ETSI TS 103 799 V1.2.1 (2026-04)

TECHNICAL SPECIFICATION

Publicly Available Specification (PAS); DASH-IF Content Protection Information Exchange Format

CAUTION

*The present document has been submitted to ETSI as a PAS produced **by the DASH-IF WG of SVTA** and approved by the Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).*

Reference
RTS/JTC-122

Keywords
DASH, encryption, internet, PAS, video

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights 5
Foreword 5
Modal verbs terminology 5
Introduction 6
1 Scope 7
2 References 7
2.1 Normative references 7
2.2 Informative references 7
3 Definition of terms, symbols and abbreviations 8
3.1 Terms 8
3.2 Symbols 8
3.3 Abbreviations 8
4 Use Cases and Requirements 9
4.1 Introduction 9
4.2 Overview of the End-to-End Architecture 9
4.3 Use Cases for the Preparation of Content 10
4.3.1 Introduction 10
4.3.2 On-Demand Content 10
4.3.3 Live Content 11
4.3.4 Catch-up 12
4.4 Exchange over an Interface 12
4.4.1 Introduction 12
4.4.2 Content Key Delivery to One Entity 12
4.4.3 Secure Content Key Delivery to Several Entities 12
4.4.4 Content Key Delivery with Usage Rules 13
4.4.4.1 Introduction 13
4.4.4.2 Label Filter 13
4.4.4.3 Key Period Filter 13
4.4.4.4 Policy-based Filters 13
4.4.5 Content Key Delivery with DRM Signaling 13
4.4.6 Incremental Update and Extension of the document 14
4.4.7 Multiple Content Keys Delivery for Multiples Assets 15
4.4.8 Content Key Hierarchy Delivery for Content Packaging 15
4.4.9 Root Key Delivery for License Server Operation 15
4.5 Workflow Examples 15
4.5.1 Encryptor Producer and Encryptor Consumer 15
4.5.1.1 Introduction 15
4.5.1.2 Encryptor Producer 16
4.5.1.3 Encryptor Consumer 17
4.5.2 Multiple Producers 18
5 XSD Schema Definition 19
5.1 Introduction 19
5.2 Requirements 19
5.3 Structure Overview 20
5.4 Hierarchical Data Model 22
5.4.1 Introduction 22
5.4.2 CPIX Element 22
5.4.3 DeliveryDataList Element 23
5.4.4 DeliveryData Element 24
5.4.5 DocumentKey Element 25
5.4.6 ContentKeyList Element 26
5.4.7 ContentKey Element 27
5.4.8 HDCPData Element 28

5.4.9 DRMSystemList Element 29
5.4.10 DRMSystem Element 29
5.4.11 ContentProtectionData Element 31
5.4.12 HLSSignalingData Element 32
5.4.13 ContentKeyPeriodList Element 32
5.4.14 ContentKeyPeriod Element 33
5.4.15 ContentKeyUsageRuleList Element 34
5.4.16 ContentKeyUsageRule Element 35
5.4.17 Usage Rules Filters 36
5.4.17.1 Introduction 36
5.4.17.2 KeyPeriodFilter Element 37
5.4.17.3 LabelFilter Element 37
5.4.17.4 VideoFilter Element 38
5.4.17.5 AudioFilter Element 39
5.4.17.6 BitrateFilter Element 39
5.4.18 UpdateHistoryItemList Element 40
5.4.19 UpdateHistoryItem Element 40

6 Key Management 41
6.1 Key Encryption and Authentication in the CPIX Document 41
6.1.1 Introduction 41
6.1.2 Encryption 41
6.1.3 Authenticated Encryption 42
6.1.4 Digital Signature 43
6.1.5 Mandatory Algorithms 43
6.2 Key Rotation Support 43
6.3 Content Keys with Several Protection Encryption Schemes 44

Annex A (normative): CPIX XSD 45

History 49

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Technical Specification (TS) has been produced by and approved by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Modal verbs terminology

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 ETSI deliverables except when used in direct citation.

Introduction

The present 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 the present document.

1 Scope

The scope of the present document is to define a Content Protection Information Exchange (CPIX) Format. 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.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found in the ETSI docbox.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] Guidelines for Implementation: "DASH-IF Interoperability Points; Part 6: Content Protection", version 5.0, Janvier 2022.

[2] DASH-IF registry of DRM System IDs.

[3] IETF RFC 6030: "Portable Symmetric Key Container (PSKC)", October 2010.

[4] W3C® Recommendation 5 April 2012: "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes", David Peterson et al.

[5] W3C® Recommendation 11 April 2013: "XML Encryption Syntax and Processing Version 1.1", Donald Eastlake, Joseph Reagle, 10 December 2002.

[6] 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.

[7] ISO/IEC 23001-7:2023: "Information technology -- MPEG systems technologies -- Part 7: Common encryption in ISO base media file format files", 2023.

[8] R. Pantos. HTTP Live Streaming 2nd Edition. Internet Draft.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents may be useful in implementing an ETSI deliverable or add to the reader's understanding, but are not required for conformance to the present document.

Not applicable.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

content: one or more audio-visual elementary streams and the associated MPD if in DASH format

content key: cryptographic key used for encrypting part of the content

content key context: portion of a media stream which is encrypted with a specific content key

content protection: mechanism ensuring that only authorized devices get access to content

document key: cryptographic key used for encrypting the content key(s) in the CPIX document

DRM signaling: DRM specific information to be added in content for proper operation of the DRM system when authorizing a device for this content

NOTE: It is made of proprietary information for licensing and key retrieval.

Protection System Specific Header (PSSH): part of an ISO BMFF file

NOTE: This box contains DRM Signaling.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

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
HDCP	High-bandwidth Digital Content Protection
HDR	High Dynamic Range
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

4 Use Cases and Requirements

4.1 Introduction

Content Keys and DRM Signaling, as known as 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. Clause 4.2 is an overview of the general context in which exchange of content protection information is happening, clause 4.3 describes some workflows for content creation and clause 4.4 goes in the details of how content protection information can be exchanged over an interface between two entities.

4.2 Overview of the End-to-End Architecture

This clause gives a general overview of the context in which content protection information needs to be exchanged between entities in the backend. It completes clause 4 of [1] 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 [8].

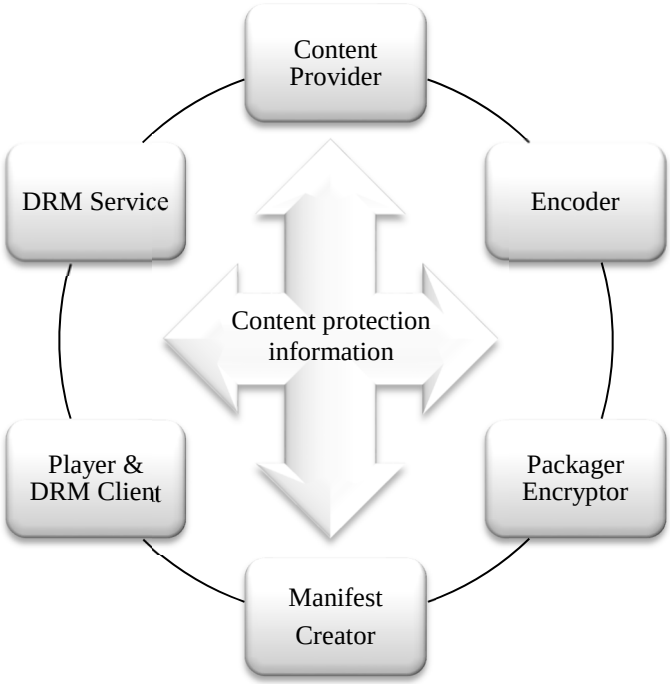

Figure 1: Logical roles that exchange DRM information and media

Figure 1 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, 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.

4.3 Use Cases for the Preparation of Content

4.3.1 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.

4.3.2 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. When the encrypted DASH Content is ready, the MPD is generated by an 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.

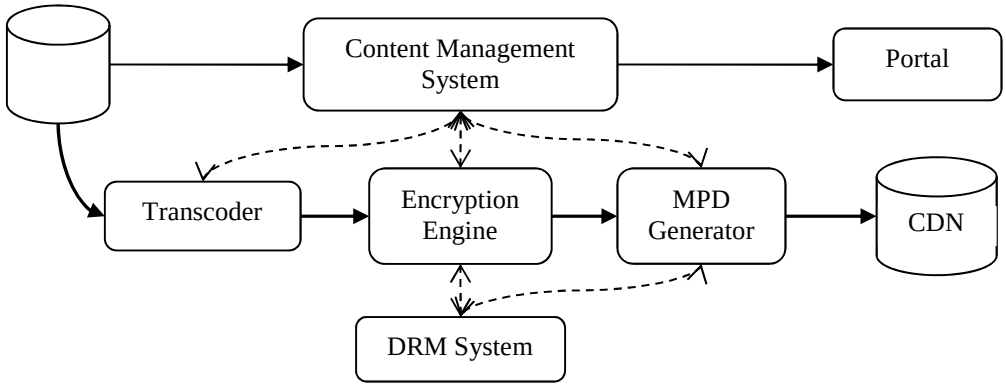

Figure 2: Example of workflow for On-Demand Content preparation

4.3.3 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 CMS 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.

