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

September 6th, 2016

DASH Industry Forum

Version 2.0

Scope

The scope of this document is to define a Content Protection Information Exchange Format (CPIX). The 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 content 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 2.0 of the CPIX document. Changes with respect to Version 1.0 are:

- The CPIX document structure has been extensively revised in order to make it more generic so that media formats other than those described in [DASH-IF-IOP] can be accommodated.
    - It no longer mirrors the structure of an MPD file, but rather contains rules to describe the mapping of streams to content keys.
    - It allows for supporting additional DRM signaling schemes by allowing alternative types of DRM signaling elements.
- Sets of elements can be authenticated by means of XML element signing.
- Versioning has been added so that the history of added elements can be tracked.

Disclaimer

Contents

1 Introduction

1.1 General

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.

1.2 References

[DASH] ISO/IEC 23009-2:2014 Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats.

[DASH-IF-IOP] Guidelines for Implementation: DASH-IF Interoperability Points, version 3.3, June 2016.

[DASH-attributes] http://www.dashif.org/identifiers/protection/

[CENC] ISO/IEC 23001-7:2016, Third Edition, Information technology – MPEG systems technologies – Part 7: Common encryption in ISO base media file format files.

[RFC6030] IETF RFC 6030, “Portable Symmetric Key Container (PSKC)”, October 2010.

[CPIX-XML] http://dashif.org/guidelines/

[XML-DSIG] XML Signature Syntax and Processing (Second edition), https://www.w3.org/TR/xmldsig-core/

[XML-ENC] XML Encryption Syntax and Processing, http://www.w3.org/TR/xmlenc-core/

1.3 Normative Language

See [DASH-IF-IOP] section 2.3.

1.4 Terms & Definitions

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

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.

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

PSSH: “Protection System Specific Header” box that is part of an ISO_BMFF file. This box contains DRM Signaling.

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

2 Use Cases and Requirements

2.1 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 section presents different use cases where such exchanges are required. Section 2.2 is an overview of the general context in which exchange of content protection information is happening, Section 2.3 describes some workflows for content creation and section to go in the details of how content protection information can be exchanged over an interface between two entities.

2.2 Overview of the End to End Architecture

This informative section gives a general overview of the context in which content protection information need to be exchanged between entities in the backend. It completes section 7.5 of [DASH-IF-IOP] by putting more emphasis on the backend aspects.

This informative section 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.

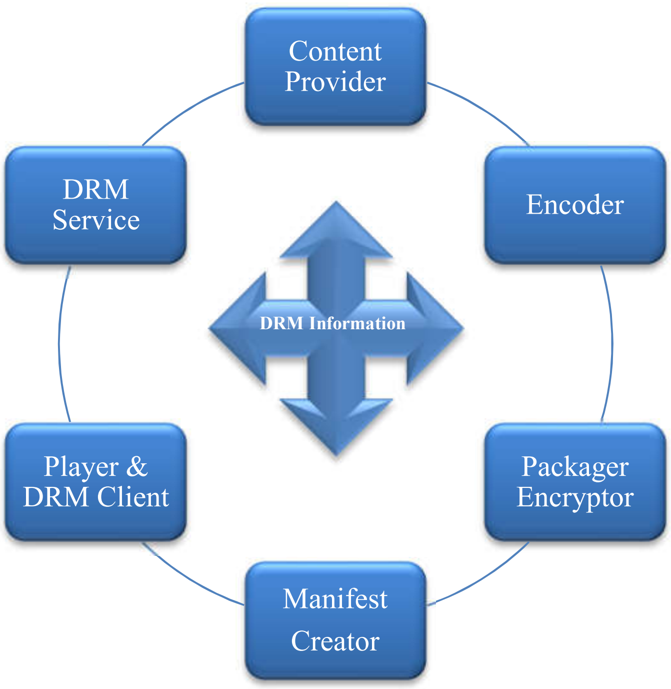


Figure 1: Logical roles that exchange DRM information and media.

Figure 1 shows logical entities that may send or receive DRM information such as media keys, asset identifiers, licenses, and license acquisition information. A physical entity may combine multiple logical roles, and the point of origin for information, such as media keys and asset identifiers, can differ; so various information flows are possible. This is an informative 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), 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 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 Provider). 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 Provider and Streamer, 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 Provider.

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 must include descriptors for Common Encryption and DRM key management systems, and should include identification of the default_KID for each AdaptationSet element, and sufficient information in UUID ContentProtection Descriptor elements to acquire a DRM license. The default_KID is available from the Packager and any other role that created it, and the DRM specific information is available from the DRM Provider.

DRM Client – Gets information from different sources: media manifest files, media files, and DRM licenses.

DRM Service – The DRM Provider creates licenses containing a protected Content Key that can only be decrypted by a trusted client.

The DRM Provider needs to know the default_KID and DRM SystemID and possibly other information like asset ID and player domain 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 Provider typically must supply the Streamer and the Packager license acquisition information for each UUID ContentProtection Descriptor 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.

2.3 Use Cases for the Preparation of Content

2.3.1 Introduction

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

As for the previous section, this informative section 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.

2.3.2 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 Figure 2 where arrows show the flow of information.

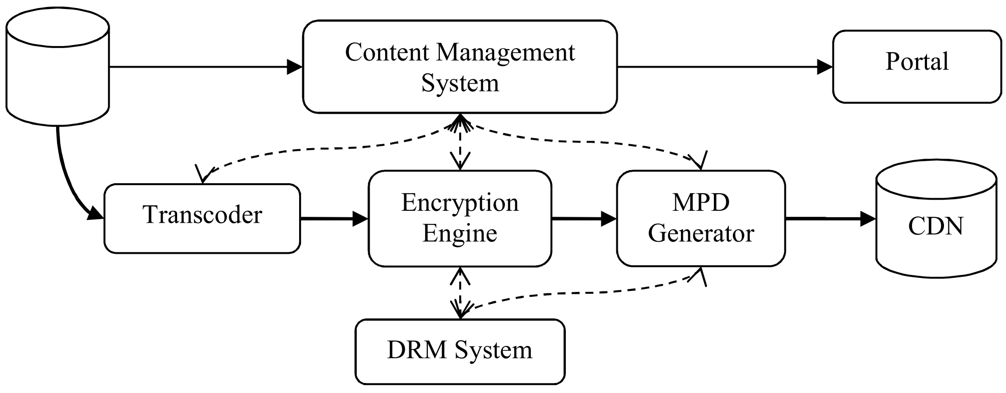


Figure 2: Example of workflow for On-Demand Content preparation.

2.3.3 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…

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 Figure 3 where arrows show the flow of information.

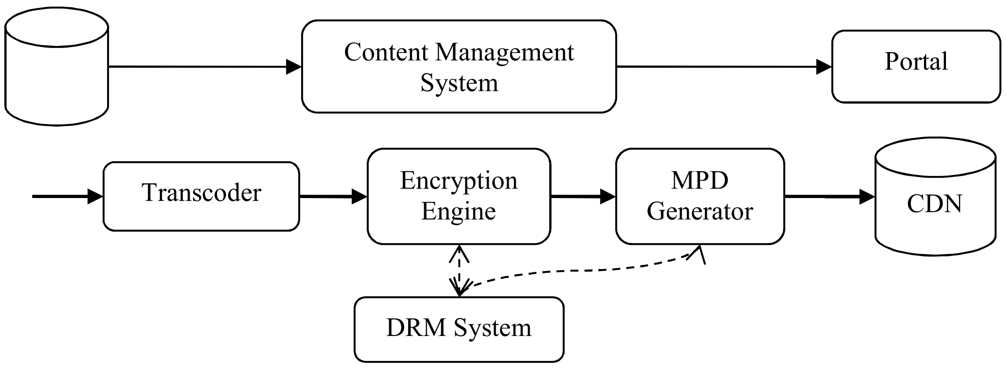


Figure 3: Example of workflow for Live Content preparation.

2.3.4 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.

2.3.5 Electronic Sell Through

In order to make available its Content in a defined and controlled quality, a content owner is preparing it. Preparation includes transcoding to the desired format and encryption of the resulting segments. The content owner is generating also the Content Key(s). At the end of the process, Content is ready and stored along with the Content Key(s).

Later the content owner distributes the prepared Content to multiple locations, in addition metadata describing it is also made available to retail platforms so that Content becomes salable on multiples Portals. In parallel, the content owner distributes the Content Key(s) to any authorized DRM system. A DRM system is authorized if it is one used by one of the Portal that has this Content for sale.

