ETSI TS 102 474 V1.1.1 (2007-11)

Technical Specification

Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Service Purchase and Protection

European Broadcasting Union

Union Européenne de Radio-Télévision

EBU·UER

Reference
DTS/JTC-DVB-190

Keywords
broadcasting, data, digital, DVB, IP, 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 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced 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

Founded in September 1993, the DVB Project is a market-led consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

Introduction

IP Datacast over DVB-H is an end-to-end broadcast system for delivery of any types of digital content and services using IP-based mechanisms optimized for devices with limitations on computational resources and battery. An inherent part of the IP Datacast system is that it comprises a unidirectional DVB broadcast path that may be combined with a bi-directional mobile/cellular interactivity path. IP Datacast is thus a platform that can be used for enabling the convergence of services from broadcast/media and telecommunications domains (e.g. mobile/cellular).

1 Scope

The present document defines the set of specification documents applicable to Service Purchase and Protection for IP Datacast services over DVB-H.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at *http://docbox.etsi.org/Reference*.

For online referenced documents, information sufficient to identify and locate the source shall be provided. Preferably, the primary source of the referenced document should be cited, in order to ensure traceability. Furthermore, the reference should, as far as possible, remain valid for the expected life of the document. The reference shall include the method of access to the referenced document and the full network address, with the same punctuation and use of upper case and lower case letters.

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

2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] ETSI TS 103 197: "Digital Video Broadcasting (DVB); Head-end implementation of DVB SimulCrypt".

[2] ISO/IEC 11770-3: "Information technology - Security techniques - key management - Part 3: Mechanisms using asymmetric techniques".

[3] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

[4] ETSI TS 102 221: "Smart-cards; UICC-Terminal interface; Physical and logical characteristics".

[5] ETSI TS 101 220: "Smart-cards; ETSI numbering system for telecommunication application providers".

[6] ETSI TS 131 101: "Universal Mobile Telecommunications System (UMTS); UICC-terminal interface; Physical and logical characteristics (3GPP TS 31.101 version 6.5.1 Release 6)".

[7] ISO/IEC 7816-4: "Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange".

[8] OMA-TS-DRM-DCF-V2-0-20050901-C: "DRM Content Format; Candidate Version 2.0".

[9] IETF RFC 3280: "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[10] IETF RFC 3447: "Public-Key Cryptography Standards (PKCS)#1; RSA Cryptography Specifications Version 2.1". .

[11] DRM: "ISMA Encryption and Authentication".

[12] ISO/IEC 14496-12: "Information technology - Coding of audio-visual objects - Part 12: ISO base media file format".

[13] NIST (2001): "AES Key Wrap Specification".

[14] ETSI EN 300 468: "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

[15] ETSI EN 301 192: "Digital Video Broadcasting (DVB); DVB specification for data broadcasting".

[16] CENELEC EN 50094: "Access control system for the MAC/packet family: EUROCRYPT".

[17] FIPS PUB 197 (2001): "Advanced Encryption Standard (AES)".

[18] ISO 639 (2002): "Codes for the representation of names of languages -- Part 1: Alpha-2 code".

[19] ISO 3166: "Codes for the representation of names of countries and their subdivisions".

[20] ISO 4217: "Codes for the representation of currencies and funds".

[21] "PKCS #1 v2.1: RSA Cryptography Standard", RSA Laboratories, June 14, 2002".

[22] IETF RFC 768: "User Datagram Protocol".

[23] IETF RFC 2104: "HMAC: Keyed-Hashing for Message Authentication".

[24] IETF RFC 4301: "Security Architecture for the Internet Protocol".

[25] IETF RFC 4305: "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)".

[26] IETF RFC 4303: "IP Encapsulating Security Payload (ESP)".

[27] IETF RFC 2451: "The ESP CBC-Mode Cipher Algorithms".

[28] IETF RFC 3566: "The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec".

[29] IETF RFC 3602: "The AES-CBC Cipher Algorithm and Its Use with IPsec".

[30] IETF RFC 3629: "UTF-8, a transformation format of ISO 10646".

[31] IETF RFC 3664: "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE) ".

[32] IETF RFC 3711: "The Secure Real-time Transport Protocol (SRTP)".

[33] OMA-TS-DRM-DRM-V2-0-20060303-A: "DRM Specification", Approved Version 2.0.

[34] IETF RFC 3388: "Grouping of Media Lines in the Session Description Protocol (SDP)".

[35] ETSI TS 102 471: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Electronic Service Guide (ESG)".

[36] ETSI TS 102 472: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Content Delivery Protocols".

[37] ITU-T Recommendation X.509: "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks".

[38] IETF draft-lehtovirta-srtp-rcc-06.txt (October 2006): "Integrity Transform Carrying Roll-over Counter".

2.2 Informative references

[39] DVB: "_CA systems".

NOTE: http://www.dvb.org/products_registration/dvb_identifiers/ca_systems/.

[40] JSR 118: "Mobile Information Device Profile 2.0", Sun Microsystems.

[41] JSR 177: "Security and Trust Services API for J2METM", Sun Microsystems.

[42] JSR 211: "Content Handler API", Sun Microsystems.

[43] Amos Fiat and Moni Naor: "Broadcast encryption", 1998.

[44] FIPS PUB 180-2 (2002): "Secure Hash Standard (SHS)".

[45] FIPS PUB 198 (2002): "The Keyed-Hash Message Authentication Code (HMAC)".

[46] NIST Special Publication 800-38A (2001): "Recommendation for Block Cipher Modes of Operation Methods and Techniques". .

[47] IrDA: "Object Exchange (OBEX) Protocol v1.3".

[48] OMA-AD-DRM-V2-0-20060303-A: "DRM Architecture", Approved Version 2.0.

[49] OMA-ERP-DRM-V2-0-20060303-A: "OMA Digital Rights Management V2.0" Approved Enabler.

[50] OMA-TS-DRM-REL-V2-0-20060303-A: "DRM Rights Expression Language", Approved Version V2.0".

[51] Bruce Schneier: "Applied Cryptography: Protocols, Algorithms, and Source Code in C", John Wiley & Sons, Inc, 1996.

[52] Verhoef J.: "Error detecting decimal codes (Mathematical Centre Tract 29)", The mathematical Centre, Amsterdam, 1969.

[53] DVB A094 (2005): "Digital Video Broadcasting (DVB); Content Protection & Copy Management".

[54] Java 2 Platform, Micro Edition: "Connected Limited Device Configuration (CLDC) Version 1.1".

[55] ISO/IEC 14496-10: "Information technology - Coding of audio-visual objects - Part 10: Advanced Video Coding".

[56] Java 2 Platform, Micro Edition: "Connected Device Configuration (CDC) Version 1.0".

NOTE: http://java.sun.com/products/cdc/index.jsp.

[57] ISO 2015 (1976): "Numbering of weeks" (withdrawn standard).

3 Definitions, symbols and abbreviations

3.1 Definitions

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

broadcast channel: See broadcast network.

broadcast Device: Device which is capable of receiving protected broadcasts but which cannot access an interactivity channel

NOTE: All messages sent to such a Device are delivered via the broadcast channel, and all communication from the Device to Rights Issuers is done out of band.

broadcast domain: group of Devices for which Rights Objects granting the same rights to the whole group can be issued via the broadcast channel

broadcast network: digital transmission supporting only unidirectional communication from the broadcaster to the Device

broadcast Rights Object: content, programme or service Rights Object delivered over the broadcast network

broadcast service: service carried over a broadcast network

broadcast: unidirectional distribution to all receivers

conditional access: Refers to the same as DRM.

content protection: protection of content such that it can only be presented by authorized Devices

Content Rights Object: Rights Object containing a content key and pertaining to content protection

content: any form of digital media which can be acquired and presented by a Device

Customer Operation Centre: entity responsible for billing the customer

NOTE: It is assumed that the customer will have a contractual relationship with this organization.