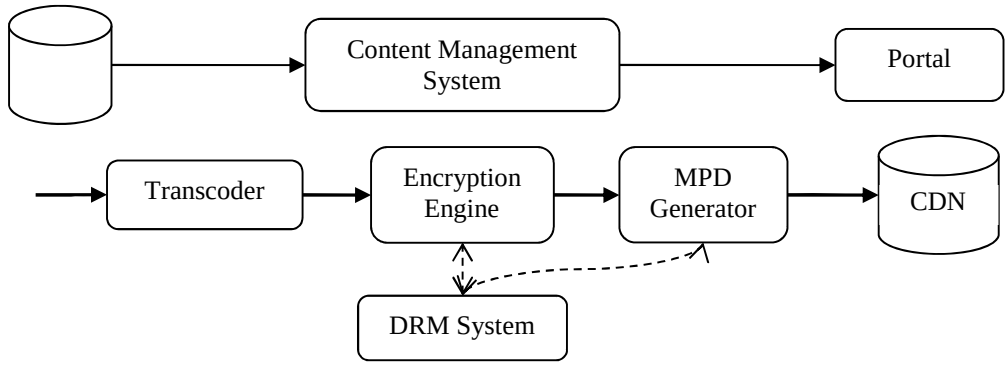

Figure 3: Example of workflow for Live Content preparation

4.3.4 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.

4.4 Exchange over an Interface

4.4.1 Introduction

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

4.4.2 Content Key Delivery to One Entity

In the simplest use case shown in Figure 4, content protection information is made of a Content Key. One entity sends a Content Key to the other entity.

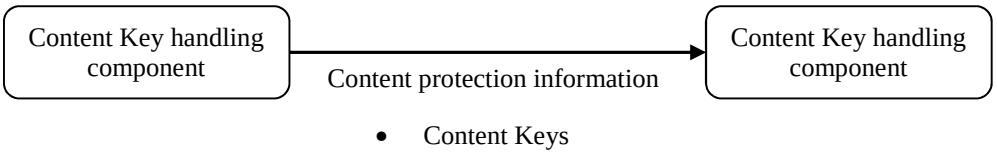

Figure 4: Content Key delivery to one entity

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.

4.4.3 Secure Content Key Delivery to Several Entities

This use case shown in Figure 5 is an extension of the use case of clause 4.4.2 and is compatible with the use cases presented in the following clauses.

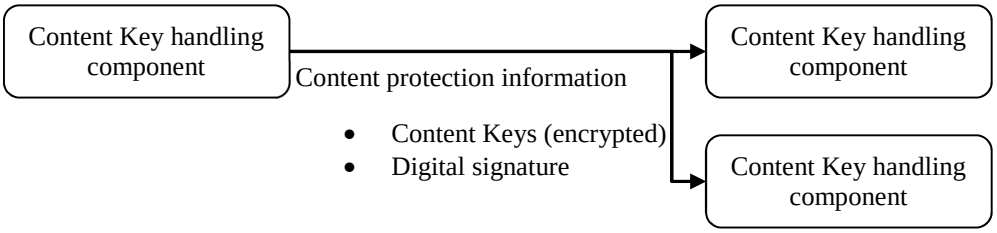

Figure 5: Secure Content Key Delivery to Several Entities

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.

4.4.4 Content Key Delivery with Usage Rules

4.4.4.1 Introduction

These use cases are extension of the use case of clause 4.4.2 and present different rules that can be applied on a Content Key when delivered to an entity as shown in Figure 6. 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.

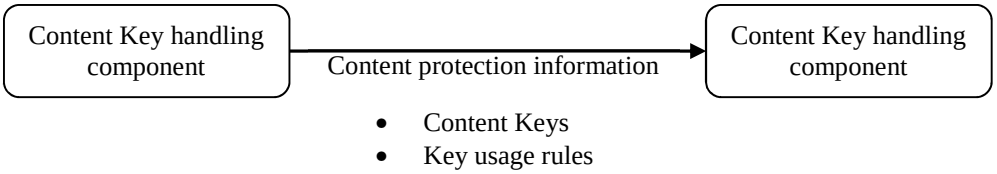

Figure 6: Content Key Delivery with usage rules

4.4.4.2 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 labelling 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.

4.4.4.3 Key Period Filter

This use case adds information to Content Keys that specifies how they are to be mapped to key periods, as known as cryptoperiods 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.

4.4.4.4 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.

4.4.5 Content Key Delivery with DRM Signaling

This use case is an extension of the use case of clause 4.4.2 and is compatible with the use case of clause 4.4.4.

This use case adds DRM Signaling information to each Content Key. The recipient may embed this signaling into the data streams it generates. See Figure 7.

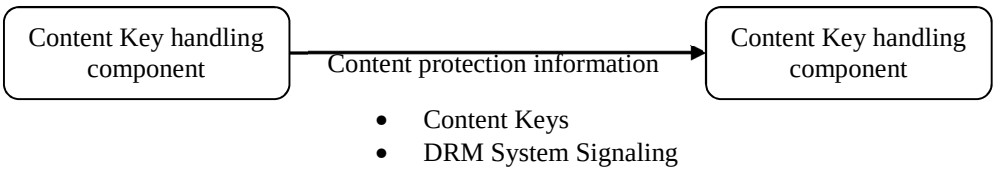

Figure 7: Content Key Delivery with DRM Signaling

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 ISO BMFF file.

The use of 3rd party extensions enables 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.

4.4.6 Incremental Update and Extension of the document

This use case, shown in Figure 8, 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.

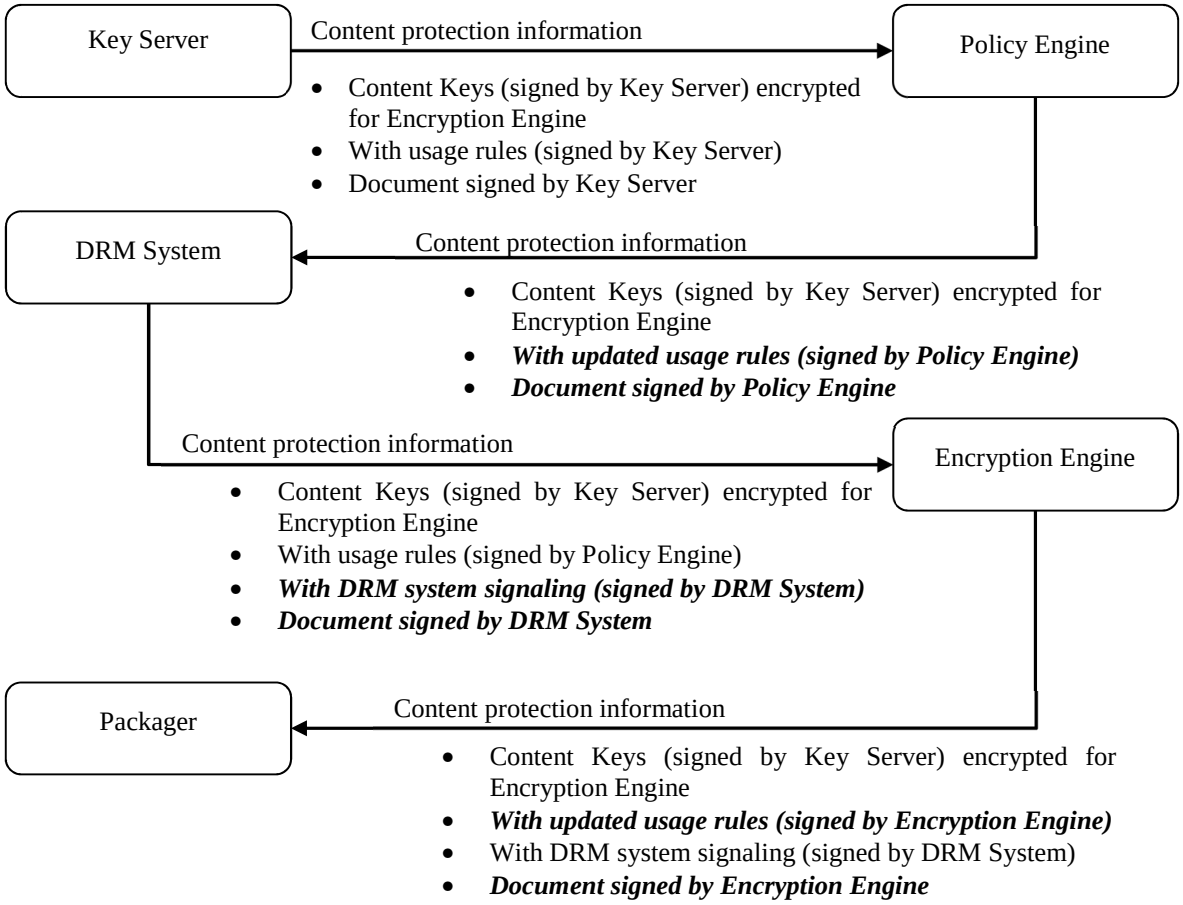

Figure 8: Incremental update and extension of the document

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.

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.

4.4.7 Multiple Content Keys Delivery for Multiples Assets

This use case, shown in Figure 9, is for the bulk transfer of Content Keys in one document. Each Content Key is associated to a different media asset, hence within the document, several media assets can be referenced. Limiting the number of documents to exchange allows for simpler transfer between entities of Content Keys and associated information such as usage rules and DRM signaling.

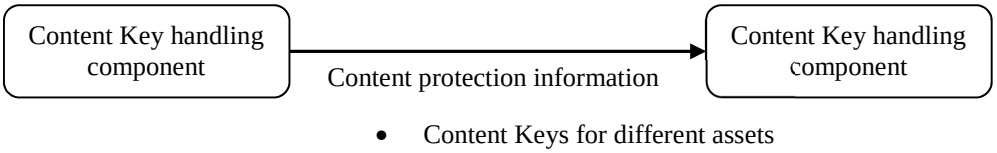

Figure 9: Bulk transfer of Content Keys referencing different assets

4.4.8 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 fulfil this use case, CPIX enables the above data to be exchanged.

4.4.9 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 clause 4.4.2. 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.