2.4 Exchange over an Interface

2.4.1 Introduction

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

2.4.2 Content Key Delivery to One Entity

In the simplest use case content protection information is made of a Content Key one entity sends some Content Keys to the other entity. This use case is summarized in Figure 4.

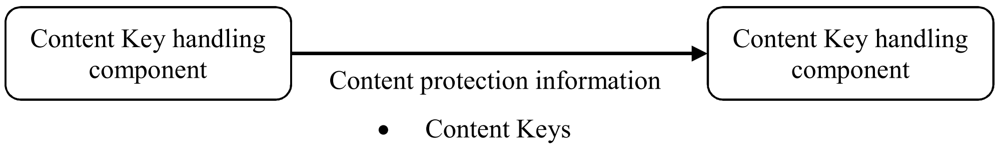

Content Key handling component
Content protection information
• Content Keys
Content Key handling component

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 KeyIDs. Any context or meaning is attributed externally. The document simply serves as a standard way to serialize content keys for delivery.

2.4.3 Secure Content Key Delivery to Several Entities

This use case is an extension of use case presented in Section 2.4.2 and is compatible with the use cases presented in the following sections. This use case is summarized in Figure 5

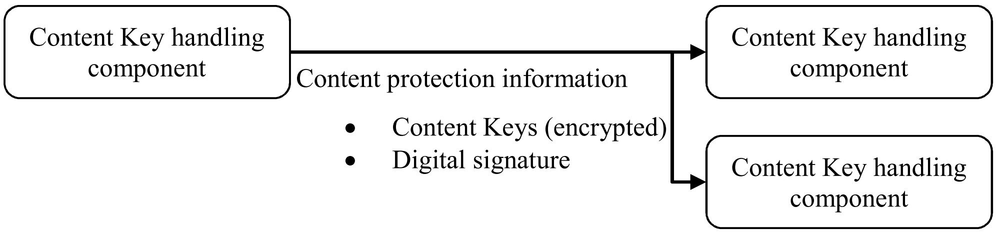

Content Key handling component
Content protection information
Content Keys (encrypted)
Digital signature
Content Key handling component
Content Key handling component

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.

2.4.4 Content Key Delivery with Usage Rules

These use cases are extension of use case presented in Section 2.4.2 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.

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

2.4.4.1 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. This use case is summarized in Figure 6.

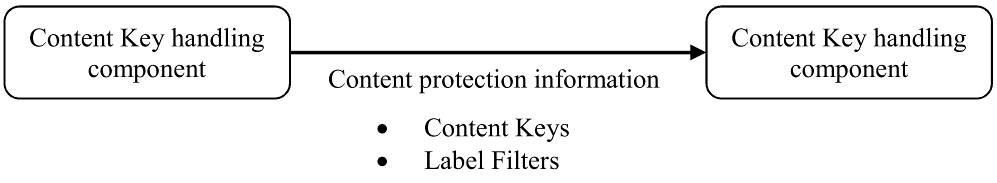

Content Key handling component
Content protection information
Content Keys
Label Filters
Content Key handling component

Figure 6: Content Key Delivery with Label Filter.

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.

2.4.4.2 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. This use case is summarized in Figure 7.

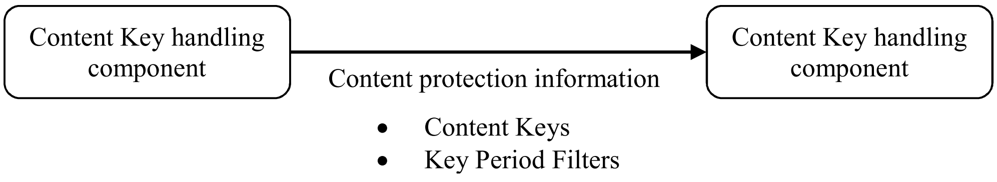

Content Key handling component
Content protection information
Content Key handling component
Content Keys
Key Period Filters

Figure 7: Content Key Delivery with Period Filter.

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

2.4.4.3 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. This use case is summarized in Figure 8.

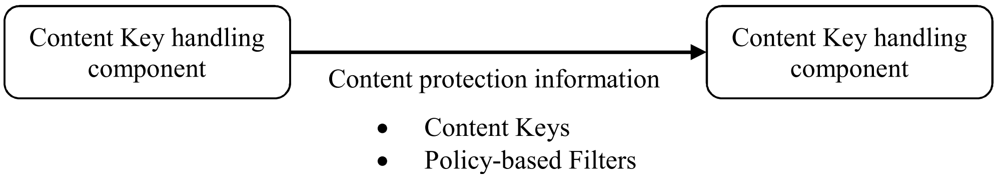

Content Key handling component
Content protection information
Content Key handling component
Content Keys
Policy-based Filters

Figure 8: Content Key Delivery with Policy-based Filters.

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

2.4.5 Content Key Delivery with DRM Signaling

This use cases is an extension of the use case presented in Section 2.4.2 and is compatible with the usage rules presented in Section 2.4.4.

This use case adds DRM System Signaling information to each Content Key. The recipient may embed this signaling into the data streams it generates. This use case is summarized in Figure 9.

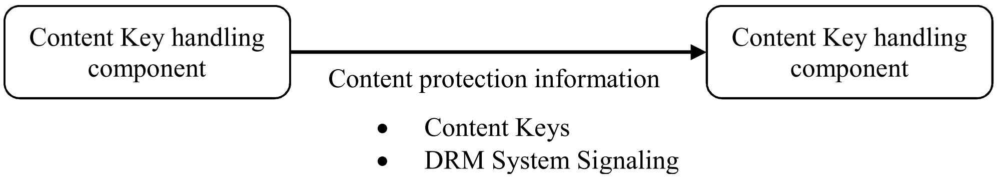

Content Key handling component
Content protection information
Content Key handling component
Content Keys
DRM System Signaling

Figure 9: 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 a HLS playlist or signaling data for an ISOBMFF file.

While the CPIX format primarily targets DASH and includes only elements necessary to carry the DRM system signaling data required by DASH, the document format is designed to be generic. The use of 3rd party extensions enable the inclusion of DRM system signaling in forms suitable for other media delivery technologies (e.g. HTTP Live Streaming).

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.

2.4.6 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. It is shown in Figure 10.