Device: terminal that is capable of receiving IP Datacast transmission over DVB-H network

NOTE: The terminal may be capable of interacting with service provision sub-system and commerce sub-system through a cellular network. The terminal may include optional embedded Secure Hardware, such as a UICC.

digital rights management: set of methods which ensures that content can only be used when the relevant conditions (e.g. access conditions) have been met

DRM Time: secure, non user-changeable time source

NOTE: The DRM time is measured in the UTC time scale.

DVB network: collection of MPEG-2 Transport Streams, each carrying a multiplex, and transmitted on a single delivery system

NOTE: DVB network is identified by network_id.

electronic service guide: used in IP Datacast systems to signal the services that are available; how they can be received and what they contain

interactive Device: Device that is capable of directly connecting to a Rights Issuer through an interactivity channel using an appropriate protocol over an appropriate transport/network layer interface, e.g. HTTP over TCP/IP

interactive domain: OMA DRM 2.0 domain, as specified in [49].

NOTE: This is a group of Devices for which Rights Objects can be issued via an Interactivity Channel granting rights to the whole group.

interactivity channel: bi-directional channel established between the IPDC Service Network and the IPDC mobile terminal for reliable exchange of messages

interactivity channel Rights Object: content, programme or service Rights Object delivered over the interactivity channel

interactivity network: network supporting bi-directional communication and the reliable delivery of messages between the Device and the Rights Issuer

interactivity network cell: cell forming part of an interactivity network

IPDCKMSId: identifier assigned for a particular IPDC Key Management System by DVB (also known as CA_System_Id in [39] and [14])

IP flow: stream of IP datagrams each sharing the same IP source and destination address

ISMACryp: defines an end-to-end content encryption system for media carried over RTP streams and ISO based media files

key management system: end-to-end system to authorize users and provide them the necessary means to access protected content

key stream message: message broadcast alongside a protected service, carrying key material to decrypt and optionally authenticate the service

media Access Unit (AU): the smallest data entity to which timing information can be attributed

NOTE: In the case of audio an AU is an audio frame and in case of video a picture.

metering Rights Object: content, programme or service Rights Object used in conjunction with consumption-based charging

mobile TV application: main Device application responsible for accessing the Mobile TV Services

NOTE: Sometimes referred to as the Player Application.

nonce: randomly chosen value, different from previous choices, inserted in a message to protect against replay attacks

Off Line: without using a direct link (such as the interactivity channel) between network entities and the device

NOTE: Also called 'Out of Band'

OMA BCAST: working group in OMA developing a fully standardized system allowing operators to control access and usage of broadcast content on mobile Devices

Out of Band: without using a direct link (such as the interactivity channel) between network entities and the device

NOTE: Also called "Off Line".

parameter set: either a sequence parameter set or a picture parameter set

NOTE: This term is used to refer to both types of parameter sets.

parameter set Elementary stream: elementary stream containing access units made up of NAL units for coded picture data

Programme Rights Object: Rights Object containing a programme key or keys and pertaining to service protection

programme: logical portion of a service with a distinct start and end

Reserved (for future use): data elements that are reserved for future standardization

NOTE: Unless otherwise specified, such data elements SHALL be set to 0 if not in use.

reserved_for_future_use: bits that are reserved for future use and SHALL be set to 0 in systems according to the present document

rights issuer context: consists of information that was negotiated with a given Rights Issuer, during the 4-pass Registration Protocol such as the Rights Issuer ID, Rights Issuer certificate chain, version, algorithms used and other information

NOTE: This Rights Issuer Context is necessary for a Device to successfully participate in all the protocols of the ROAP suite, except the Registration Protocol.

rights issuer service: IP Datacast channel used to carry Broadcast Rights Objects, registration data and other messages from a Rights Issuer over a broadcast channel

rights issuer: entity that registers Devices and provides Rights Objects allowing Devices to receive protected services

Rights Object: collection of permissions, keys and other attributes which are linked to items of content or services

roaming: end-user accessing IPDC services through another ('foreign') Service Provider than the 'home' Service Provider

NOTE: Roaming is usually based on roaming agreements between Service Providers.

service: one or more IP flows intended to be presented together

NOTE: Examples include, but are not limited to, audio and video services.

service guide: See the Electronic Service Guide.

service operation centre: See the service provider.

service protection: protection of a service such that only authorized Devices are able to receive and decode it

service provider: provider of a broadcast service - the entity that broadcasts the service, and provides key and schedule information to Rights Issuers

service Rights Object: Rights Object containing one or multiple service keys and pertaining to service protection

service roaming: while roaming, user has access to same services as through the home Service Provider

SimulCrypt: provides the ability for multiple KMSs to control access at the same time to a single content stream

terminal: consumer device that can receive, descramble, and decode mobile DVB-H services

token Rights Object: special Interactivity Channel Rights Object containing tokens for pre-/post-paid consumption-based charging of interactive Devices, or a special Broadcast Rights Object containing tokens for pre-/post-paid consumption-based charging of Broadcast Devices

unconnected Device: Device that cannot directly connect to a Rights Issuer via an interactivity channel for the acquisition of Rights Objects, but can do so via an intermediary Device.

NOTE: As defined in [48].

user: the person using the Device to receive protected services

user roaming: while roaming, user has access to the services of the foreign Service Provider

video elementary stream: elementary stream containing samples made up of only sequence and picture parameter set NAL units synchronized with the video elementary stream

3.2 Symbols

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

E{K}(M)	Encryption of message "M" using key "K"
D{K}(M)	Decryption of message "M" using key "K"
P	Public key
Q	Private key
RIQ	Rights Issuer private key
RIP	Rights Issuer public key
DQ	Device private key
DP	Device public key
A{K}(M)	Authentication of message "M" with key K
V{K}(M)	Verification of message "M" with key K
$A \oplus B$	Bit-wise exclusive OR of A and B
A | B	Bit-wise OR of A and B
A || B	Concatenation of A and B
A + B	Arithmetic addition of A and B, or in some contexts, when arguments are strings, concatenation of A and B

A << B	Bitwise shift left operation, shift A by B bits, filling with zeros
A >> B	Bitwise shift right operation, shift A by B bits, filling with zeros
LSBm(X)	The bit string consisting of the *m* least significant bits of the bit string *X*.
MSBm(X)	The bit string consisting of the *m* most significant bits of the bit string *X*.

3.3 Abbreviations

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

ACG	Access Criteria Generator
AES	Advanced Encryption Standard

NOTE: See [17].

AID	Application IDentifier
APDU	Application Protocol Data Units
API	Application Program Interface
ARC	Action Request Code
ASCII	American Standard Code for Information Interchange
AU	Access Unit
AVC	Advanced Video Codec

NOTE: See ISO/IEC 14496-10 [55].

AVP	Attribute-Value Pairs
BAK	BCRO Authentication Key
BCD	Binary Coded Decimal
BCI	Binary Content ID
BCRO	Broadcast Rights Object
BDK	Broadcast Domain Key
bslbf	bit string left bit first
CA	Conditional Access
CA	Certification Authority
CBC	Cypher Block Chaining
CDC	Connected Device Configuration
CDP	Content Delivery Protocol
CEK	Content Encryption Key
CID	Content ID
CLDC	Connected Limited Device Configuration
COC	Customer Operation Centre
CPCM	Content Protection and Copy Management
CRL	Certificate Revocation List
CTR	CounTeR
DCF	DRM Content Format
DEK	Daily Encryption Key
DIST Mgmt	service DISTribution Management
DRD	Device Registration Data
DRM	Digital Rights Management
DVB	Digital Video Broadcasting
DVB-H	DVB-Handheld
DVB-T	DVB-Terrestrial
ECB	Electronic Code Book
ECM	Entitlement Control Message
ECMG	ECM Generator
EIS	Event Information System
EMM	Entitlement Management Message
EMMG	EMM Generator
ESG	Electronic Service Guide
ESP	Encapsulating Security Payload
GCF	Generic Connection Framework
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications

HMAC	Hashed Message Authentication Code
HO	Home Operator
HTTP	Hyper Text Transfer Protocol
HTTPS	Secure - Hyper Text Transfer Protocol
ICRO	Interactivity Channel Rights Object
ID	IDentifier
IEK	Inferred Encryption Key
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
IPDC	IP DataCast
i-point	interoperability point
IPPV	Impulse Pay Per View
IPsec	IP Security
IRD	Inform Registered Device (protocol)
ISMA	Internet Streaming Media Alliance
ISO	International Standards Organization
IV	Initialization Vector
J2ME	Java 2 Micro Edition
KDA	Key management system Device Agent
KMM	Key Management Message
KMS	Key Management System
KSL	Key Stream Layer
KSM	Key Stream Message
LBDF	Long-form Broadcast Domain Filter (a.k.a. longform_domain_id)
LSB	Least Significant Bit
MAC	Message Authentication Code
MF	Master File
MII	Major Industry Identifier
MJD	Modified Julian Date
mjdutc	Modified Julian Date UTC
MK	Master Keys
MKI	Master Key Indicator
MPEG	Moving Pictures Experts Group
MTU	Maximum Transmission Unit
NAL	Network Abstraction Layer
NDD	Notification of Detailed Data
NSD	Notification of Short Data
OBEX	Object Exchange Protocol
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance
OOB	Out of Band
OTA	Over The Air (i.e. transfer over a wireless connection)
PAK	Programme Authentication Key
PAS	Programme Authentication Seed
PDR	Push Device Registration
PEAK	Programme Encryption / Authentication Key
PEK	Programme Encryption Key
PKC	Public Key Certificate
PKC-ID	PKC IDentifier: the hash of the Public Key Certificate
PKCS#1	Public Key Cryptography Standard #1
PKI	Public Key Infrastructure
PPV	Pay Per View
PSS	Probabilistic Signature Scheme
RI	Rights Issuer
RIAK	Right Issuer Authentication Key
RID	Registered application provider IDentifier
RIS	Rights Issuer Services
RML	Rights Management Layer
RO	Rights Object
ROAP	Rights Object Acquisition Protocol
ROC	Roll-Over Counter
ROT	Root Of Trust

RSA	Rivest-Shamir-Adelman public key algorithm
RTP	Real-time Transport Protocol
SA	Security Association
SAC	Secure Authenticated Channel
SAK	Service Authentication Key
SAS	Service Authentication Seed
SBDF	Short-form Broadcast Domain Filter (a.k.a. shortform_domain_id)
SCS	SimulCrypt Synchronizer
SDP	Session Description Protocol
SEAK	Service Encryption / Authentication Key
SEK	Service Encryption Key
SGK	Subscriber Group Key
SHA-1	Secure Hash Algorithm
SHW	Secure HardWare
SIM	Subscriber Identity Module
SI/PSI	Service Information / Programme Specific Information
SMS	Short Message Service
SOC	Service Operation Centre
SPI	Security Parameters Index
SPP	Service Purchase and Protection
SRTP	Secure Real-time Transport Protocol
SUB Mgmt	Service Subscription Management
TAK	Traffic Authentication Key
TAS	Traffic Authentication Seed
TDK	Token Delivery Key
TEK	Traffic Encryption Key
UDF	Unique Device Filter
UDK	Unique Device Key
UDN	Unique Device Number
UDP	User Datagram Protocol
UGK	Unique Group Key
UICC	Universal Integrated Circuit Card
uimsbf	unsigned integer, most significant bit first
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
USIM	Universal Subscriber Identity Module
UTC	Universal Time, Co-ordinated
VM	Virtual Machine
VO	Visited Operator
WAP	Wireless Application Protocol
WIM	WAP Identify Module
ZMB	Zero Message Broadcast

4 System overview

The two systems for Service Purchase and Protection (SPP) in IP Datacast (IPDC) - Open Security Framework (specified in annex A), and 18Crypt (specified in annex B) - are based on a common key hierarchy model. Both systems are referring to content/service encryption mechanisms described in clause 6 that are to be implemented by all IPDC terminals and therefore they can all be used by either of the systems.

4.1 Hierarchical Model for Content/Service Protection (Informative)

The hierarchical model for the content/service protection mechanisms specified in the present document is outlined in figure 1.

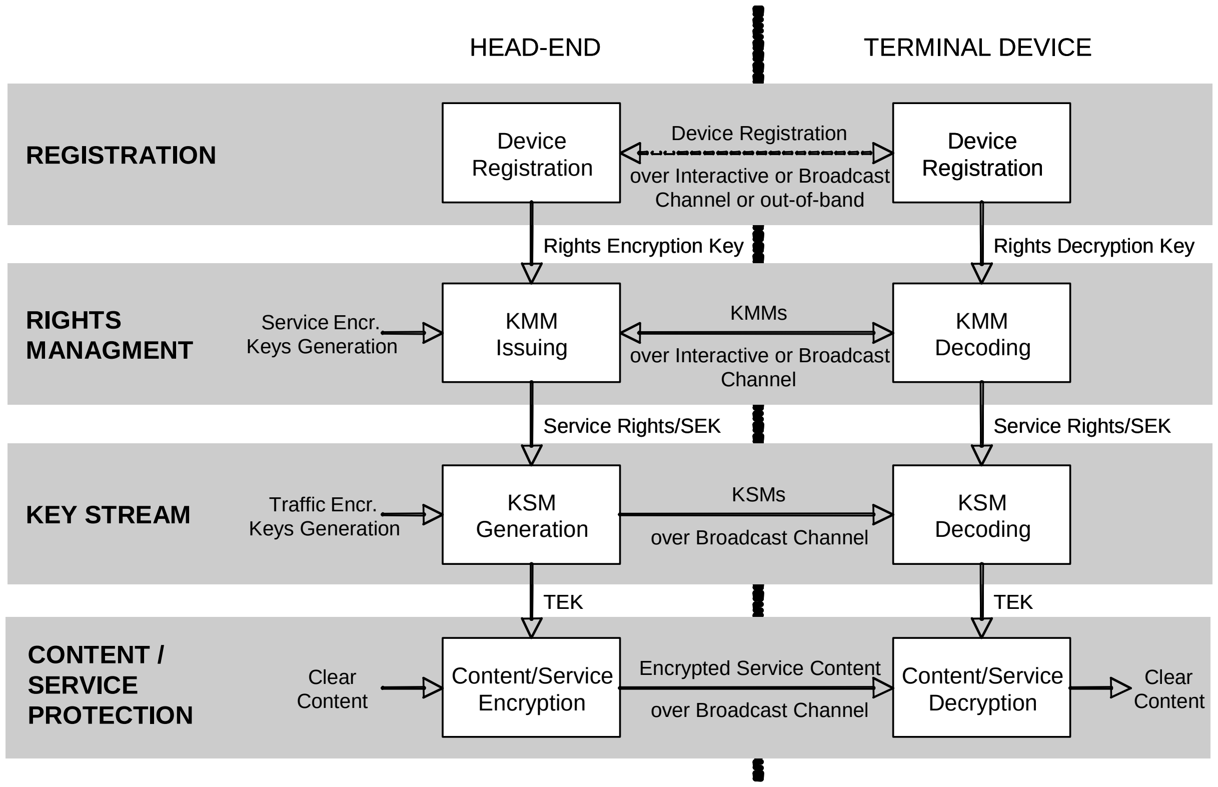


Figure 1: Hierarchical Model for Content/Service Protection

In the following description of the hierarchy levels references into the present document and its annexes are given that help to locate the specification text for each of the layers.

Registration

Key material and metadata are exchanged during the registration phase that will enable Devices to decrypt and authenticate rights and subsequently access content/services.

The IPDC SPP Open Security Framework specification (annex A) defines the registration layer as private to the Key Management System (KMS) and therefore out of scope of the present document with the exception of roaming.

The Open Security Framework specification defines a standard registration mechanism for roaming services in clause A.5.

For 18Crypt, for broadcast and out of band as well as interactivity channels, the registration layer is specified in clause B.3.4. Special considerations for roaming are described in clause C.2.