4.5 Workflow Examples

4.5.1 Encryptor Producer and Encryptor Consumer

4.5.1.1 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 in Figure 10 and Figure 11.

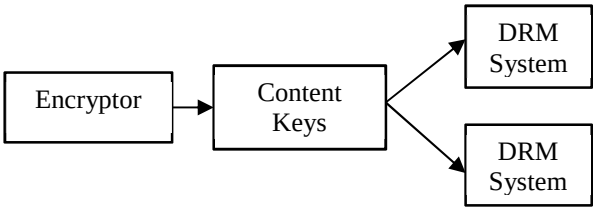

Figure 10: Encryptor Producer

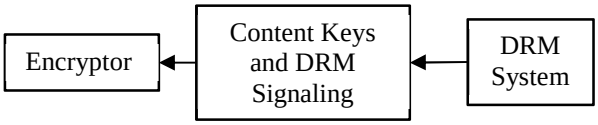

Figure 11: Encryptor Consumer

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 shall 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 the present document.

4.5.1.2 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 Figure 12.

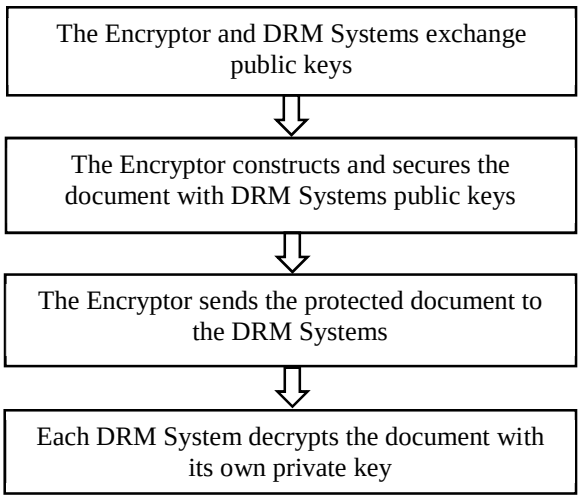

Figure 12: Encryptor Producer example steps

4.5.1.3 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 Encryptor 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 13.

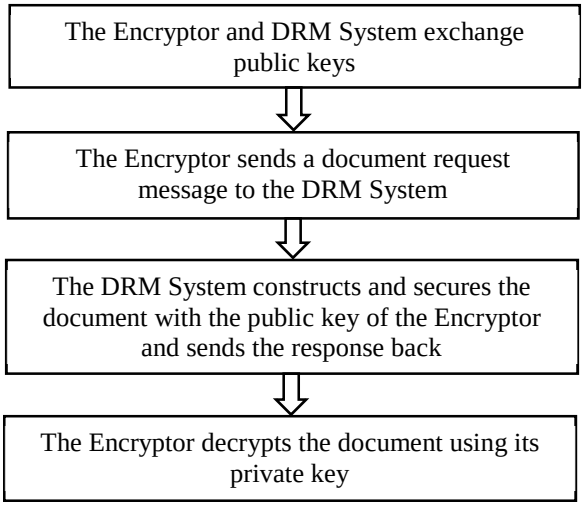

Figure 13: Encryptor Consumer example steps

4.5.2 Multiple Producers

This clause illustrates that it is possible to have more complex workflows than those previously illustrated. In one such example, shown in Figure 14, 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.

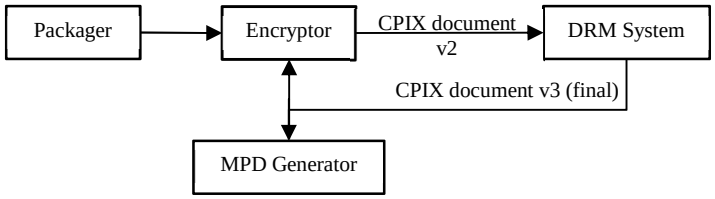

Figure 14: Multiple Producers example

- 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 Figure 15.

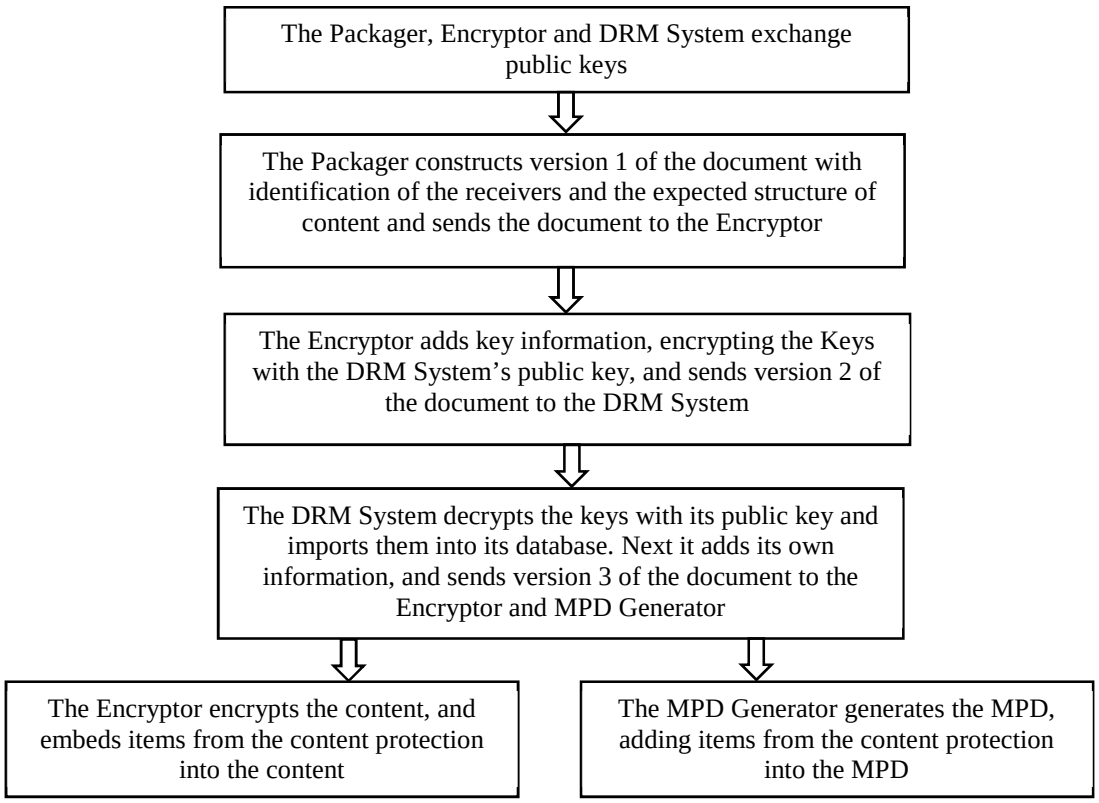

Figure 15: Multiple Producers example steps

5 XSD Schema Definition

5.1 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 4). This is an XML file that is described by the XSD. This clause describes in details elements part of the schema.

5.2 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 [1].

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

Entities exchanging content protection information should be authenticated.

5.3 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 [6] 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 16 shows the first elements and a high-level view of the structure. Detailed description of the structure is given in the following clauses.

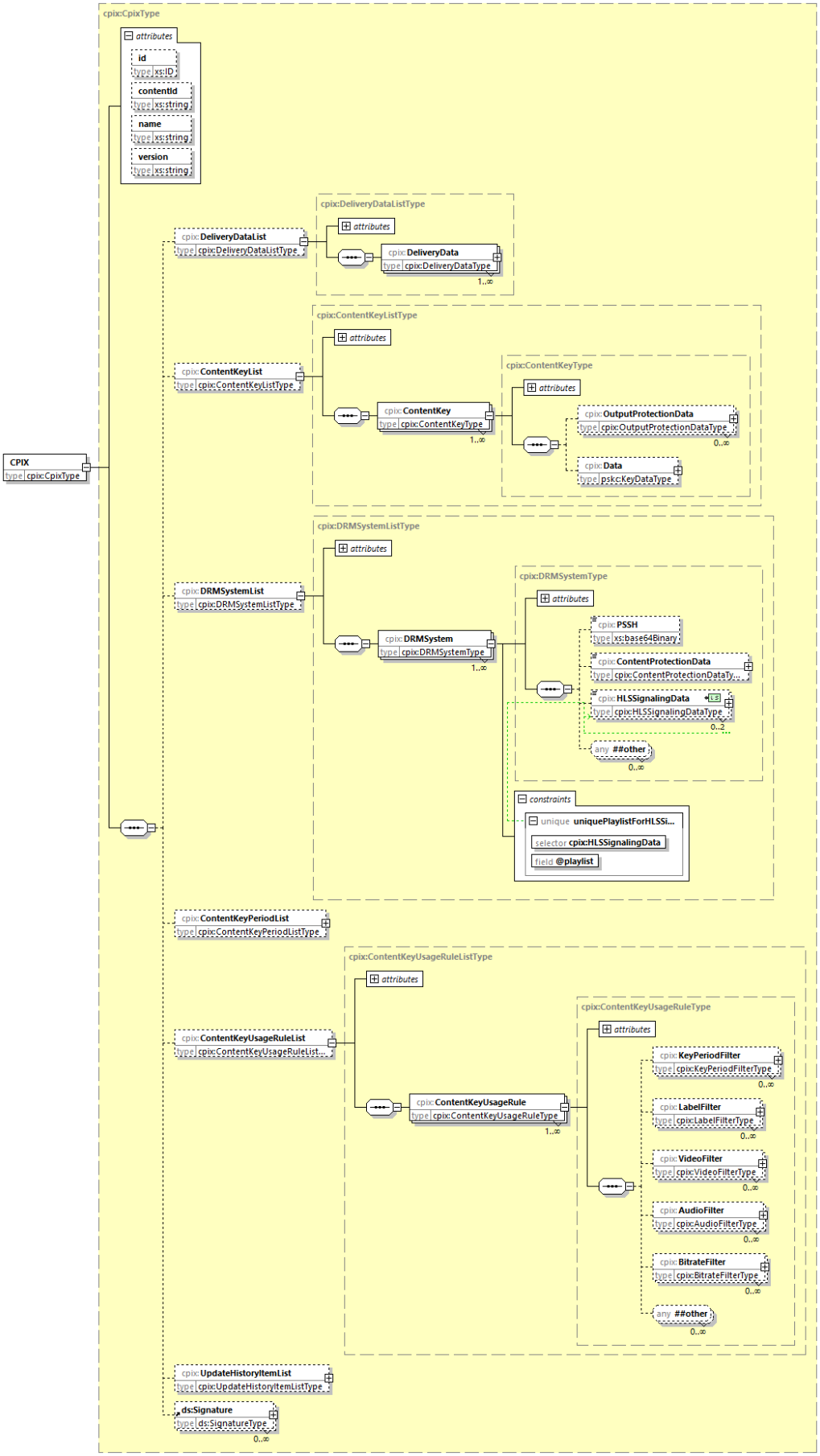

Figure 16: Content Protection Information Exchange Format high level view

5.4 Hierarchical Data Model

5.4.1 Introduction

In all tables of this clause, the following convention is used:

- Elements are in bold and the number of allowed instances is defined by <minOccurs>…<maxOccurs> (where N for <maxOccurs> means unbounded). Elements shall be in the order specified in this clause.
- Attributes are non-bold 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 in Annex A.

In addition to types defined in the present document that come with the prefix cpix:, the CPIX data model references types defined in [3], [4], [5] and [6]. External data types are prefixed with pskc:, xs:, xenc: and ds: respectively.

5.4.2 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 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. It is mutually exclusive with the attribute `@contentId` defined in the ContentKey element.

`@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 2.4.
>
> If a CPIX client does not support all the features of a given CPIX version, it shall behave according to the recommendations of the API used for exchanging the CPIX document.

DeliveryDataList (0...1, cpix:DeliveryDataList)

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

ContentKeyList (0...1, cpix:ContentKeyList)

> A container for ContentKey elements.

DRMSystemList (0...1, cpix:DRMSystemList)

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

ContentKeyPeriodList (0...1, cpix:ContentKeyPeriodList)

> A container for ContentKeyPeriod elements.

ContentKeyUsageRuleList (0...1, cpix: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, cpix:UpdateHistoryItemList)

A container for UpdateHistoryItem elements.

Signature (0...N, ds:Signature)

A digital signature as defined in [6]. 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 17 shows a graphical view of the element.

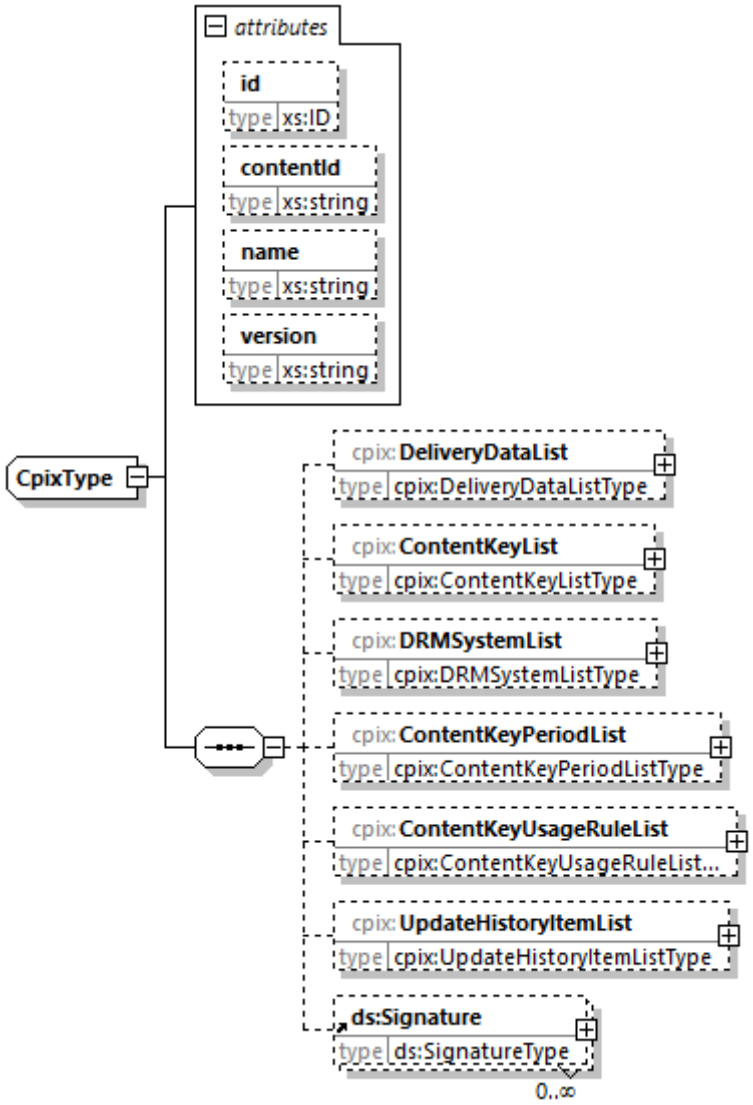

Figure 17: CPIX element

5.4.3 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 (1...N, cpix:DeliveryData)

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

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

Figure 18 shows a graphical view of the element.

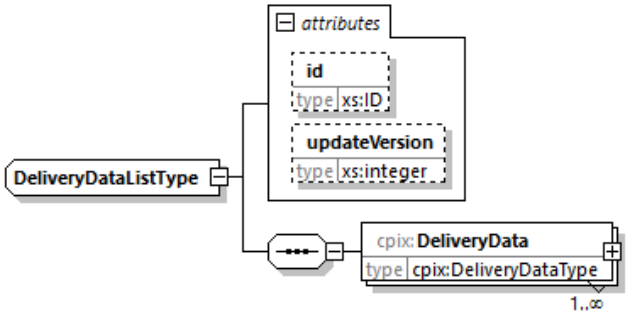

Figure 18: DeliveryDataList element

5.4.4 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 clause 6.1.2 for a description of the key management within the CPIX document.

DocumentKey (1...N, cpix:DocumentKeyType)

Contains the keys that are used for encrypting the Content Key stored in ContentKey elements.

MACMethod (0...1, pskc:MACMethodType)

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

Refer to clause 6.1 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)

Contact information, such as an email address, of the Sender.

ReceivingEntity (0...1, xs:string)

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

Figure 19 shows a graphical view of the element.

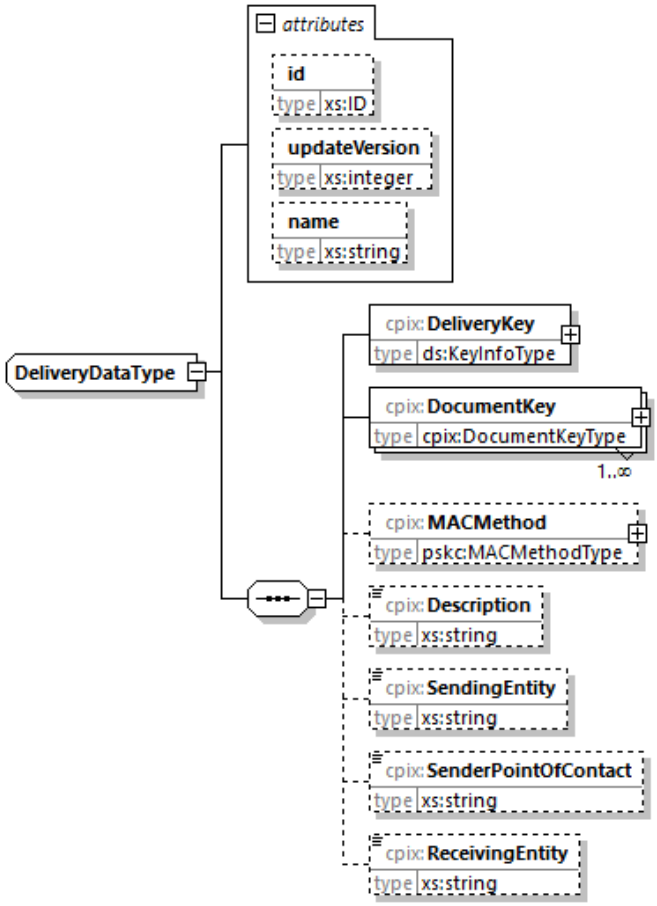

Figure 19: DeliveryData element

5.4.5 DocumentKey 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.

`@encryptsKey` (O, cpix:UUID)

Matches the `@kid` attribute(s) of the referenced ContentKey elements. These referenced Content Keys in the ContentKey element(s) are encrypted with the Document Key stored under the Data element. When a Document Key is used for encrypting several Content Keys, this attribute shall store a space-delimited list of those different `@kid` values.

If there are several DocumentKey elements, this attribute shall be present. If there is only one DocumentKey element, this attribute may not be present and, in this case, the Document Key encrypts all Content Keys.

Data (1, pskc:KeyDataType)

Contains the Document Key either in the clear or encrypted. The Document Keys are encrypted using the public key listed in the recipient's X.509 certificate from the DeliveryKey element.

Refer to clause 6.1 for a description of the key management within the CPIX document.

Figure 20 shows a graphical view of the element.