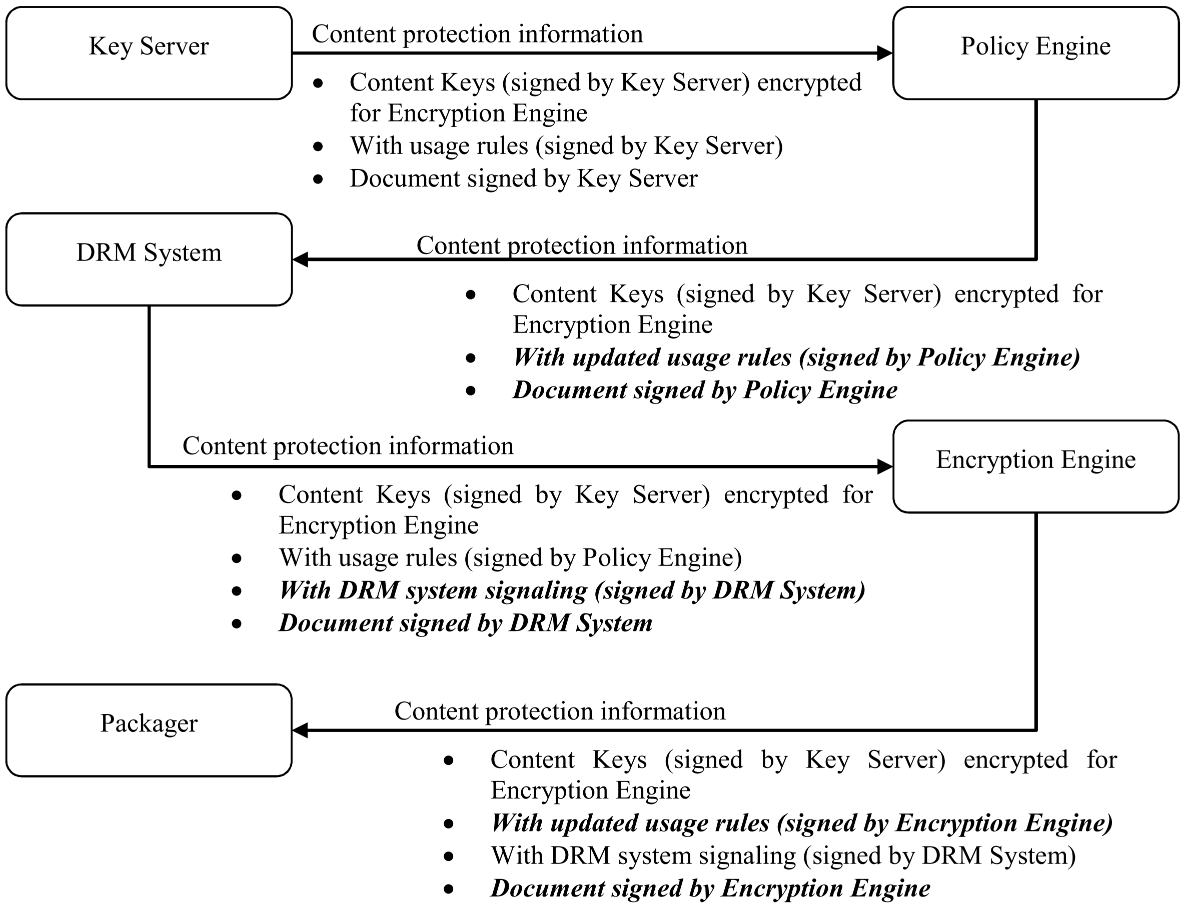

Key Server
Content protection information
Policy Engine
• Content Keys (signed by Key Server) encrypted for Encryption Engine
• With usage rules (signed by Key Server)
• Document signed by Key Server
DRM System
Content protection information
• Content Keys (signed by Key Server) encrypted for Encryption Engine
• With updated usage rules (signed by Policy Engine)
• Document signed by Policy Engine
Content protection information
Encryption Engine
• Content Keys (signed by Key Server) encrypted for Encryption Engine
• With usage rules (signed by Policy Engine)
• With DRM system signaling (signed by DRM System)
• Document signed by DRM System
Packager
Content protection information
• Content Keys (signed by Key Server) encrypted for Encryption Engine
• With updated usage rules (signed by Encryption Engine)
• With DRM system signaling (signed by DRM System)
• Document signed by Encryption Engine

Figure 10: 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.

2.5 Workflow Examples

2.5.1 Encryptor Producer and Encryptor Consumer

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:

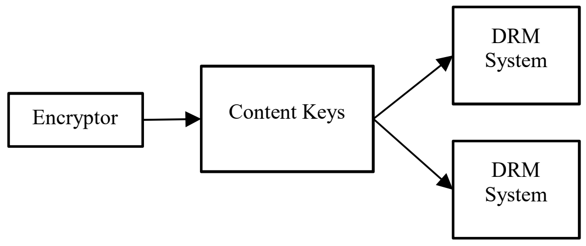


Figure 11: Encryptor Producer.

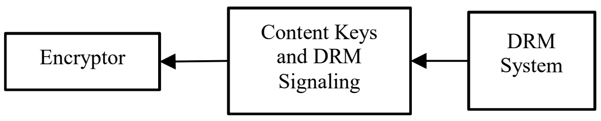


Figure 12: Encryptor Consumer.

All workflows require that content protection information and Content Keys be exchanged between two or more entities. In the examples in Figure 11 and Figure 12 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.

Implementations are recommended to encrypt the Content Keys as they are very sensitive data.

Implementations are recommended to sign any part of the document they generate, as well as the document as a whole, to enable recipients to verify the authenticity of the received data.

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.

Encryptor Producer

This informative section shows a possible workflow for securing the exchange of the key information between entities when the Encryptor generates the Content Keys. In this example, shown in Figure 11, 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 must 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 Figure 13.

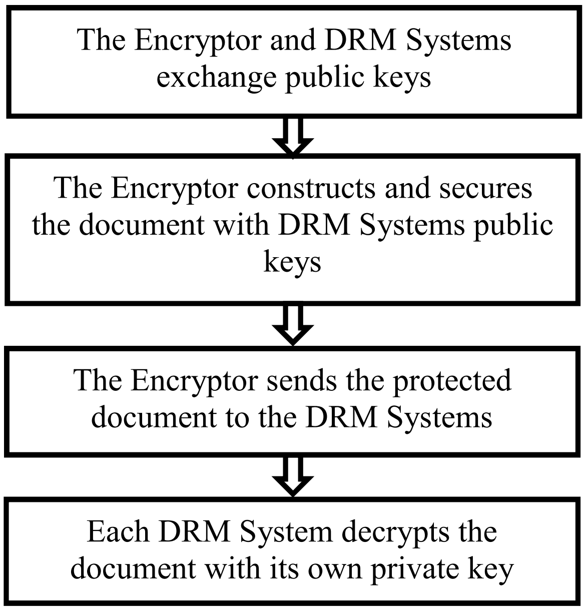

The Encryptor and DRM Systems exchange public keys
The Encryptor constructs and secures the document with DRM Systems public keys
The Encryptor sends the protected document to the DRM Systems
Each DRM System decrypts the document with its own private key

Figure 13: Encryptor Producer example steps.

Encryptor Consumer

This informative section 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, shown in Figure 12, 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 must 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 Figure 14.

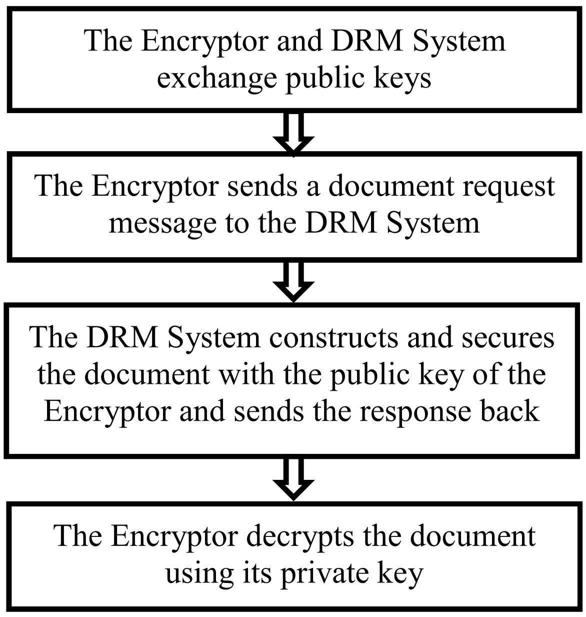

The Encryptor and DRM System exchange public keys
The Encryptor sends a document request message to the DRM System
The DRM System constructs and secures the document with the public key of the Encryptor and sends the response back
The Encryptor decrypts the document using its private key

Figure 14: Encryptor Consumer example steps.

Multiple Producers

This informative section 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.

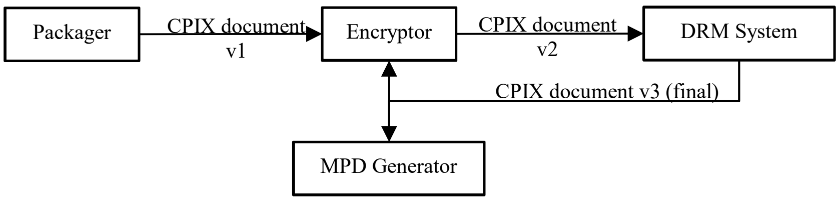


Figure 15: Multiple Producers example.

- The first step is the Trust establishment. Public keys must 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 related 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 Figure 16.

