streaming over HTTP

DRM

Digital Right Management

EPG

Electronic Program Guide

FPS

Frames Per Second

HD

High Definition

HDR

High Dynamic Range

HDS

HTTP Dynamic Streaming

HLS

HTTP Live Streaming

ISO

International Organization for Standardization

IV

Initialization Vector

KID

Key IDentifier

MAC

Message Authentication Code

MPD
Media Presentation Description

OD
Optional with Default value

PKCS
Public Key Cryptography Standards

PSSH
Protection System Specific Header

RSA
Rivest, Shamir, & Adleman

SD
Standard Definition

SHA
Secure Hash Algorithm

UHD
Ultra High Definition

URI
Uniform Ressource Identifier

UUID
Universally Unique IDentifier

VOD
Video On Demand

WCG
Wide Color Gamut

XML
eXtensible Markup Language

XSD
XML Schema Definition

References[§]

Normative References[§]

[DASHIF-IOPv5p6]

Guidelines for Implementation: DASH-IF Interoperability Points; Part 6 Content Protection, version 5.0, January 2022. URL: <https://dashif.org/guidelines/iop-v5/iop-v5/>

[HLS]

R. Pantos. HTTP Live Streaming 2nd Edition. Internet Draft. URL: <https://datatracker.ietf.org/doc/html/draft-pantos-hls-rfc8216bis-12>

[MPEGCENC]

ISO/IEC 23001-7:2016: Information technology - MPEG systems technologies - Part 7: Common encryption in ISO base media file format files, February 2016. URL: <https://www.iso.org/standard/68042.html>

[RFC6030]

IETF RFC 6030: Portable Symmetric Key Container (PSKC), October 2010. URL: <https://www.rfc-editor.org/rfc/rfc6030>

[XMLDSIG-CORE]

W3C® Recommendation 11 April 2013: XML Signature Syntax and Processing Version 1.1, Donald Eastlake, Joseph Reagle, David Solo, et al. (Second Edition), 10 June 2008. URL: <https://www.w3.org/TR/xmlsig-core/>

[XMLENC-CORE]

W3C® Recommendation 11 April 2013: XML Encryption Syntax and Processing Version 1.1, Donald Eastlake, Joseph Reagle, 10 December 2002. URL: <https://www.w3.org/TR/xmlenc-core/>

[XMLSCHEMA11-2]

W3C® Recommendation 5 April 2012: W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, David Peterson et al. URL: <https://www.w3.org/TR/xmlschema11-2/>