Rights Management

Content/service access rights are delivered to the Device using the Key Management Messages (KMMs). KMMs are typically exchanged as a result of a purchase transaction and transferred to terminals via the interactive or broadcast channel. The content/service access rights can consist of a Service Encryption Key (SEK), used to access the Key Stream Messages (KSMs), and/or information such as entitlements.

In the IPDC SPP Open Security Framework specification KMMs are referred to as Entitlement Management Messages (EMMs). The Open Security Framework defines the EMM stream signalling in accordance with clause 5 and defines the format and content of the EMM as private to the KMS. Clause A.3.1 provides a description of the EMM security mechanisms.

The Open Security Framework specification defines a standard Rights Management Layer (RML) for roaming services in clause A.5.

In the 18Crypt specification KMMs are delivered via the broadcast channel or the interactivity channel and can carry Rights Objects (ROs) containing rights on the service or programme levels. 18Crypt specifies the RML and Rights Issuer Services (RIS) in clause B.3.3 and B.4, respectively. Special considerations for roaming are described in clause C.2.

Key Stream

The Key Stream Layer (KSL) implements the delivery of Traffic Encryption Keys (TEKs) by transmission of Key Stream Messages (KSMs) to the terminal on the broadcast channel. These messages, in essence, contain information that allows the terminal to reconstruct the TEKs needed to decrypt the content/service. KSMs may contain additional information to control access to the content service, such as access criteria.

In the IPDC SPP Open Security Framework specification the KSMs are referred to as Entitlement Control Messages (ECMs). The Open Security Framework defines the ECM stream signalling in accordance with clause 5 and defines the format and content of the ECM as private to the KMS. Clause A.3.2 provides a description of the ECM security mechanisms.

The Open Security Framework specification defines a standard KSL for roaming services in clause A.5.5.6.

In the 18Crypt specification Key Stream Messages are referred to as KSMs. 18Crypt specifies the KSL in clause B.3.2. Special considerations for roaming are described in clause C.2.

Content/Service Protection

The content/service is encrypted by a symmetric encryption algorithm using a Traffic Encryption Key (TEK). The encryption can be performed at the link layer (IPsec), session layer (SRTP), or content layer (ISMACryp). TEKs change frequently to prevent real-time key distribution attacks.

The three content/service protection mechanisms are specified in clause 6.

4.2 The two approaches (informative)

The Service Purchase and Protection system referred to as the IPDC SPP Open Security Framework (annex A) specifies a security framework that allows any KMS to be used (or even multiple KMSs in parallel using Simulcrypt [1])within that framework. The specification of KMSs to be used within that framework is out of scope of the present document,

The Open Security Framework leaves both the KSM and KMM systems as private, allowing the possibility to deploy a KMS that defines both the Key Stream and Rights Management layers.

The SPP system referred to as 18Crypt (annex B) is a fully specified system with respect to all layers of the content/service protection model described above. It references OMA [48] as the common solution for the registration and rights management over the interactive channel and specifies a set of protocols for use in broadcast and out of band channels, while not precluding the use of other, optional rights management systems for the same purpose.

At the Key Stream layer, the 18Crypt system specifies a KSM stream that is used by all rights management systems.

At the content/service protection layer SPP systems use encryption mechanisms specified in clause 6. Signalling of the SPP system actually being used by services is described in clause 5.

Architectural overviews specific to each SPP system are provided in corresponding annexes. For the Open Security Framework this is found in clause A.2, for 18Crypt this is found in clauses B.1 and B.2.

4.3 Use of the IP Datacast Specification for Service Purchase and Protection (normative)

The SPP specification for IPDC over DVB-H consists of three components:

- A common signalling mechanism describing the SPP systems used in the received signal (clause 5).
- A set of content/service encryption mechanisms (clause 6).

- Two alternative specifications: IPDC SPP Open Security Framework (annex A) and 18Crypt (annex B).

The common signalling mechanism SHALL be based on clause 5.

If the terminal supports SPP it SHALL implement the protection mechanisms described in clause 6.

If the terminal supports SPP it SHALL implement one or both of the systems described in annexes A and B; the normative part of each annex is applicable within the context of that annex.

5 Signalling of Service Purchase and Protection System (normative)

Signalling of the SPP system provides a means to signal to a terminal whether a service is protected and which KMSs and key streams are used to protect the service.

Signalling is provided at two layers: the Electronic Service Guide (ESG) and the Session Description Protocol (SDP). The signalling is defined in TS 102 471 [35] and TS 102 472 [36] respectively. The two clauses below identify the parts of these specifications that are relevant to SPP signalling.

5.1 ESG signalling

The signalling provided within the ESG specification TS 102 471 [35] enables the following functionality:

- The inclusion of SPP-specific data for the purchasing of rights to consume content/services.
- The ability to signal the SPP system or systems that will enable consumption of a service or content item.
- The ability to signal a service on which KMS-specific data is delivered.
- The ability to describe details about how and from where rights to the content/services can be obtained.

TS 102 471 [35], clause 5.4 describes the Service fragment, which may be used to signal to the terminal the presence of a service carrying KMS-specific data. A service carrying KMS-specific data is declared by setting the ServiceType element of the fragment appropriately. Any KMS-specific fields may be included using the PrivateData element of the Service fragment.

TS 102 471 [35], clause 5.8 defines a PurchaseRequestType within the Purchase fragment that enables the inclusion of SPP-specific data required to make a purchase. This type has 3 elements:

- DRMSystem: This uniquely identifies the KMS from which the purchase can be made.
- PurchaseData: This uses an abstract data type, allowing the inclusion of KMS-specific fields to support the purchasing of the described offer.
- PurchaseChannelIDRef: This is an optional element that enables the referencing of a PurchaseChannel fragment. The PurchaseChannel fragment describes an entity from which the purchase may be made.

A Purchase fragment allows multiple PurchaseRequests to be defined, therefore enabling the purchase of content from multiple KMSs.

TS 102 471 [35], clause 5.9 describes the PurchaseChannel fragment, and provides the ability to describe a point of purchase. Data specific to a KMS may be included using the PrivateData element.

TS 102 471 [35], clause 5.10 describes the Acquisition fragment. This fragment defines a KeyStream element that enables the signalling of the KMS and the operator used for the delivery of the content associated with the Acquisition fragment.

5.2 SDP signalling

TS 102 472 [36] describes how SDP is included in an SDP file:

Clause 10.1 specifies the fields used within an SDP file to signal the presence of a key stream.

Clause 10.2 specifies the fields used within an SDP file to describe a KMM stream.

Clause 10.3 specifies how to bind a key stream to one or more components forming a service.

Additionally, the following attributes SHALL be included in SDP when SRTP is used:

```
a=SRTPAuthentication:n
```

where n is the SRTP authentication algorithm value for the mode of the RCC transform used, as specified in [38].

```
a=SRTPROCTxRate:R
```

where R is the value of the ROC transmission rate parameter, an integer between 1 and 65535 inclusive, as specified in [38].

6 Protection of content and media streams (normative)

6.1 IPsec

The broadcast network MAY use IPsec [24] to protect broadcast services. All Devices SHALL support IPsec.

The IPsec implementation in the Device SHALL be such that it does not interfere with the usage of IPsec for applications other than IPDC. This implies that the Security Parameter Index (SPI) allocation and Security Association (SA) look-ups SHALL be implemented in such a way that they interoperate with existing IPsec implementations.

An IPsec SA consists of a tuple of the following parameters:

- Selectors (IP protocol version, source IP address, destination IP address, protocol, source port and destination port).
- SPI.
- Destination IP address.
- Security protocol, security protocol mode and security protocol parameters.
- Algorithms and algorithm parameters.
- Key material.

An IPsec SA SHALL be uniquely identified by a destination IP address and SPI pair.

6.1.1 Selectors

Selectors are provided by the KSL. The selectors MAY contain wildcards, ranges or point values, but all the other parameters SHALL be exactly defined. All address selectors SHALL be point values and the destination address selector SHALL match the destination IP address of the SA.