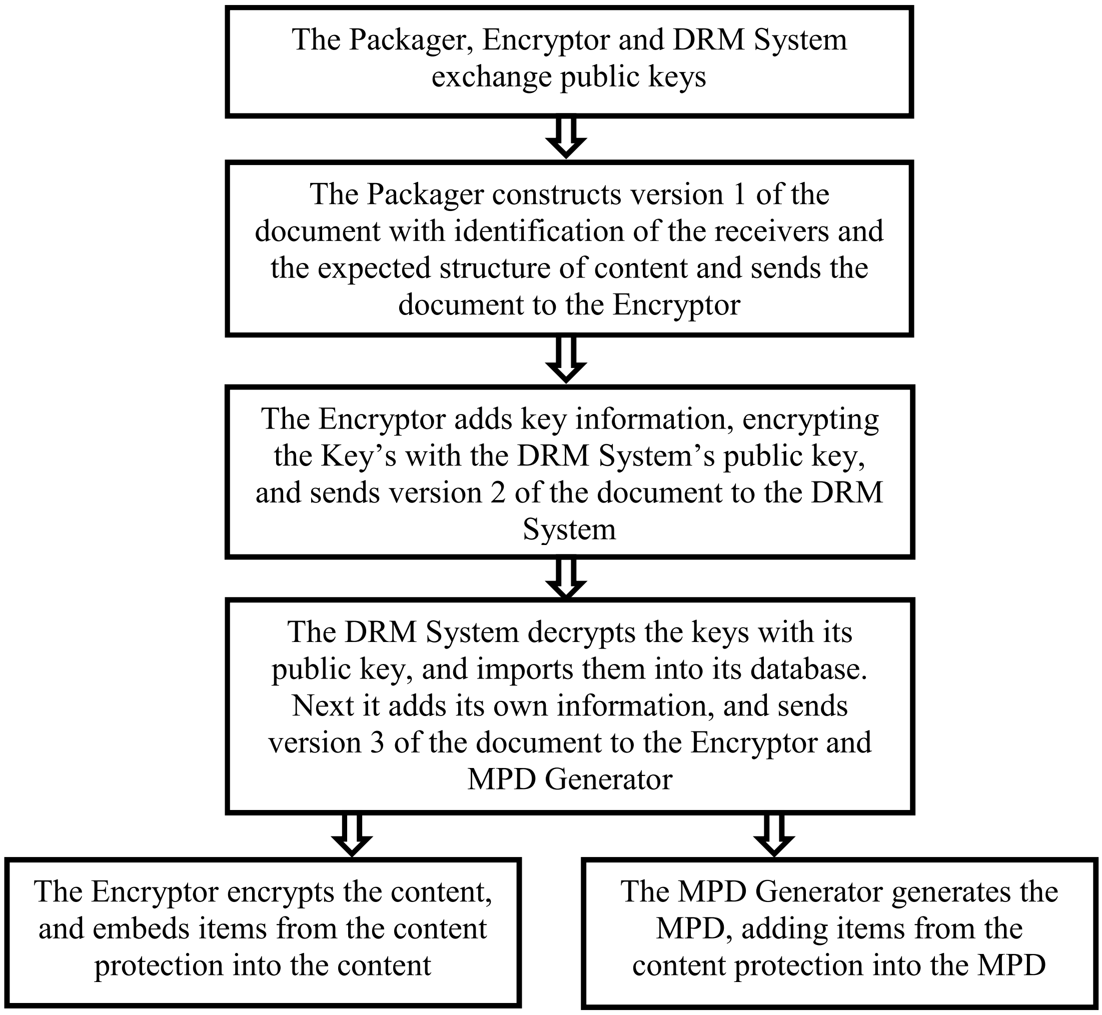

The Packager, Encryptor and DRM System exchange public keys
The Packager constructs version 1 of the document with identification of the receivers and the expected structure of content and sends the document to the Encryptor
The Encryptor adds key information, encrypting the Key's with the DRM System's public key, and sends version 2 of the document to the DRM System
The DRM System decrypts the keys with its public key, and imports them into its database. Next it adds its own information, and sends version 3 of the document to the Encryptor and MPD Generator
The Encryptor encrypts the content, and embeds items from the content protection into the content
The MPD Generator generates the MPD, adding items from the content protection into the MPD

Figure 16: Multiple Producers example steps.

2.6 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 [DASH-IF-IOP].

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

Entities exchanging content protection information should be authenticated.

3 XSD Schema Definition

3.1 Introduction

This section 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 Section 2). This is an XML file that is described by the XSD provided in [CPIX-XML]. This section describes in details elements part of the schema.

3.2 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 must be able to decrypt keys and must 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 (`@updateVersionNumber`).
- **`Signature`**: Each instance of this element contains a digital signature [XML-DSIG] 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.

Figure 17 shows the first elements and a high level view of the structure. Detailed description of the structure is given in the following sections.

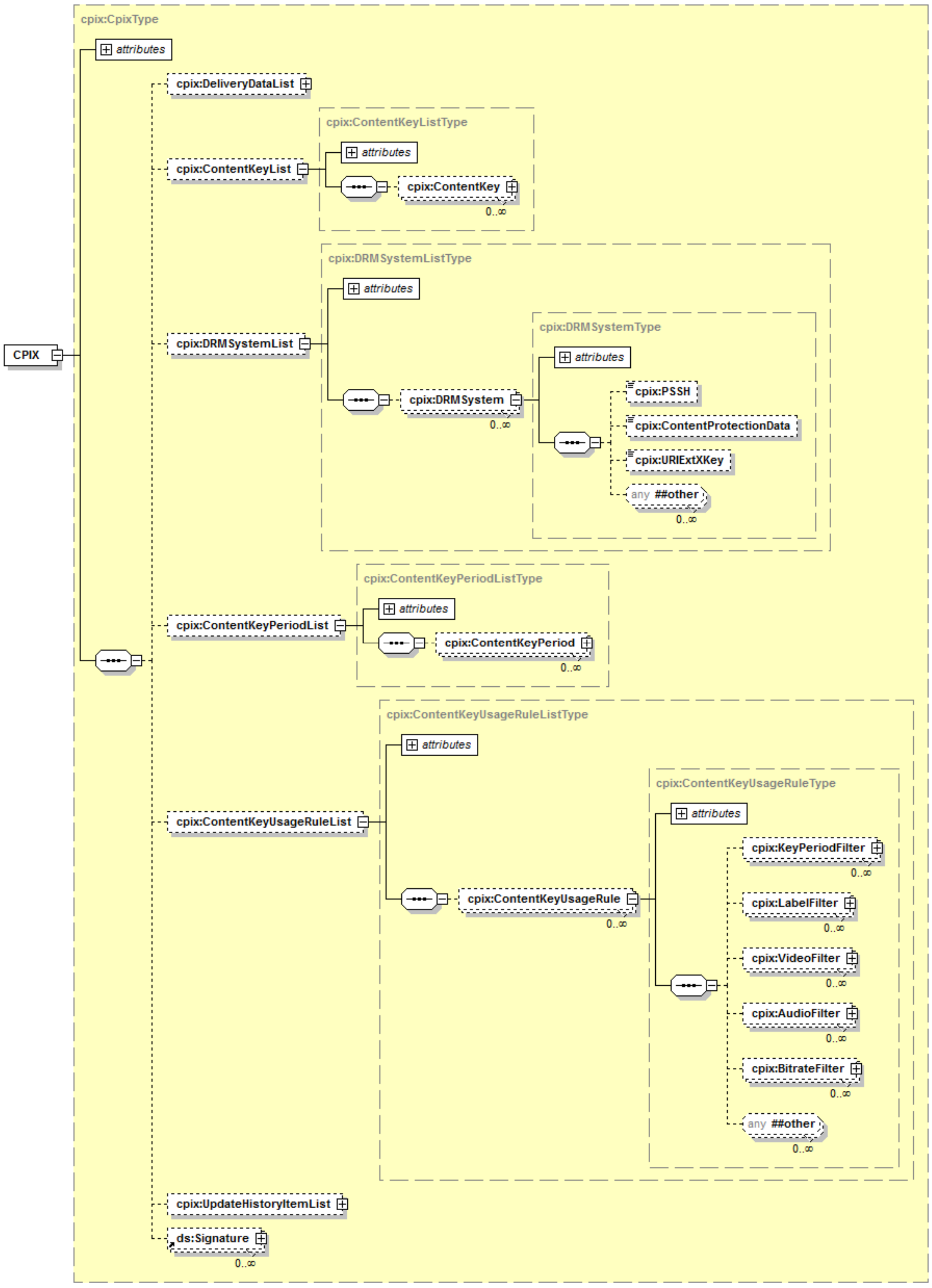


Figure 17: Content Protection Information Exchange Format high level view.