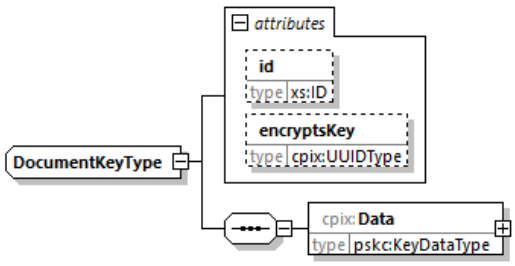

Figure 20: DocumentKey element

5.4.6 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 (1...N, cpix:ContentKey)

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

Figure 21 shows a graphical view of the element.

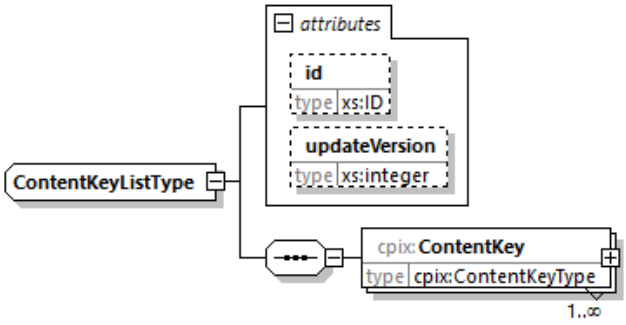

Figure 21: ContentKeyList element

5.4.7 ContentKey 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.

`@contentId` (O, xs:string)

An identifier for the asset or content that is protected by this key. It is mutually exclusive with the attribute `@contentId` defined in the CPIX element.

When present, every ContentKey element may have different or identical value. This attribute shall not be present if the `@dependsOnKey` attribute is present. In a key hierarchy, the root key defines this value for all keys in the hierarchy.

The use of this attribute is recommended only when exchanging multiples content keys that do not share the same `@contentId` value. That allows reducing the number of CPIX documents that need to be exchanged. See clause 4.4.7 for additional details.

`@kid` (M, cpix:UUIDType)

The unique identifier of the Content Key. It shall be formatted as defined in [7], 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 [7] or of the `IV` attribute under `#EXT-X-KEY` or `#EXT-X-SESSION-KEY` defined in [8] if content is delivered in HLS format.

Use of this attribute is not recommended except for compatibility with some DRM systems that explicitly need it, meaning when a "Constant IV" needs to be provided within a DRM license.

`@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.

Note all DRMs support key hierarchy, see clause 6.2 for more details.

`@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 in [7] or one of the encryption method defined in [8]. If the attribute is omitted, then content may be encrypted using any encryption scheme.

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

HDCPData (0...1, cpix:HDCPData)

Contains the HDCP information for this Content Key.

This attribute shall not be present if the `@dependsOnKey` attribute is present. In a key hierarchy, the root key defines the HDCP properties for all keys in the hierarchy.

Data (0...1, pskc:KeyDataType)

Contains the Content Key either in the clear or encrypted. If encrypted, the Content Key is encrypted with a key that is under a DocumentKey element.

Refer to clause 6.1 for a description of the key management within the CPIX document.

Figure 22 shows a graphical view of the element.

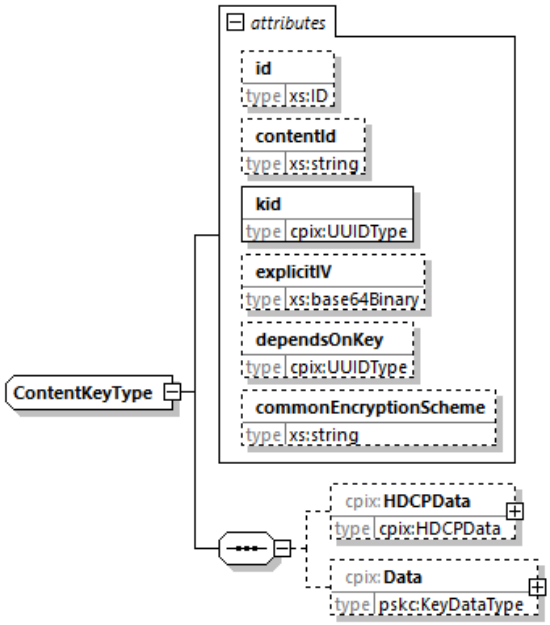

Figure 22: ContentKey element

5.4.8 HDCPData Element

`@HLSHDCPLevel` (O, xs:string)

This attribute specifies the value of the HDCP-LEVEL attribute of the EXT-X-STREAM-INF tag in the multiVariant playlist. Its format is as specified in clause 4.4.6.2 of [8].

This attribute has meaning only when an HLS playlist is created for the media content.

HDCPOutputProtectionData (0...1, xs:base64binary)

This is the full well-formed standalone XML fragment to be added to the DASH manifest for the HDCP OutputProtection element for this Content Key. This is UTF-8 text without a byte order mark. See in [1], clause 7.4 for more details.

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

Figure 23 shows a graphical view of the element.

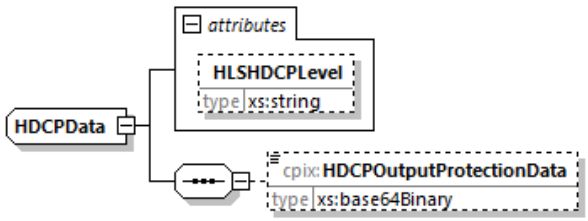

Figure 23: OutputProtectionData element

5.4.9 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 (1...N, cpix:DRMSystem)

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

Figure 24 shows a graphical view of the element.

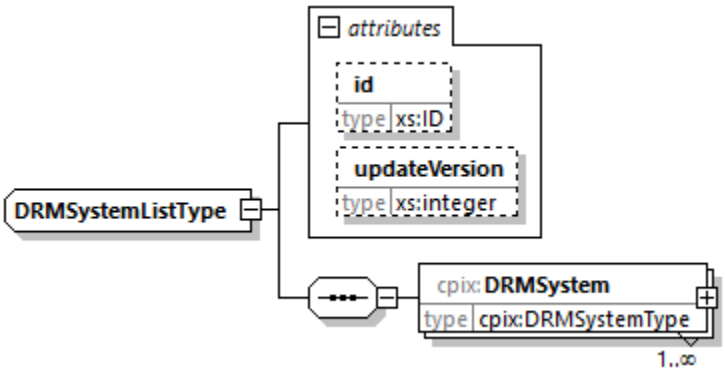

Figure 24: DRMSystemList element

5.4.10 DRMSystem Element

The DRMSystem element contains all information on a DRM system that can be used for retrieving licenses for getting access to content. The present document defines elements for DRM system signaling in DASH, ISO BMFF, Smooth Streaming and HLS 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 Common Encryption 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 at [2].

`@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.

`@HLSAllowedCPC` (O, xs:string)

This attribute specifies, for the DRM identified by the `@systemId` value, the value to be added in the ALLOWED-CPC attribute of the EXT-X-STREAM-INF tag in the multiVariant playlist. Its format is as specified in clause 4.4.6.2 of [8].

The final value of the ALLOWED-CPC is the concatenation, each separated by a comma, of all `@HLSAllowedCPC` values present in the CPIX document, except if ContentKey elements have a `@contentId` value. In this latter case, the concatenation is limited to those that have a referenced ContentKey element with the same `@contentId` value.

This attribute has meaning only when an HLS playlist is created for the media content.

If the referenced ContentKey element includes a `@dependsOnKey` attribute, this element shall not be present.

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 includes a `@dependsOnKey` attribute, the value shall be inserted under the `moof` box.

If the referenced ContentKey element does not include a `@dependsOnKey` attribute, the value may be inserted under the `moov` box or the `moof` box. In this case, see [1], clause 6 for more details.

ContentProtectionData (0...1, cpix:ContentProtectionData)

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 [1], in this case, all `dashif:xxx` elements are children of the ContentProtection element and are therefore provided in this element.

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

If the referenced ContentKey element includes a `@dependsOnKey` attribute, this element shall not be present.

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

HLSSignalingData (0...2, cpix:HLSSignalingData)

This is the full data including the `#EXT-X-KEY` or `#EXT-X-SESSION-KEY` tag of an HLS playlist [8] depending on the destination of the data (see clause 5.4.10). 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 has meaning only when an HLS playlist is created for the media content.

If the referenced ContentKey element includes a `@dependsOnKey` attribute, this element shall not be present.

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

If the referenced ContentKey element includes a `@dependsOnKey` attribute, this element shall not be present.

Additional child elements may be present containing signaling data for other media formats. Such elements shall appear after any elements defined here.

Figure 25 shows a graphical view of the element.

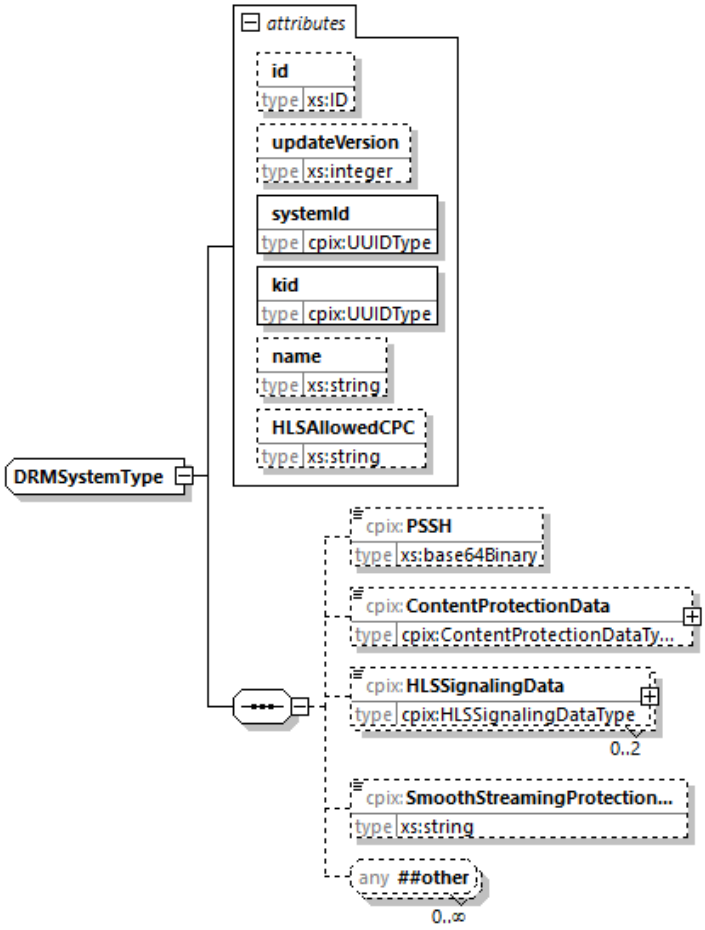

Figure 25: DRMSystem element

5.4.11 ContentProtectionData Element

The ContentProtectionData shall be base64 encoded text. It has an optional attribute allowing to define the robustness level that is expected for this DRM.