6.1.2 Encapsulation protocol and mode

If IPsec is used for encryption of IPDC, the protocol and mode SHALL be ESP in Transport Mode, according to [24] and [26]. Other IPsec encapsulation protocols or modes SHALL NOT be used.

6.1.3 Encryption algorithm

The encryption algorithm for IPsec ESP SHALL be AES-128-CBC with explicit IV in each IP packet, as defined in [27] and [29]. Other encryption algorithms or key sizes or chaining modes SHALL NOT be used.

6.1.4 Authentication algorithm

The authentication algorithm for IPsec ESP SHALL be HMAC-SHA-1-96, as defined in [23] and [25]. Other authentication algorithms or truncations SHALL NOT be used.

An IPDC system MAY use authentication. If no authentication is desired, the NULL authentication algorithm SHALL be specified. In this case, replay protection SHALL NOT be performed by the Device.

6.1.5 Security Association Management

The KSL defines how often the transport layer keys are re-keyed. This sets the following requirements:

- The TEK provided by KSL SHALL be used as the key for the ESP encryption.
- The Traffic Authentication Key (TAK) provided by the KSL SHALL be used as the key for the ESP Message Authentication Code (MAC) if authentication is used.
- The IPsec implementation SHALL be able to manage SAs relating to the KSL separately from those managed manually or by any other protocol such as IKE. This implies the ability to identify whether an SA relates to the KSL.
- SAs relating to KSL SHALL be prioritized lower than those SAs that have a locally defined policy or a policy that is provided by a trustworthy party.
- SAs relating to KSL are simplex and SHALL be applied only to inbound traffic on the recipient side.

The re-keying of existing SAs by the KSL SHOULD be managed on a resource basis by the IPsec layer according to the following recommendations:

- The IPsec implementation SHOULD be able to keep alive at least the two most recently instantiated IPsec SAs for a particular set of selectors.
- The IPsec implementation SHOULD provide a least-recently-instantiated mechanism for cleaning up SAs as resources reserved for IPDC IPsec SAs are exhausted.
- The number of IPDC SAs required to exhaust the resources such that the cleanup mechanism is triggered SHOULD be 3 per service key per set of IP selectors.
- A Device SHOULD be able to rekey any SA at least for every 20 received ESP packets without a significant loss in performance. This rekey consists of installing a new SA with a defined set of selectors, and possibly, eliminating an old SA with an equal set of selectors. Both SAs in this case are managed by the KSL.

NOTE: A broadcaster is not recommended to re-key existing SAs for every 20 packets, as the amount of traffic one can place in 20 packets varies heavily with the maximum packet size. The impact on the Device in terms of time is also hard to estimate, as the timing between packets may be significantly altered in a broadcasting environment. Therefore a broadcaster SHALL NOT re-key an IPsec SA more often than every two seconds at the point of sending the messages through KSL.

6.2 ISMA Encryption and authentication (ISMACryp)

6.2.1 Streamed Content

Streamed content MAY be encrypted and carried over RTP as specified in ISMA Encryption and Authentication [11]. All Devices SHALL support ISMA Encryption and Authentication [11] ("ISMACryp").

The ISMACrypSalt parameter SHALL be used to signal the ISMACryp salt in the attributes of each encrypted media streams.

The default ISMACryp cipher, mode and configuration (AES-128-CTR, as defined in [11]) SHALL be used.

An ISMACryp broadcast implementation SHALL change the key over time, but no faster than once per second for each encrypted stream.

The length of the Key Indicator SHALL be greater than or equal to 1 byte.

6.2.2 Downloadable Audio/Visual content (stored in MP4 files)

Encryption of downloadable audio/visual content, when stored in an ISO Base Media File Format (MP4) [12] file, MAY be performed as specified in ISMA Encryption and Authentication [11].

The default ISMACryp cipher, mode and configuration (AES-128-CTR, as defined in [11]) SHALL be used.

The ISMACrypSaltBox box SHALL be used to signal the ISMACryp salt in the SchemeInformation box of each encrypted media stream.

An ISMACryp broadcast implementation SHALL change the key over time, but no faster than once per second for each encrypted stream.

To support a Simulcrypt [1] approach, the SchemeInformationBox of each encrypted media stream MAY contain multiple occurrences of the ISMAKMSBox, one for each KMS.

6.3 SRTP

The broadcast network MAY use SRTP [32] to protect broadcast services. Devices SHALL support SRTP.

An SRTP session is defined as the cryptographic context for an RTP session. The cryptographic context for SRTP, when used for service protection, consists of the following elements:

- Roll-Over Counter (ROC);
- receiving sequence number;
- cipher and mode definition;
- Message Authentication Code (MAC) method definition;
- list of received packets;
- array of Master Keys (MKs);
- Master Key Indicator (MKI) indicator bit;
- length of the MKI field;
- value of currently active MKI;
- array of counters of processed packets for each Master Key;
- length of encryption and authentication keys;
- Master Salt;
- context id.

A cryptographic context is uniquely identified by its context id. The context id consists of the Synchronization Sources, destination network address and destination transport port number.

6.3.1 Key management

The IPDC SRTP application SHALL use the MKI value for looking up decryption keys. This means that a cryptographic context SHALL have the MKI indicator bit set to 1. The <From, To> value method of key lookup SHALL NOT be used.

The Master Salt SHALL NOT be used.

The TEKs provided by the KSL SHALL be used as the SRTP MK.

The key derivation rate SHALL be 0. Exactly one SRTP session encryption key SHALL be derived from one MK. If SRTP authentication is enabled, exactly one SRTP session authentication key SHALL be derived from one MK.

The KSL SHALL provide and update the cryptographic contexts (excluding the ROC) to the SRTP implementation. Note that some fields are initialized and/or managed internally within the SRTP implementation, such as the list of received packets used in replay protection, receiving sequence number, and the ROC.

The ROC value is included in the plaintext over which a MAC is computed (assuming authentication is used) and is included in the (implicit) IV for the AES-CM encryption, and therefore the ROC is needed to encrypt/decrypt a packet.

The Sender's ROC SHALL be transferred in every R-th packet according to [38], where R is a configurable parameter which is signalled out of band and must be greater than 0. See clause 5.2, SDP Signalling.

Because the SRTP key-derivation rate is not used and the <From,To> values are also not used, the SRTP crypto context will be rekeyed by the KSL. The SRTP implementation SHALL be able to handle installing a new crypto context every 20 packets. An SRTP broadcasting implementation SHALL NOT require an install or an update of a new crypto context more than once a second for a single SRTP context id, at the point of sending the messages through the KSL.

6.3.2 Encryption algorithm

The encryption algorithm for SRTP packets SHALL be AES-128-CTR, as defined in [32]. Other encryption algorithms or key sizes or chaining modes SHALL NOT be used.

6.3.3 Authentication algorithm

The broadcast system SHALL use one of the Rollover Counter Carrying (RCC) transforms defined in [38]. The mode of RCC transform used is a configurable parameter which is signalled out-of-band (see clause 5.2, SDP Signalling.)

The broadcast system MAY choose to provide authentication for all packets (RCCm1), or it MAY choose to provide authentication only for those packets conveying the Sender's ROC (RCCm2), or it MAY choose not to provide authentication for any of the packets (RCCm3).

Where used, the authentication algorithm SHALL be as defined in [38], based on HMAC-SHA-1-80 as defined in [23] and [32]. Other authentication algorithms or truncations SHALL NOT be used. For those packets for which no authentication is provided, also replay protection SHALL NOT be performed by the Device.

Annex A (normative): IPDC SPP Open Security Framework

A.1 Introduction

This annex specifies the IPDC SPP Open Security Framework. This Open Security Framework is designed to provide a secure and flexible solution for operators, horizontal-market channel providers and device manufacturers, and end-users. Its features include:

- **Adaptability:** The ability to download updates of key security features and new business models to Devices in the field. Thus a flaw in the security system can be fixed by the security provider without waiting for a standard to be agreed upon. Moreover, new business models can be developed by operators and content providers and rapidly provided to end-users.