3.3 Hierarchical data model

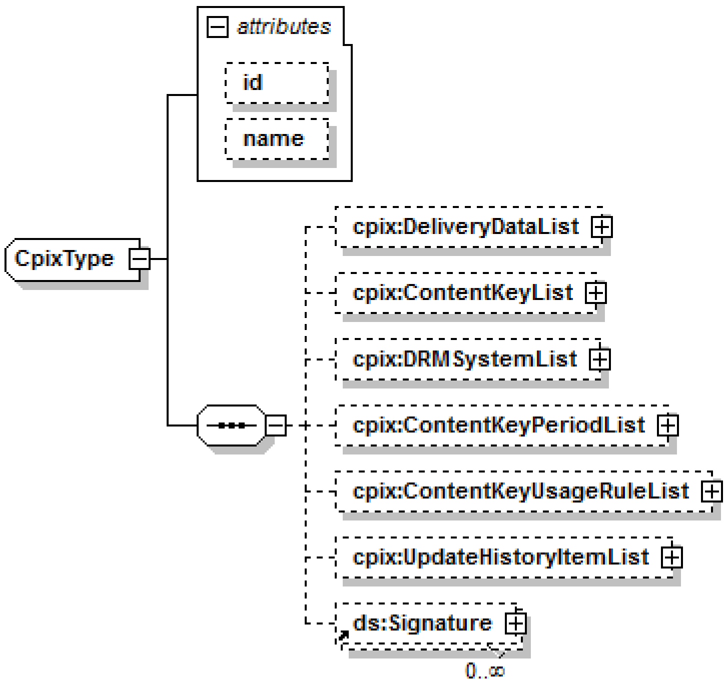
In all tables of this section, the following convention is used

- **Elements** are bold and the "Use" values are <**minOccurs**>...<**maxOccurs**> (N=unbounded).
- Attributes are non-bold preceded with an @ and the "Use" values are M=Mandatory, O=Optional, OD=Optional with Default Value, CM=Conditionally Mandatory.

The XSD schema for this model is provided in [CPIX-XML].

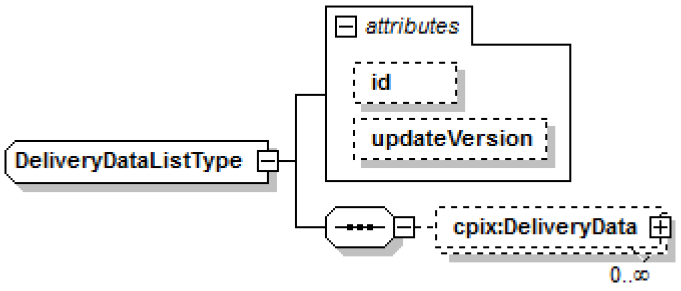
3.3.1 CPIX

Element or Attribute	**Use**	**Description**
CPIX		The root element that carries the Content Protection Information for a set of media assets.
@id	O	It specifies an identifier for the Media Presentation. It is also referred as the Asset or content ID. It is recommended to use an identifier that is unique within the scope in which this file is published.
@name	O	The name of the Presentation.
DeliveryDataList	0…1	A container for **DeliveryData** elements. If not present, Content Keys in the document are delivered in the clear, without encryption.
ContentKeyList	0…1	A container for **ContentKey** elements.
DRMSystemList	0…1	A container for **DRMSystem** elements. If not present, the document does not contain any DRM system signaling data.
ContentKeyPeriodList	0…1	A container for **ContentKeyPeriod** elements.
ContentKeyUsageRuleList	0…1	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	Container for **UpdateHistoryItem** elements.
Signature	0…N	Digital signature as defined in [XML-DSIG]. Each signature signs either the full document or any set of elements within the CPIX document. Every digital signature must contain an X.509 certificate identifying the signer and the associated public key.

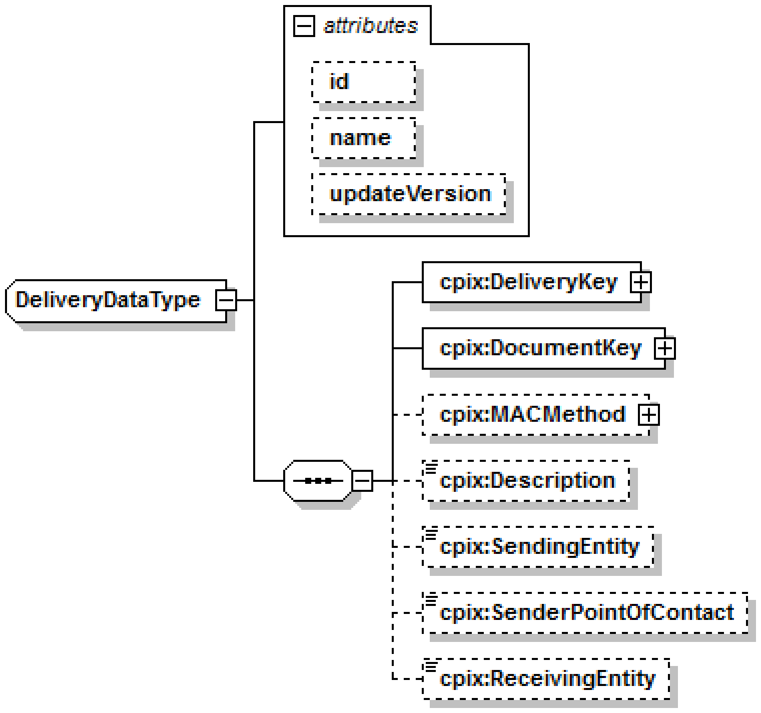


3.3.2 DeliveryDataList and DeliveryData

Element or Attribute	Use	Description
`DeliveryDataList`		
`@id`	O	It specifies 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	It has the same value as the `@id` of the **`UpdateHistoryItem`** element giving details on when the **`DeliveryDataList`** element was added.
`DeliveryData`	0…N	It 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 Contents Keys stored in this document. If this element is not present, then the Content Keys are in the clear in the file.

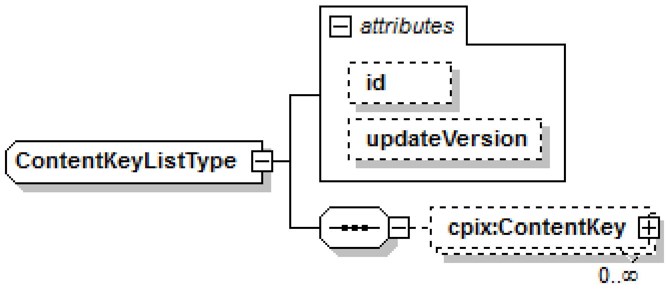


Element or Attribute	**Use**	**Description**
`DeliveryData`		
`@id`	O	It specifies an identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.
`@name`	O	It is the name of the Delivery Data.
`@updateVersion`	O	It has the same value as the `@id` of the **`UpdateHistoryItem`** element giving details on when the **`DeliveryData`** element was added.
`DeliveryKey`	1	Contains an X.509 certificate that identifies the intended recipient and the public key that was used to encrypt the document key. Refer to Section 4.1 for a description of the key management within the CPIX document.
`DocumentKey`	1	Contains the key that was used for encrypting any Content Key stored in a **`ContentKey`** element. The document key is encrypted using the public key listed in the recipient's X.509 certificate. Refer to Section 4.1 for a description of the key management within the CPIX document.
`MACMethod`	0…1	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 Section 4.1 for a description of the key management within the CPIX document.
`Description`	0…1	A description of the element.
`SendingEntity`	0…1	The name, of the entity generating this CPIX document.
`SenderPointOfContact`	0…1	Contact information, such as an email address, of the Sender.
`ReceivingEntity`	0…1	The name, of the entity capable of decrypting Content Keys in this CPIX document.