`@robustness` (O, xs:string)

> The value of this attribute is DRM specific. It announces what robustness level is expected from the DRM system for the representations that are encrypted by the referenced Content Key.
>
> This is the value of the `@robustness` attribute of the ContentProtection element in the DASH manifest for this DRM system.

Figure 26 shows a graphical view of the element.

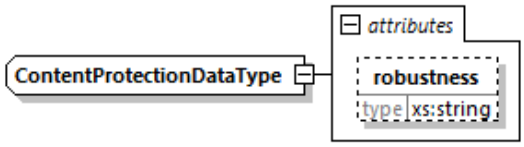

Figure 26: ContentProtectionData element

5.4.12 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 multiVariant 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 multiVariant 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 `multiVariant` 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 27 shows a graphical view of the element.

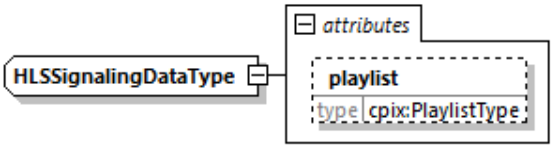

Figure 27: HLSSignalingData element

5.4.13 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 (1...N, cpix:ContentKeyPeriod)

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

Figure 28 shows a graphical view of the element.

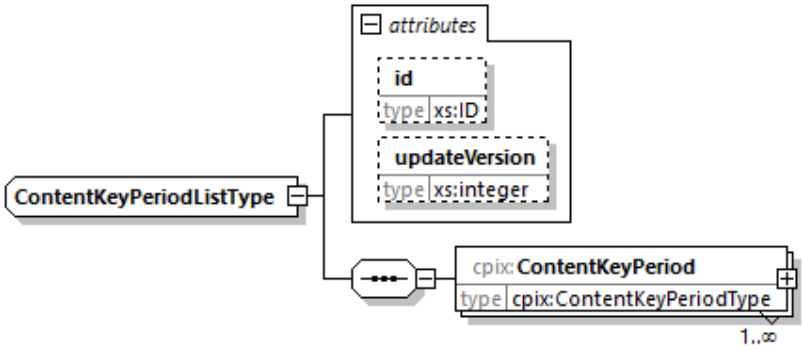

Figure 28: ContentKeyPeriodList element

5.4.14 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. It shall increase. When reaching MAX_UINT32, the value rolls over.

`@label` (O, xs:string)

String identifier for the key period. As an example, the value of this attribute may be used to match a SCTE-35 `segmentation_event_id`, in this case, it allows matching this content key to a specific program.

`@start` (O, xs:dateTime)

For Live content, this is the wall clock time for the start time for the period.

`@end` (O, xs:dateTime)

For Live content, this is the wall clock time for the end time for the period. Mutually exclusive with `@duration`.

`@startOffset` (O, xs:duration)

For VOD content, this is the start time for the period.

`@endOffset` (O, xs:duration)

For VOD content, this is the end time for the period. Mutually exclusive with `@duration`.

`@duration` (O, xs:duration)

For VOD and Live content, this is the duration for the period. Mutually exclusive with `@end` and `@endOffset`.

The valid combinations of attributes that are time-based are:

- `@start` and `@end` are present, the interval is defined by [`@start`, `@end`).
- `@start` and `@duration` are present, the interval is defined by [`@start`, `@start+@duration`).
- `@startOffset` and `@endOffset` are present, the interval is defined by [`@startOffset`, `@endOffset`).
- `@startOffset` and `@duration` are present, the interval is defined by [`@startOffset`, `@startOffset+@duration`).

If none of these combinations is specified, then 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`) or a string index (`@label`). An example of this use of `@index` would be an encryptor which rotates the keys once an hour, and not necessarily at specific times.

Figure 29 shows a graphical view of the element.

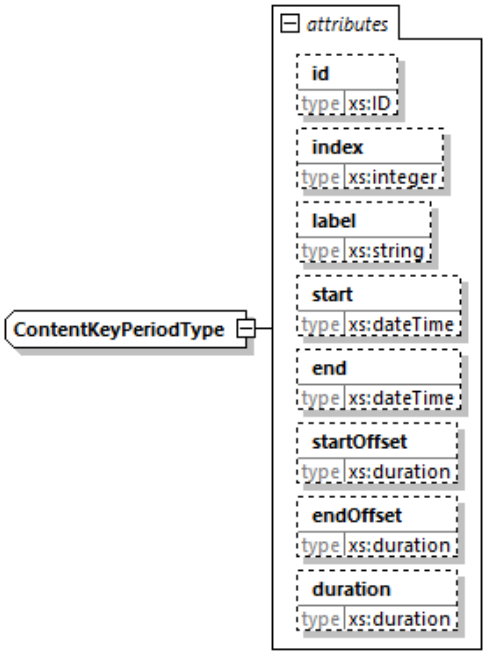

Figure 29: ContentKeyPeriod element

5.4.15 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 (1...N, cpix:ContentKeyUsageRule)

A rule which defines a Content Key Context.

Figure 30 shows a graphical view of the element.

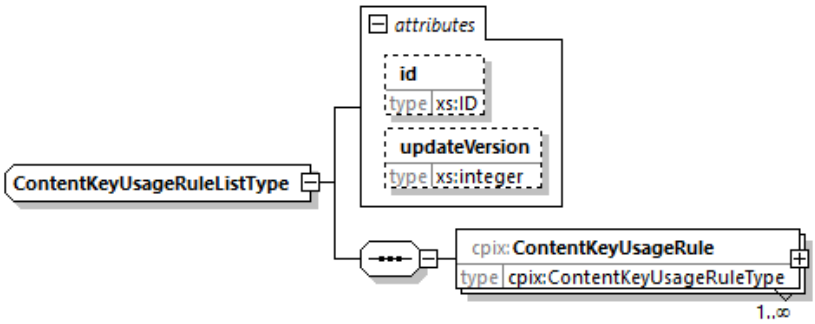

Figure 30: ContentKeyUsageRuleList element

5.4.16 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:UUIDType)

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 clause 5.4.17.3 for more details.

KeyPeriodFilter (0...N, cpix:KeyPeriodFilter)

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

This filter links ContentKey and ContentKeyPeriod elements.

LabelFilter (0...N, cpix:LabelFilter)

Defines a label association for the Content Key Context.

VideoFilter (0...N, cpix: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, cpix: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, cpix:BitrateFilter)

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

Additional child elements may be present containing proprietary filters. Such elements shall appear after any elements defined here.

Figure 31 shows a graphical view of the element.

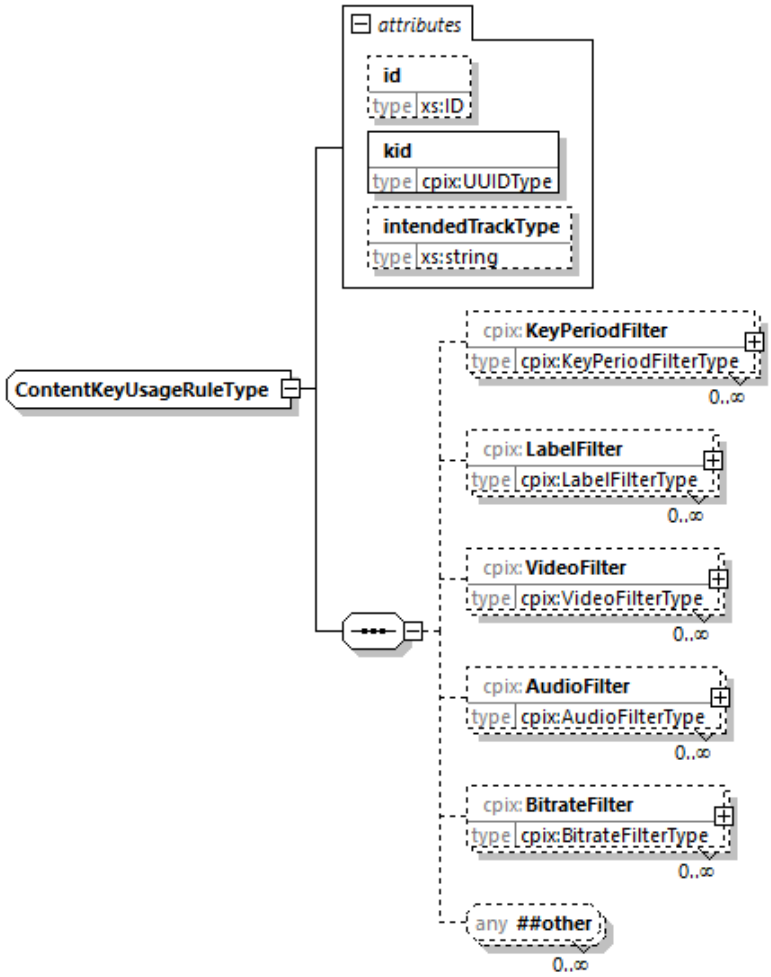

Figure 31: ContentKeyUsageRule element

5.4.17 Usage Rules Filters

5.4.17.1 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` 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.