- **Vendor independence:** Operators have the freedom to seamlessly switch between technical security solutions from different providers without the need to replace the mobile Device. Moreover, the Open Security Framework is ideally suited for implementation in horizontal-market (standard) Devices.

- **Proven approach:** The Open Security Framework is modelled after the proven PayTV content security paradigms that protect high quality content world-wide.

- **Interoperability and roaming:** The Open Security Framework provides a global roaming solution between different independent KMSs. Furthermore, the Open Security Framework infrastructure enables the deployment of DVB SimulCrypt [1], which provides highly secure roaming amongst a set of operators, each with its own independent security system. Using SimulCrypt, the effect of any security system compromise can be negated and has no impact on the others.

- **Control of KMS:** The KMS is the security component responsible for the generation of TEKs and business model enforcement. For better security and control the KMS can be implemented inside a Universal Integrated Circuit Card (UICC) under the control of the operator. This annex specifies the standard protocols used to enable communication between device and UICC.

Security features include:

- **Renewable security:** The main element of this security system is its KMS; the ability of a security system to renew its KMS is crucial for long-term maintenance of security.

- **Support for multiple security systems:** The Open Security Framework is designed to support any number of security providers" technologies by allowing seamless replacement (by software download) of one security provider's KMS with another.

- **KMS compartmentalization:** Having a choice from many independent KMSs enhances security as a compromised KMS can be switched off in favour of a new one. The Open Security Framework enables seamless transition to another KMS vendor. A KMS supplier can use variants of their security solution in different markets to minimize the likelihood and impact of any security compromise.

- **Security roadmap:** The choice of ISMACryp as the content-level cipher preferred by the Open Security Framework allows the easy integration of the descrambler and decoder thus protecting the clear-compressed content. Furthermore, ISMACryp is a secure and efficient cipher for streaming content, (further details listed below).

- **Secure delivery of TEKs:** A Secure Authenticated Channel (SAC) is specified for secure delivery of TEKs from the KMS (preferably in the protected UICC) to the descrambler in the handset.

The proven approach for DVB PayTV systems has been adopted and extended by the IPDC SPP Open Security Framework. Three standardized methods are defined for encrypting content, in clause 6 of the main body of the present document. The standardized method for signalling the binding of content with a KMS and with its key management streams is defined in clause 5 of the main body of the present document. Further definition and explanation of the use of the standard encryption and signalling methods within the Open Security Framework is provided in clauses A.3 and A.4. A SAC protocol between the KMS application and descrambler is defined in clause A.7. The adaptation of the DVB SimulCrypt head-end interfaces to the IPDC over DVB-H context is defined in clause A.8. In addition, to support the roaming of users between operators, roaming service key and management streams are specified in clause A.5. Data exchanges between operators to support roaming are also defined in clause A.5.

For the horizontal market, in which terminal devices may be sold without any prior customization to a particular operator, a further extension beyond the DVB PayTV architecture has been defined in the form of the Mobile Device Security Framework, which is described in clause A.10. The Mobile Device Security Framework facilitates secure mobile device implementations and interoperability with different KMS solutions. It defines a KMS Device Agent (KDA) Platform on which the KDA can run as a Java application. The KDA platform is specified as an extension to the widely deployed MIDP 2.0 profile of J2ME. APIs are defined to allow any KDA to access the security resources of the device.

The layered approach provided by the Open Framework is illustrated in figure A.1. The Open Security Framework sits above the common layer of content encryption, and allows any KMS system to be plugged in to the Framework. For example, figure A.1 illustrates that a KMS solution that plugs into the Open Security Framework could be from various suppliers 'A', 'B', 'C' or could be based on OMA DRM ("D").

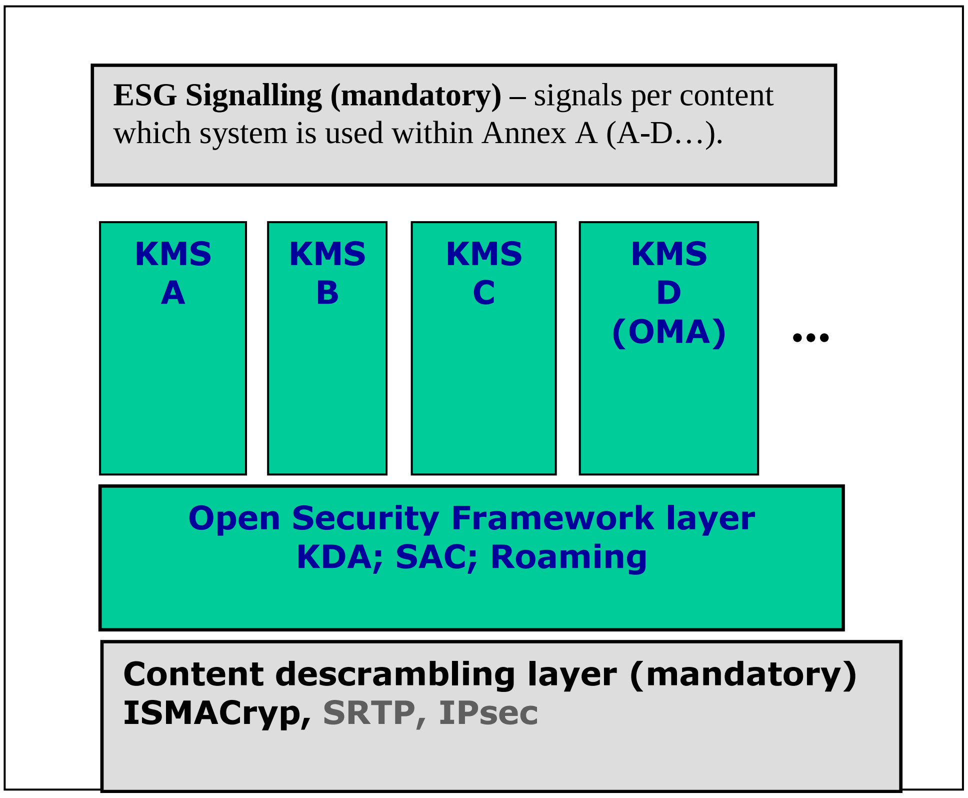

ESG Signalling (mandatory) – signals per content which system is used within Annex A (A-D…).
KMS A
KMS B
KMS C
KMS D (OMA)
…
Open Security Framework layer KDA; SAC; Roaming
Content descrambling layer (mandatory) ISMACryp, SRTP, IPsec

Figure A.1: Open Security Framework Layer Model

There are a number of features of the Open Security Framework approach in retaining flexibility in the KMS definition and its update. Some major features are listed below:

- The operator can choose the KMS solution to match their business models and security needs. The business capabilities of a KMS solution can be further tailored to an operator's needs allowing differentiation between operators" business models.
- The operator can switch to a different KMS solution from the same or different supplier.

- The existence of multiple deployed KMS solutions reduces their attractiveness to attack in comparison with a standard KMS. A KMS supplier can further use variants of their security solution in different markets to minimize the likelihood and impact of any security compromise.
- The design of a KMS can be kept confidential, making it harder to circumvent.
- A KMS supplier can readily deploy upgrades to their system to support new business models and to deal with any security compromises. The commercial and technical responsibilities for maintaining system and content security are well defined and lie between the KMS supplier and the operator.
- The KMS can be based on solutions proven in deployment in satellite, terrestrial, cable and IP Broadband systems.

There are 3 methods for encryption defined in the present document, which operate at different layers of the protocol stack. For services delivering content secured by the Open Security Framework, the selection of the ISMACryp method is strongly recommended for technical reasons outlined below:

- ISMACryp is well suited for protecting the clear compressed content as it is easy to integrate the ISMACryp descrambler with the decoder.
- ISMACryp provides end-to-end protection of the content and facilitates decryption in close proximity to the decoder.
- ISMACryp supports pre-encryption of the content.
- ISMACryp is transport and content independent, allowing encrypted content to be carried over multiple transports (file, RTP), without intermediate decryption. This is useful for pre-encryption and recording of encrypted content.
- ISMACryp is well suited for broadcast transport with explicit support for crypto-periods and a reliable key synchronization mechanism.