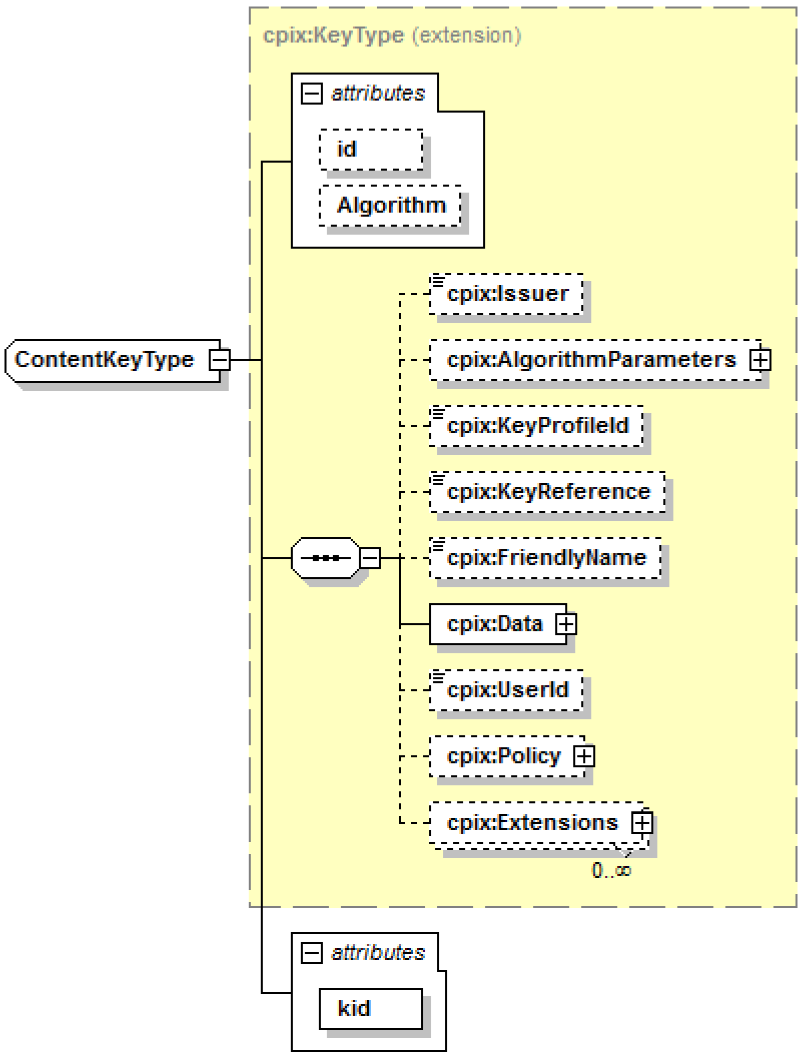
3.3.3 ContentKeyList and ContentKey

Element or Attribute	Use	Description
`ContentKeyList`		
`@id`	O	It specifies 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	It has the same value as the `@id` of the **`UpdateHistoryItem`** element giving details on when the **`ContentKeyList`** element was added.
`ContentKey`	0…N	Contains all information on a Content Key used to encrypt one or more Content Key Contexts.



Element or Attribute	Use	Description
`ContentKey`		

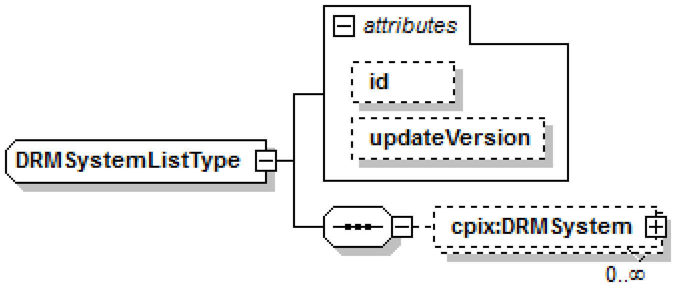
@id	O	Specifies an identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.
@Algorithm	O	This has the semantics defined in [RFC6030] and is made optional in this extension.
@kid	M	The unique identifier of the Content Key.
Extends keyType defined in [RFC6030]. The attribute *@id* and *@Algorithm* are optional in this extension. The key it 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 Section 4.1 for a description of the key management within the CPIX document.		



3.3.4 DRMSystemList and DRMSystem

Element or Attribute	Use	Description
DRMSystemList		

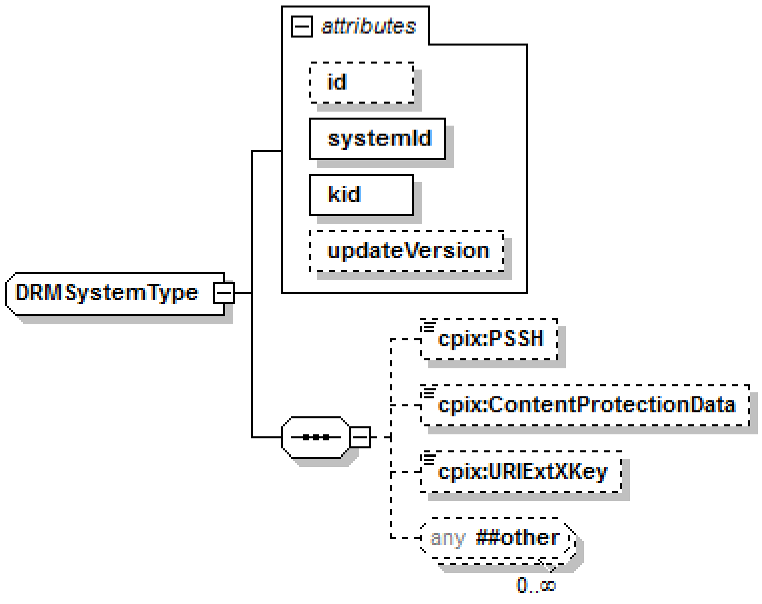
@id	O	It specifies an identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.
@updateVersion	O	It has the same value as the `@id` of the **`UpdateHistoryItem`** element giving details on when the **`DRMSystemList`** element was added.
DRMSystem	0…N	DRM system signaling information of a DRM system associated with a Content Key.



The **`DRMSystem`** 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, ISOBMFF and/or HLS formats. Implementations may extend CPIX documents with additional elements to provide DRM system signaling information for other formats.

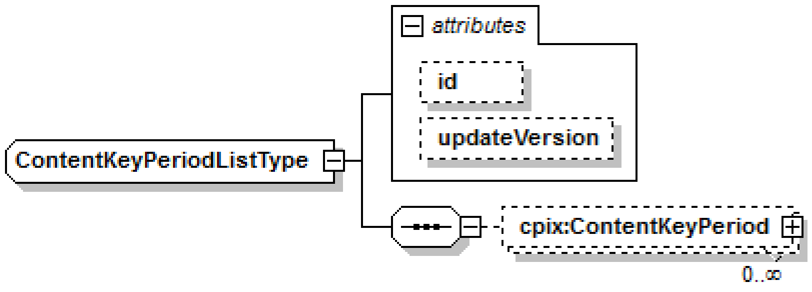
Element or Attribute	Use	Description
DRMSystem		
@id	O	It specifies an identifier for the element. It is recommended to use an identifier that is unique within the scope in which this CPIX document is published.
@systemId	M	This is the unique identifier of the DRM system. Values are defined on [DASH-attributes].
@kid	M	References the `@kid` field of the Content Key this **`DRMSystem`** element applies to.
@updateVersion	O	It has the same value as the `@id` of the **`UpdateHistoryItem`** element giving details on when the **`DRMSystem`** element was added.
PSSH	0…1	This is the full PSSH box that is added in the ISOBMFF. If a **`KeyPeriodFilter`** is defined on the Content Key, then the PSSH box goes under the moof box otherwise, the PSSH box goes under the moov box. This element is present only when the media content is in the ISOBMFF format.

`ContentProtectionData`	0…1	This is the full XML element to be added in the MPD under the **`ContentProtection`** element for this DRM. This element is present only when the content is in the DASH format.
`URIExtXKey`	0…1	This is the full data to be added in the URI parameter of the **`EXT-X-KEY`** tag of a HLS playlist. This element is present only when the content is in the HLS format.
*		Additional elements may be present containing signaling data for other media formats.



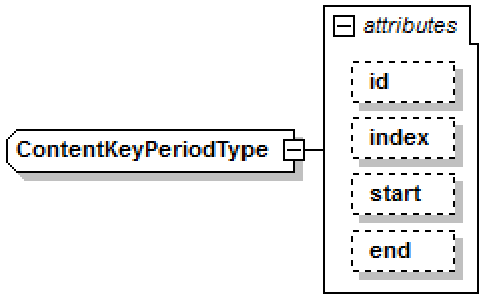
3.3.5 ContentKeyPeriodList and ContentKeyPeriod

Element or Attribute	**Use**	**Description**
`ContentKeyPeriodList`		
`@id`	O	It specifies 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	It has the same value as the `@id` of the **`UpdateHistoryItem`** element giving details on when the **`ContentKeyPeriodList`** element was added..
`ContentKeyPeriod`	0…N	One **`ContentKeyPeriod`** element for each period of time.



Element or Attribute	Use	Description
`ContentKeyPeriod`		
`@id`	O	It specifies 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	Numerical index for the key period. Mutually exclusive with `@start` and `@end`
`@start`	O	Wall clock (Live) or media time (VOD) for the start time for the period. Mutually inclusive with `@end`, and mutually exclusive with `@index`.
`@end`	O	Wall clock (Live) or media time (VOD) for the end time for the period. Mutually inclusive with `@start`, and mutually exclusive with `@index`.