5.4.17.2 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 32 shows a graphical view of the element.

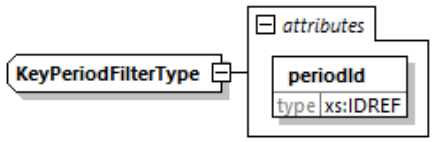

Figure 32: KeyPeriodFilter element

5.4.17.3 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 33 shows a graphical view of the element.

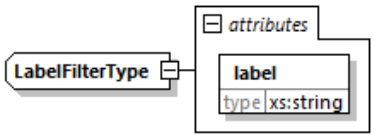

Figure 33: LabelFilter element

The `@label` attribute is meant for triggering a particular ContentKeyUsageRule by using pre-agreed upon label strings. Its value may correspond to media track types but it needs not. 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 needs not to 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`.

5.4.17.4 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 `@minPixels` 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 34 shows a graphical view of the element.

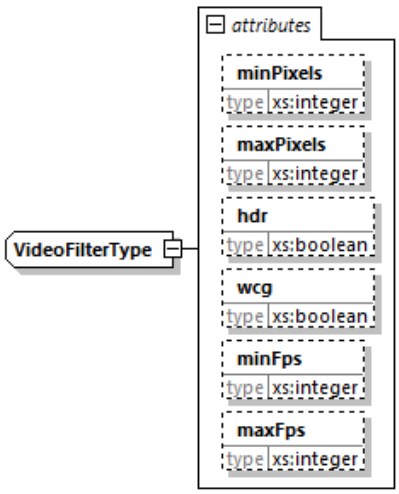

Figure 34: VideoFilter element

5.4.17.5 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 35 shows a graphical view of the element.

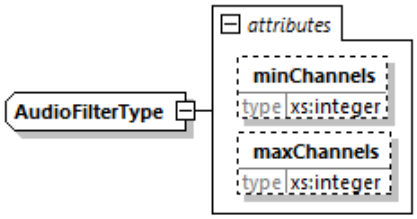

Figure 35: AudioFilter element

5.4.17.6 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 36 shows a graphical view of the element.

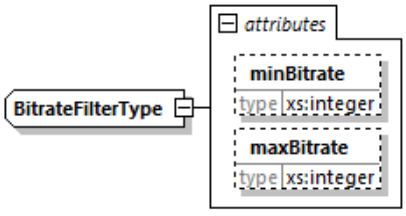

Figure 36: BitrateFilter element

5.4.18 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 (1...N, cpix: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 37 shows a graphical view of the element.

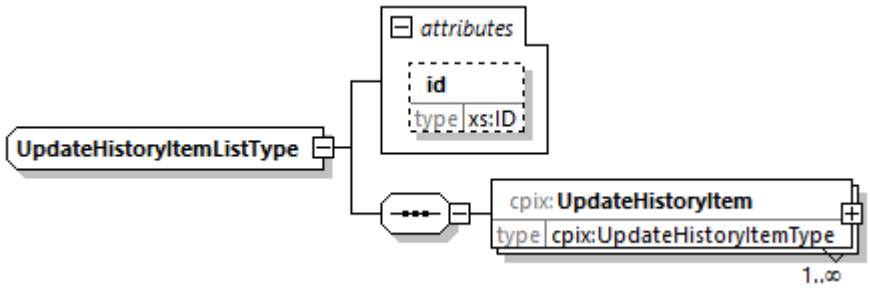

Figure 37: UpdateHistoryList element

5.4.19 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` (O, xs:integer)

The 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 38 shows a graphical view of the element.

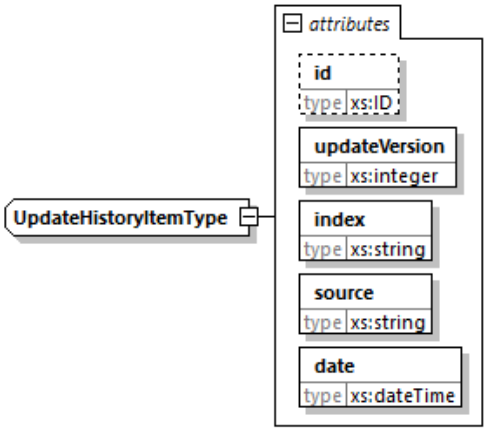

Figure 38: UpdateHistory element

6 Key Management

6.1 Key Encryption and Authentication in the CPIX Document

6.1.1 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 and authentication of the Content Keys in the document is used.

6.1.2 Encryption

The document contains the following keys for encrypting Content 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 [1], these keys are 128-bit keys used with the AES cipher.

Document Keys

For every CPIX document, one or several Document Keys may be created. It is used for encrypting Content Keys. These Document Keys are 256-bit key and the encryption algorithm used for encrypting every Content Key is AES. These are part of each DeliveryData element. These are encrypted in the document, using the public key of recipients.

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 Document Keys using an algorithm that is described within the CPIX document, according to [5].

Figure 39 gives the schema of encryption of the different keys when there are several DeliveryData elements, one DocumentKey element 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.

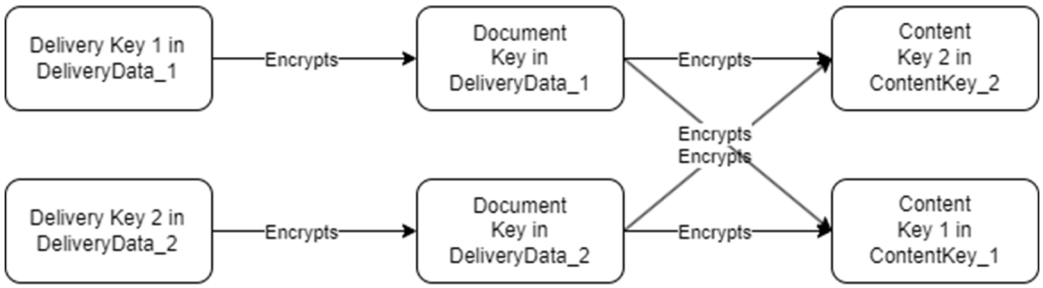

Figure 39: Encryption relationships within the CPIX document with one Document Key

Figure 40 gives the schema of encryption of the different keys when there are several DeliveryData elements, several DocumentKey elements and several ContentKey elements. In this example, the recipient identified in "DeliveryData_2" is entitled to access a subset of the Content Keys that are in the CPIX document while the recipient identified in "DeliveryData_1" is entitled to access another subset of the Content Keys.

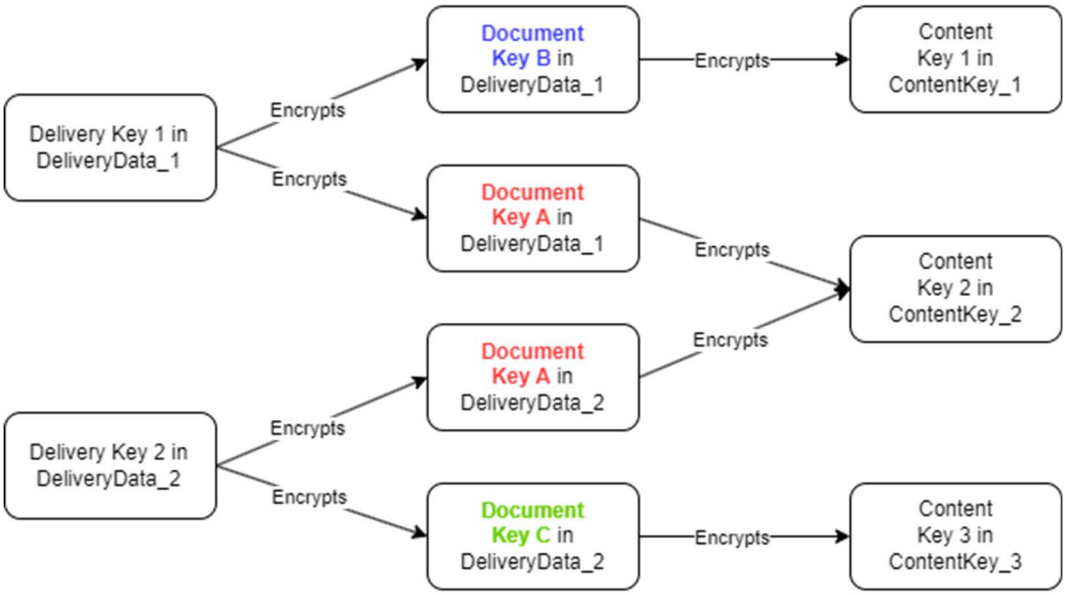

Figure 40: Encryption relationships within the CPIX document with several Document Keys

6.1.3 Authenticated Encryption

The document contains the following key for authenticating keys:

MAC Key

> For every CPIX document, a MAC Key may be 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.

For authenticated encryption of Content Keys, every encrypted Content Key shall have a MAC value and it shall be verified 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.

6.1.4 Digital Signature

Every element in the document that has an `@id` attribute can be signed according to [6]. 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. 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 the present document).

6.1.5 Mandatory Algorithms

Table 1 gives the identification of the algorithms that shall be used for encryption, signature, MAC creation.

Table 1: List of algorithms for the different security usages

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 1.1 (omits comments)

For RSA, the recommended minimum key size is 3 072 bits and is it not recommended to use certificates that are signed using SHA-1.