The other encryption methods of IPsec and SRTP are also supported by the Open Security Framework, and may be used if services are constrained to operate with these methods.

A.2 DVB IPDC over DVB-H System Architecture

A.2.1 Overview

Figure A.2 shows the overall high-level view of the end-to-end system.

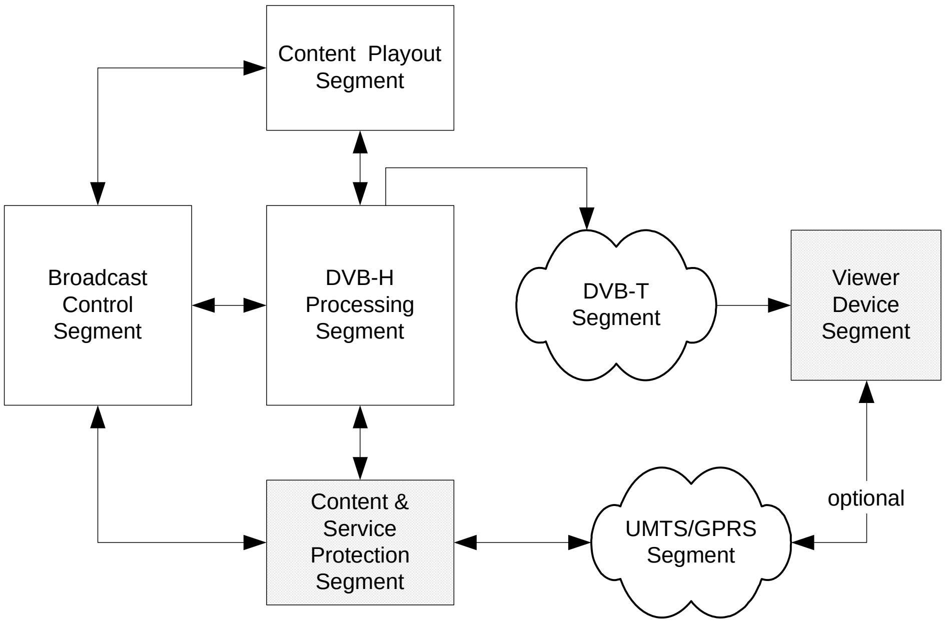


Figure A.2: Overview - DVB IPDC over DVB-H Architecture

In this high level view the following segments are presented:

Table A.1

Segment	Function
Broadcast Control Segment	Controls what content is being played at a given time, and using what access criteria.
Content Playout Segment	Responsible for the encoding, storage, and playout of content.
DVB-H Processing Segment	Responsible for: 1. Creation of the DVB-H stream, including combining any data presented by the Content and Service Protection Segment. 2. Presenting this data to the DVB-T Segment for transmission.
Content and Service Protection Segment	Responsible for: 1. Scrambling content. 2. Maintaining containers used to carry entitlements and access rights. 3. (Optional) Communication with the viewer Device via an interactivity channel, if such a channel exists.
Viewer Device Segment	This segment consists of the viewer Device and optional embedded UICC. Receives the DVB-T signal, parses the DVB-H stream, and is responsible for descrambling and rendering.

This annex concentrates on the Viewer Device Segment and the Content and Service Protection Segment.

A.2.2 Content and Service Protection Architecture

An overview of the Content and Service Protection architecture is depicted in figure A.3.

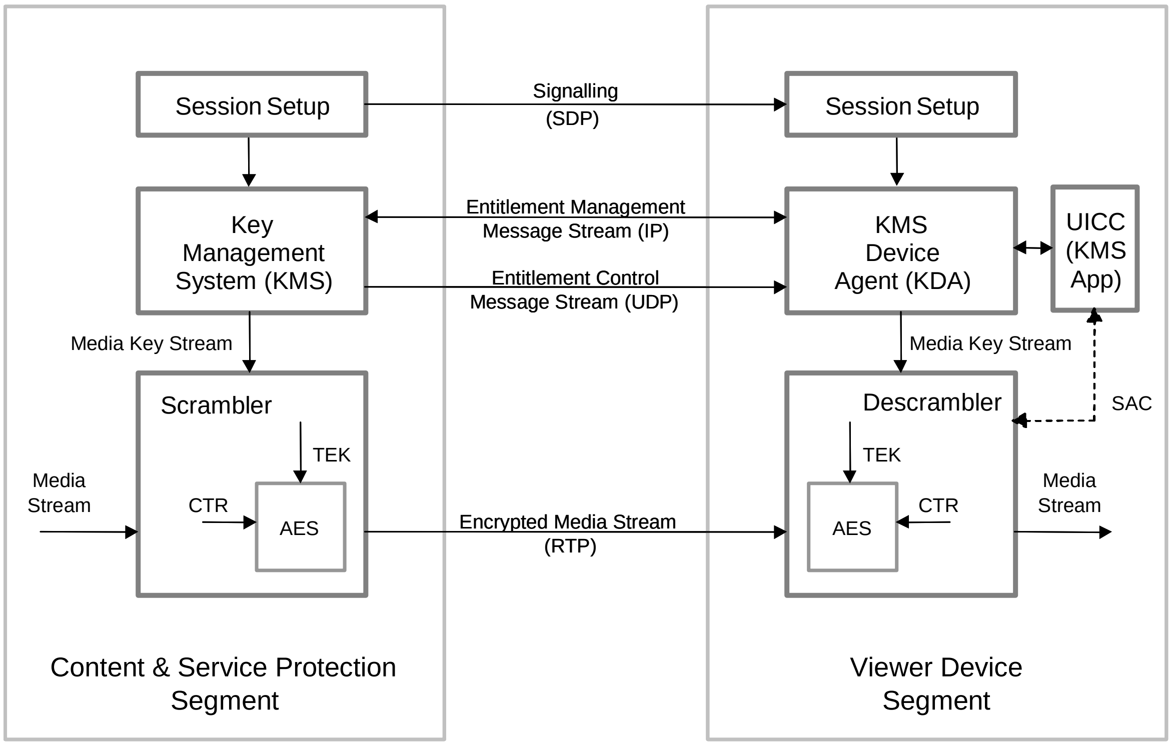


Figure A.3: Overview - Content and Service Protection Architecture

A.2.2.1 Key Management System (KMS)

The Key Management System supports the following security functions:

A.2.2.1.1 Registration

Device registration shall take advantage of the interactivity channel, when available. The use of the interactivity channel for Device registration shall be defined as private to the KMS.

A.2.2.1.2 Authorization and Rights Issuing

Device authorizations and content/service access rights shall be delivered to the Device using the Entitlement Management Message (EMM). The present document defines the signalling and transport mechanisms for the EMM stream. The contents and format of the EMMs are defined as private to the KMS.

The KMS may utilize either the broadcast channel or interactivity channel for delivering EMMs to the Device.

A.2.2.1.3 Content/Service Protection

Traffic Encryption Keys (TEKs) are applied directly to the scrambler to protect the media stream. The TEKs are sent to the Device in Entitlement Control Messages (ECMs) which are delivered over the broadcast channel alongside the encrypted media stream. The messages are cycled to support random access to broadcast services. The contents and format of the ECMs are KMS specific.

The security mechanisms supported by the present document include regular key refreshing and just-in-time key distribution. Together, these mechanisms protect real-time broadcast content against any key distribution and security attacks.

Figure A.4 shows the timing relationship between the reception of the EMM and ECM messages by the Device and the application of the TEKs to the descrambler.