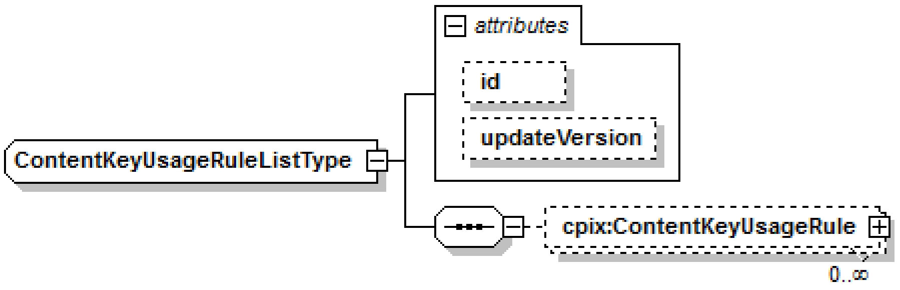
When `@start` and `@end` are present, the interval is defined by [`@start`,`@end`), meaning that the key is been used at time `@start` but not at time `@end`.



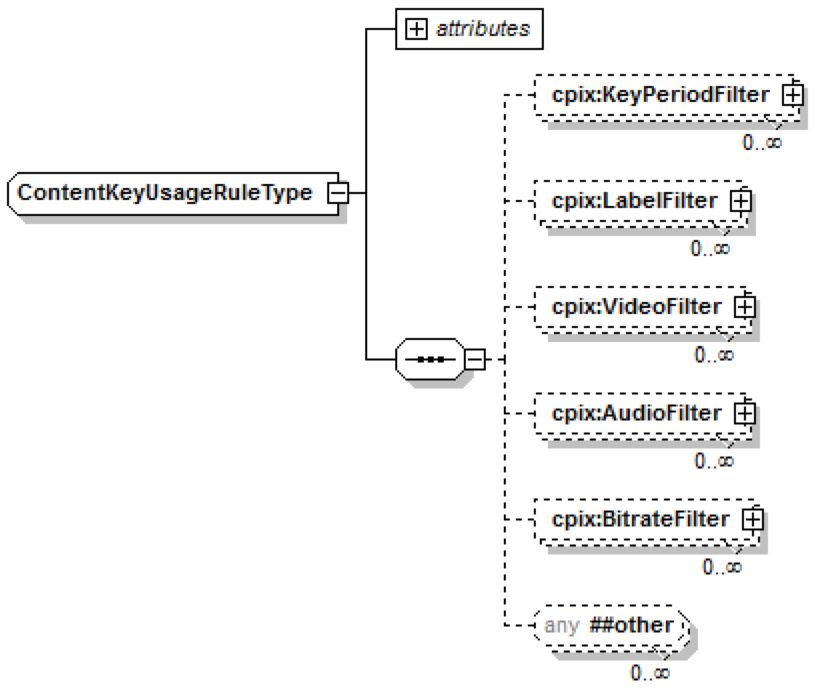
3.3.6 ContentKeyUsageRuleList and ContentKeyUsageRule

Element or Attribute	Use	Description
`ContentKeyUsageRuleList`		
`@id`	O	It specifies 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	It has the same value as the @id of the **UpdateHistoryItem** element giving details on when the **ContentKeyUsageRuleList** element was added.
ContentKeyUsageRule	0…N	A rule which defines a Content Key Context.



Element or Attribute	**Use**	**Description**
ContentKeyUsageRule		
@id	O	It specifies 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	It specifies the @kid field of the Content Key this **ContentKeyUsageRule** element applies to.
KeyPeriodFilter	0…N	It defines period of time constraints for the Content Key identified by @kid. This filters links **ContentKey** and **ContentKeyPeriod** elements.
LabelFilter	0…N	It defines a label association for the Content Key identified by @kid.
VideoFilter	0…N	It defines video constraints to be associated with the Content Key identified by @kid. This filter can only be used on media content of type video
AudioFilter	0…N	It defines audio constraints to be associated with the Content Key identified by @kid. This filter can only be used on media content of type audio.
BitrateFilter	0…N	It defines bitrate constraints to be associated with the Content Key identified by @kid.
*		Additional elements may be present containing proprietary filters



3.3.7 Usage Rules Filters

3.3.7.1 Introduction

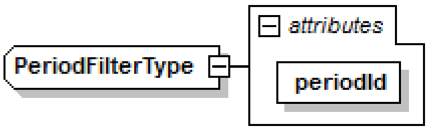
There can be several filters defined within a single **`ContentKeyUsageRule`**. In this case, all rules apply identically, the entity generating the **`ContentKeyUsageRule`** element or adding a new rule is responsible for ensuring that they do not contradict each other. A set of rules that would match 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 usage rule shall be considered invalid 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. **`VideoFilter`**`@minPixels` is defined but the implementation does not know the pixel count of the video samples). This condition must be treated as a fatal error in the processing of the CPIX document.

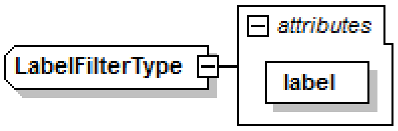
3.3.7.2 KeyPeriodFilter

Element or Attribute	**Use**	**Description**
`KeyPeriodFilter`		
`@periodId`	M	This references a **`ContentKeyPeriod`** element by its `@id` attribute. The filter will only match samples that belong to the referenced key period.



3.3.7.3 LabelFilter

Element or Attribute	Use	Description
`LabelFilter`		
`@label`	M	The filter will only match samples that carry a matching label. The exact meaning of labels is implementation-defined and must be agreed upon in advance by the producer and consumer of the CPIX document.



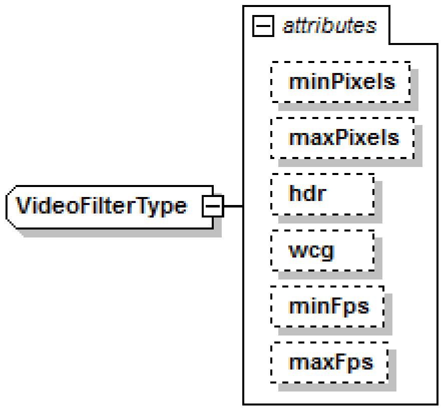
3.3.7.4 VideoFilter

Element or Attribute	Use	Description
`VideoFilter`		If present, even without any attributes, the filter will only match video samples.
`@minPixels`	O	The filter will only match video samples that contain at least this many pixels (encoded width x height before considering pixel/sample aspect ratio). The default value is 0 (zero).
`@maxPixels`	O	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	Boolean value indicating whether the matching video stream is encoded in HDR.
`@wcg`	O	Boolean value indicating whether the matching video stream is encoded in WCG.
`@minFps`	O	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	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.

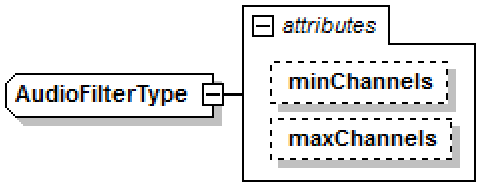
When `@minFps` and `@maxFps` are present, the interval is defined by (`@minFps,@ maxFps`], meaning that the filter is not used for content with nominal FPS equal to `@minFps` but is used for content with nominal FPS equal to `@maxFps`.



3.3.7.5 AudioFilter

Element or Attribute	Use	Description
`AudioFilter`		If present, even without any attributes, the filter will only match audio samples.
`@minChannels`	O	The filter will only match audio samples that contain at least this many channels. The default value is 0 (zero).
`@maxChannels`	O	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.