6.2 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 a well-formed ContentKeyPeriod element as described in clause 5.4.14.

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 [1] 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 5.4.7.

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: Not all DRM systems support key hierarchy.

Content encrypted using key rotation may also have periods in the clear that are not encrypted. In this case, there is no Content Key for these periods. A CPIX document will carry no specific information allowing to know that some periods are not encrypted. The information that content is encrypted or not would be known by either looking at the playlist/manifest or any other specific deployment source of data (such as EPG, a metadata server).

6.3 Content Keys with Several Protection Encryption Schemes

In [7], 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` of the ContentKey element which will take a different value depending on the protection scheme. In addition, depending on the DRM system, some elements under the DRMSystem element may also be different.

Annex A (normative): CPIX XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:cpix="urn:dashif:org:cpix" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" targetNamespace="urn:dashif:org:cpix"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
   <xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-
core-schema.xsd"/>
   <xs:import namespace="urn:ietf:params:xml:ns:keyprov:pskc"
schemaLocation="pskc.xsd"/>
   <xs:simpleType name="UUIDType">
      <xs:restriction base="xs:string">
         <xs:pattern value="[A-Fa-f0-9]{8}-[A-Fa-f0-9]{4}-[A-Fa-f0-9]{4}-[A-Fa-f0-9]{4}-
[A-Fa-f0-9]{12}"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="PlaylistType" final="restriction">
      <xs:restriction base="xs:string">
         <xs:enumeration value="multiVariant"/>
         <xs:enumeration value="media"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="UpdateHistoryItemType">
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="required"/>
      <xs:attribute name="index" type="xs:string" use="required"/>
      <xs:attribute name="source" type="xs:string" use="required"/>
      <xs:attribute name="date" type="xs:dateTime" use="required"/>
   </xs:complexType>
   <xs:complexType name="UpdateHistoryItemListType">
      <xs:sequence>
         <xs:element name="UpdateHistoryItem" type="cpix:UpdateHistoryItemType"
minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
   </xs:complexType>
   <xs:complexType name="KeyPeriodFilterType">
      <xs:attribute name="periodId" type="xs:IDREF" use="required"/>
   </xs:complexType>
   <xs:complexType name="LabelFilterType">
      <xs:attribute name="label" type="xs:string" use="required"/>
   </xs:complexType>
   <xs:complexType name="VideoFilterType">
      <xs:attribute name="minPixels" type="xs:integer" use="optional"/>
      <xs:attribute name="maxPixels" type="xs:integer" use="optional"/>
      <xs:attribute name="hdr" type="xs:boolean" use="optional"/>
      <xs:attribute name="wcg" type="xs:boolean" use="optional"/>
      <xs:attribute name="minFps" type="xs:integer" use="optional"/>
      <xs:attribute name="maxFps" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="AudioFilterType">
      <xs:attribute name="minChannels" type="xs:integer" use="optional"/>
      <xs:attribute name="maxChannels" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="BitrateFilterType">
      <xs:attribute name="minBitrate" type="xs:integer" use="optional"/>
```

```
      <xs:attribute name="maxBitrate" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="ContentKeyUsageRuleType">
      <xs:sequence>
         <xs:element name="KeyPeriodFilter" type="cpix:KeyPeriodFilterType"
minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="LabelFilter" type="cpix:LabelFilterType" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="VideoFilter" type="cpix:VideoFilterType" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="AudioFilter" type="cpix:AudioFilterType" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:element name="BitrateFilter" type="cpix:BitrateFilterType" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="kid" type="cpix:UUIDType" use="required"/>
      <xs:attribute name="intendedTrackType" type="xs:string" use="optional"/>
   </xs:complexType>
   <xs:complexType name="ContentKeyUsageRuleListType">
      <xs:sequence>
         <xs:element name="ContentKeyUsageRule" type="cpix:ContentKeyUsageRuleType"
minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="ContentKeyPeriodType">
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="index" type="xs:integer" use="optional"/>
      <xs:attribute name="label" type="xs:string" use="optional"/>
      <xs:attribute name="start" type="xs:dateTime" use="optional"/>
      <xs:attribute name="end" type="xs:dateTime" use="optional"/>
      <xs:attribute name="startOffset" type="xs:duration" use="optional"/>
      <xs:attribute name="endOffset" type="xs:duration" use="optional"/>
      <xs:attribute name="duration" type="xs:duration" use="optional"/>
   </xs:complexType>
   <xs:complexType name="ContentKeyPeriodListType">
      <xs:sequence>
         <xs:element name="ContentKeyPeriod" type="cpix:ContentKeyPeriodType"
minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="HLSSignalingDataType">
      <xs:simpleContent>
         <xs:extension base="xs:base64Binary">
            <xs:attribute name="playlist" type="cpix:PlaylistType" use="optional"/>
            <xs:attribute name="allowedCPC" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="ContentProtectionDataType">
      <xs:simpleContent>
         <xs:extension base="xs:base64Binary">
            <xs:attribute name="robustness" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
```

```
   </xs:complexType>
   <xs:complexType name="DRMSystemType">
      <xs:sequence>
         <xs:element name="PSSH" type="xs:base64Binary" minOccurs="0" maxOccurs="1"/>
         <xs:element name="ContentProtectionData" type="cpix:ContentProtectionDataType"
minOccurs="0" maxOccurs="1"/>
         <xs:element name="HLSSignalingData" type="cpix:HLSSignalingDataType"
minOccurs="0" maxOccurs="2"/>
         <xs:element name="SmoothStreamingProtectionHeaderData" type="xs:string"
minOccurs="0"/>
         <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
      <xs:attribute name="systemId" type="cpix:UUIDType" use="required"/>
      <xs:attribute name="kid" type="cpix:UUIDType" use="required"/>
      <xs:attribute name="name" type="xs:string" use="optional"/>
      <xs:attribute name="HLSAllowedCPC" type="xs:string" use="optional"/>
   </xs:complexType>
   <xs:complexType name="DRMSystemListType">
      <xs:sequence>
         <xs:element name="DRMSystem" type="cpix:DRMSystemType" minOccurs="1"
maxOccurs="unbounded">
            <xs:unique name="uniquePlaylistForHLSSignalingData">
               <xs:selector xpath="cpix:HLSSignalingData"/>
               <xs:field xpath="@playlist"/>
            </xs:unique>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="HDCPData">
      <xs:sequence>
         <xs:element name="HDCPOutputProtectionData" type="xs:base64Binary"
minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="HLSHDCPLevel" type="xs:string" use="optional"/>
   </xs:complexType>
   <xs:complexType name="ContentKeyType">
      <xs:sequence>
         <xs:element name="HDCPData" type="cpix:HDCPData" minOccurs="0" maxOccurs="1"/>
         <xs:element name="Data" type="pskc:KeyDataType" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="contentId" type="xs:string" use="optional"/>
      <xs:attribute name="kid" type="cpix:UUIDType" use="required"/>
      <xs:attribute name="explicitIV" type="xs:base64Binary" use="optional"/>
      <xs:attribute name="dependsOnKey" type="cpix:UUIDType" use="optional"/>
      <xs:attribute name="commonEncryptionScheme" type="xs:string" use="optional"/>
   </xs:complexType>
   <xs:complexType name="ContentKeyListType">
      <xs:sequence>
         <xs:element name="ContentKey" type="cpix:ContentKeyType" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
   </xs:complexType>
   <xs:complexType name="DocumentKeyType">
```

```
      <xs:sequence>
        <xs:element name="Data" type="pskc:KeyDataType" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="encryptsKey" type="cpix:UUIDType" use="optional"/>
    </xs:complexType>
    <xs:complexType name="DeliveryDataType">
      <xs:sequence>
        <xs:element name="DeliveryKey" type="ds:KeyInfoType" minOccurs="1"
maxOccurs="1"/>
        <xs:element name="DocumentKey" type="cpix:DocumentKeyType" minOccurs="1"
maxOccurs="unbounded"/>
        <xs:element name="MACMethod" type="pskc:MACMethodType" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="SendingEntity" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="SenderPointOfContact" type="xs:string" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="ReceivingEntity" type="xs:string" minOccurs="0"
maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
      <xs:attribute name="name" type="xs:string" use="optional"/>
    </xs:complexType>
    <xs:complexType name="DeliveryDataListType">
      <xs:sequence>
        <xs:element name="DeliveryData" type="cpix:DeliveryDataType" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="updateVersion" type="xs:integer" use="optional"/>
    </xs:complexType>
    <xs:complexType name="CpixType">
      <xs:sequence>
        <xs:element name="DeliveryDataList" type="cpix:DeliveryDataListType"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="ContentKeyList" type="cpix:ContentKeyListType" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="DRMSystemList" type="cpix:DRMSystemListType" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="ContentKeyPeriodList" type="cpix:ContentKeyPeriodListType"
minOccurs="0" maxOccurs="1"/>
        <xs:element name="ContentKeyUsageRuleList"
type="cpix:ContentKeyUsageRuleListType" minOccurs="0" maxOccurs="1"/>
        <xs:element name="UpdateHistoryItemList" type="cpix:UpdateHistoryItemListType"
minOccurs="0" maxOccurs="1"/>
        <xs:element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="optional"/>
      <xs:attribute name="contentId" type="xs:string" use="optional"/>
      <xs:attribute name="name" type="xs:string" use="optional"/>
      <xs:attribute name="version" type="xs:string" use="optional"/>
    </xs:complexType>
    <xs:element name="CPIX" type="cpix:CpixType"/>
</xs:schema>
```

History

Version	Date	Status
V1.1.1	April 2021	Publication
V1.2.1	April 2026	Publication