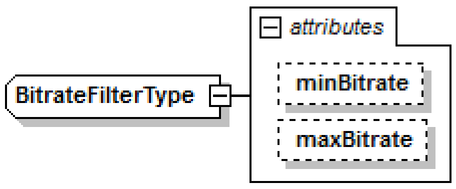


3.3.7.6 BitrateFilter

Element or Attribute	Use	Description
`BitrateFilter`		

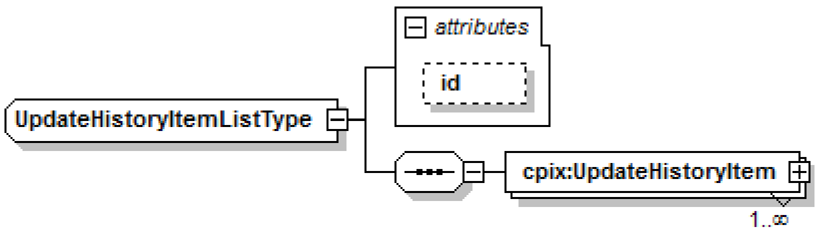
@minBitrate	O	The filter will only match samples from streams with a nominal bitrate in Mb/s of at least this value. The default value is 0 (zero). At least one of @minBitrate and @maxBitrate must be specified.
@maxBitrate	O	The filter will not match samples from streams with a nominal bitrate in Mb/s that exceeds this value. The default value is MAX_UINT32. At least one of @minBitrate and @maxBitrate must 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.



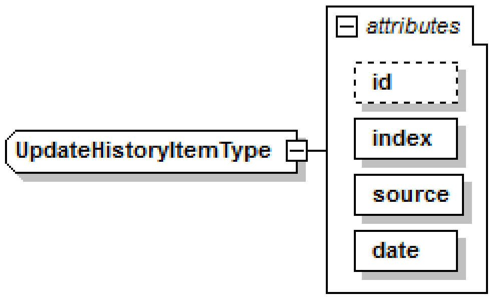
3.3.8 UpdateHistoryItemList and UpdateHistoryItem

Element or Attribute	Use	Description
UpdateHistoryItemList		
@id	O	It specifies 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	1…N	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.



Element or Attribute	Use	Description
UpdateHistoryItem		

@id	O	It specifies 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	M	This is the version number for the document update. Each **UpdateHistoryItem** element contains a unique @index value. It is a monotonically increasing number, starting at value 1.
@source	M	This is the identifier for the entity which performed the document update.
@date	M	This is the date and time when the document update was performed.



4 Key Management

4.1 Key Encryption in the CPIX document

4.1.1 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 key hierarchy is used for an efficient encryption avoiding duplication of encrypted content and expensive encryption methods. This section describes the mechanism that shall be used when encryption of the Content Keys in the document is used.

4.1.2 Key Hierarchy in 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. Typically, for Common Encryption as supported in [DASH-IF-IOP], 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 [XML-ENC].

Figure 18 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.

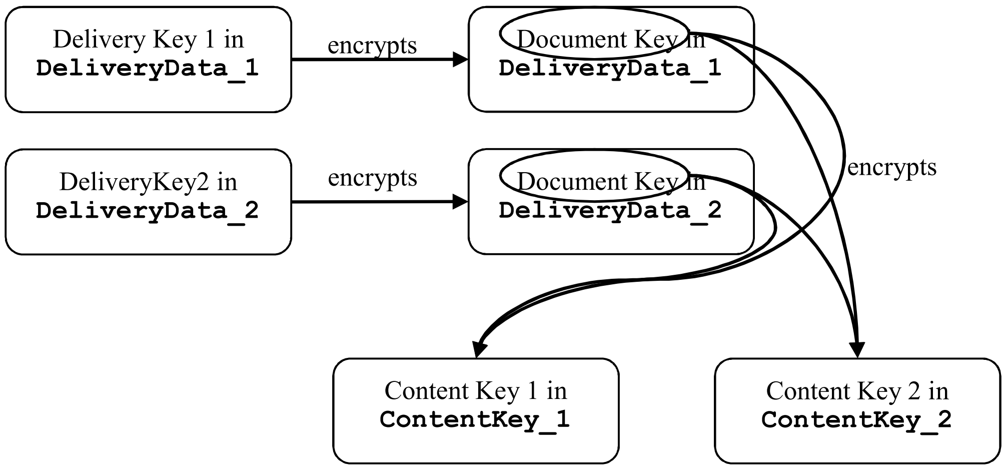


Figure 18: Key hierarchy encryption in the CPIX document.

4.1.3 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.

4.1.4 Digital Signature

Every element in the document that has an ***`@id`*** attribute can be signed according to [XML-DSIG]. 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).

4.1.5 Mandatory Algorithms

Table 1 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)

Table 1: Algorithms mandatory to support.

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

4.2 Key Rotation Support (informative)

A CPIX document can contain content protection information for multiple crypto-periods, or period of time for content encrypted using key rotation. In this case, the document shall contain one or more **`ContentKey`** elements, one 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).

Some key rotation implementations may store the Content Keys within the PSSH data. These keys are usually wrapped (encrypted) with another key, which is delivered to the client as part of the content license. This 'root' key should never be encoded in the CPIX document; instead the CPIX document shall only contain the Content Keys in the **`ContentKey`** elements.

5 XML Examples

The zip file available on DASH-IF web site contains four examples of XML files [CPIX-XML]. The XML example files are syntactically correct but do not contain real data. For examples, signature and encrypted data are dummy data. Examples with valid data can be found at https://github.com/Axinom/cpix-test-vectors.

Encrypted Keys

This example shows a CPIX document where keys can be decrypted by two entities ("Authorization Service 1234" and "Authorization Service 5678"). Both are identified by their X509 certificates.

The Content Key is encrypted by the Document Key, the Document Key is therefore encrypted two times (once for Authorization Service 1234 and once for Authorization Service 5678).

In term of DRMs, there are two different DRMs that can be used for accessing content. There are therefore two **`DRMSystem`** elements.

Clear Keys

This example shows a CPIX document where keys are not encrypted. It can be read by any entity and therefore does not contain **`DeliveryData`** elements.

The Content Key is available as plain data.

In term of DRMs, there are two different DRMs that can be used for accessing the content. There are therefore two **`DRMSystem`** elements.

Multiple Updates

This example, consisting of multiple files (v1…v3), shows a CPIX document which has been built up by successive updates by various entities as illustrated in the example in section 2.4.6. The document has three **`UpdateHistoryItem`** within it referencing the entities which performed the updates. Various elements in the document reference the **`UpdateHistoryItem`** for the update in which they were added by **`UpdateVersion`**.

It contains two Content Keys. The keys can be decrypted by the entity "DRM System 1234".

The document contains information for a single DRM system, so it only contains a single **`DRMSystem`** element.

Key Rotation

This example shows a CPIX document where keys can be decrypted by two entities ("Authorization Service 1234" and "Authorization Service 5678"). Both are identified by their X509 certificates.

It contains two Content Keys for two cryptoperiods. Two **`ContentKeyPeriod`** elements with by their start and end times are added, each referencing one **`ContentKey`** element using a **`PeriodFilter`** element. The Content Key is encrypted by the Document Key, the Document Key is therefore encrypted two times (once for Authorization Service 1234 and once for Authorization Service 5678).

In addition, the document contains **`DrmSystem`** elements for the two DRM systems.

6 Transfer Protocol

The preferred method of exchange of CPIX documents between entities is through a RESTful API. This API is defined as follows:

```
[BaseURL]/copyProtectionData/{{CPIX_ID}}
Where [BaseURL] is the scheme and host(e.g. https://acme-drm-service.com) of the target
resource.
Method: POST
Behaviour: The request to create copyProtectionData for an encryptor.
RequestBody:
Element: //CPIX
Definition: A CPIX element.
ResponseBody:
Element: //CPIX
Definition: A CPIX element.


—
[BaseURL]/copyProtectionData/{{CPIX_ID}}
Where [BaseURL] is the scheme and host(e.g. https://acme-drm-service.com) of the target
resource.
Method: GET
Behaviour: Return copyProtectionData for the referenced CPIX_ID.
RequestBody:
Element: NONE
ResponseBody:
Element: //CPIX
Definition: A CPIX element.


—
[BaseURL]/copyProtectionData/{{CPIX_ID}}
Where [BaseURL] is the scheme and host(e.g. https://acme-drm-service.com) of the target
resource.
Method: PUT
Behaviour: Update copyProtectionData referenced by the CPIX_ID
RequestBody:
Element: //CPIX
Definition: A CPIX element.
ResponseBody:
Element: NONE
—
[BaseURL]/copyProtectionData/{{CPIX_ID}}
Where [BaseURL] is the scheme and host(e.g. https://acme-drm-service.com) of the target
resource.
Method: DELETE
Behaviour: Delete/purge copyProtectionData referenced by the CPIX_ID.
RequestBody:
Element: NONE
ResponseBody:
Element: NONE
```