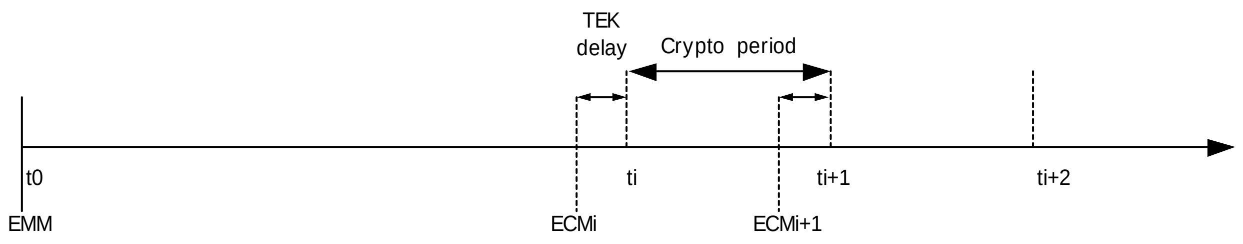


Figure A.4: EMM/ECM timing

The scrambling mechanism supports the concept of crypto-periods, allowing a time-varying sequence of TEKs to be applied to the scrambler. The encrypted media stream and the ECM stream contain the necessary information to allow synchronization of the encryption and decryption processes. The authorizations and content/service access rights are delivered to the Device ahead of consumption time using the EMM stream.

A.2.2.1.4 Copy Protection

In case content needs to be securely exported to a separate post-delivery copy-protection system such as DVB-CPCM [53], the Usage State Information needed by such a system can be carried by the KMS.

A.2.2.2 Scrambler

The media stream is encrypted with the TEKs using one of the three methods defined in clause 6.

A.2.2.3 Key Management System Device Agent (KDA)

The KMS Device Agent (KDA) contains vendor-specific logic required to control the descrambling process for a specific KMS.

The KDA performs the following security functions:

1) Reception of EMMs from the KMS by means of the broadcast or interactive networks.

2) Secure generation of Device authorizations from the received EMMs.

3) Reception of the ECM stream for the selected service from the broadcast network.

4) Secure generation of the TEKs from the received ECM stream.

5) Application of TEKs to the descrambler, controlling the decryption of the encrypted media stream.

The KDA typically communicates with secure hardware (e.g. a UICC) to perform secure generation of the TEKs from the received ECMs. The KDA also establishes a Secure Authenticated Channel (SAC) between the secure hardware and the descrambler to control the secure exchange of TEKs.

A.2.2.4 Descrambler

The media stream is decrypted with the TEKs using the descrambling method corresponding to the scrambling method selected by the broadcast system. The details of the decryption scheme can be found in clause 6.

A.2.2.5 UICC

Security-conscious operators will use viewer Devices with some secure hardware installed, specifically a Universal Integrated Circuit Card (UICC). Any secure KMS Device functions will be contained as an application on this multi-application UICC.

The interface between the Device and the UICC is specified by ISO/IEC 7816-4 [7]. The content of the messages exchanged on this communication link is outside the scope of the present document.

The KMS application residing on the UICC performs the following security functions under control of the KDA:

1) Secure generation of Device authorizations from the received EMMs.

2) Secure generation of the TEKs from the received ECM stream.

3) If signalled by the KMS, the KMS application and the descrambler shall establish a SAC.

A.2.2.6 Session Setup

The session information used by the KDA for controlling access to the media stream is carried in its own session description using SDP. This is specified in TS 102 472 [36] and further described in clause A.3.2.

A.3 IPDC security mechanisms

A.3.1 Entitlement Management Message (EMM) stream

EMMs are generated by the KMS to dynamically associate profiles to the end users in a secure manner. EMMs convey the necessary authorizations/rights for the reception of broadcast media content for a single end user or group of end users. EMMs encompass the different business rules that allow broadcasters to implement multiple business models, such as subscription, pay-per-view, pay-per-time, etc. In addition, EMMs may provide the secure mechanisms for initiating control actions on a Device, for example to control software downloads.

EMMs can be broadcast using the DVB-H network or retrieved from a URL using the interactivity channel (other delivery methods for EMMs are not precluded by the present document, e.g. SMS). In the case of broadcast, the messages may be sent on an operator configured channel, so as to optimize delivery of the message stream to Devices. During the process of receiving the EMMs, the Device will typically be authenticated, authorized, and accounted to the requested content or service.

A.3.1.1 EMM stream transport and signalling

The EMMs are transported as UDP/IP packets. Each IP packet contains any number of complete EMMs. No header or other form of signalling is carried in the payload of these UDP packets, only the EMMs themselves.

An EMM stream is described using SDP as an IPDC Key Management Message (KMM) stream as specified in TS 102 472 [36], clause 10.2. The IPDCStreamId and IPDCDRMId parameters SHALL NOT be present in the SDP description of any EMM stream.

The IPDCDRMId and IPDCStreamId parameters are not used by Open Security Framework Devices and shall be ignored.

An EMM stream is signalled by declaring a Service with the ServiceType element set to "EMM Service" (urn:dvb:ipdc:esg:cs:ServiceTypeCS:1.3.1.2) within the ESG. In addition the following data type may be included within the PrivateData element of the Service Fragment to signal the KMS with which the EMMs are associated. This data type shall be declared within the following Open Security Framework namespace: urn:dvb:ipdc:spp:OSF.

A.3.1.1.1 KMSType Syntax

```
<complexType name="KMSType">
    <complexContent>
        <extension base="esg:PrivateDataType">
            <sequence>
                <element name="CA_system_id" type="unsignedShort" />
                <element name="OperatorId" type="unsignedShort" />
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

A.3.1.1.2 KMSType Semantics

Table A.2

Field	Semantics
CA_system_id	This identifies the KMS. Allocation of the values for this element is found in [39] and [14].
OperatorId	This field identifies an operator. Allocation of the values for this element is under the control of the KMS, identified by the CA_system_id and allows differentiation between operators using the same KMS.

The ESG is located by tuning to the ESG Bootstrap IP Flow which is delivered on a well known IP address. The ESG Bootstrap announces the set of available ESGs, the terminal tunes to one of these ESG flows and acquires the ESG fragments.

The Service fragment describing the EMM Service shall have a reference to an Acquisition fragment, which contains either:

- a reference to an IP Flow carrying the EMM service's SDP file; or
- an inline SDP description for the EMM Service.

It is the SDP description that signals the IP_Flow on which the EMMs are delivered.

A.3.2 Entitlement Control Message (ECM) stream

ECMs are generated by the KMS to securely transmit the TEKs with which the media stream is encrypted. They are transmitted as an in-band stream, alongside the associated media stream. ECMs may also include any access rights definitions associated with the protected media content. The ECM stream is processed by the KDA to retrieve the TEKs required by the descrambler to decrypt the encrypted media stream:

$$TEK[i] = Decrypt(ECM[i], EMM)$$

where, in general, the EMM could be a combination of the current user profile and other service purchase characteristics associated with the end user. Characteristics include the type of subscription, or the previous purchases of the end user.

Key synchronization information (ISMACryp Key Indicator, SRTP MKI, IPsec SPI) is placed into the media stream as references to the correct ECM within the ECM stream. The media stream and synchronization information format are described in clause 6 and associated references.

ECMs should be sent in regular cycles in a low frequency transmission. Both cycles and frequency are defined by the headend and can be synchronized with the DVB-H time-slicing to minimize the number of ECMs per time-slice. This allows the Device to compensate for missing ECM packets, while providing fast zapping between services. It is envisaged that the ECM retransmission should not increase the total media stream bandwidth by more than 1 %.

The ECM stream signalling supports Simulcrypt [1], allowing multiple KMSs to control a single protected media stream within the same IPDC network. Each KMS is identified by a unique identifier: CA_system_id, as described in [39] and [14]. These identifiers are pre-defined providing a mapping between the ECM stream and the appropriate KMS.

A.3.2.1 ECM Stream transport and signalling

To support efficient ECM carriage, especially in the presence of Simulcrypt, each ECM stream is carried in its own UDP stream. A single UDP packet shall carry one or more complete ECMs. If several ECMs are carried in a single UDP packet, they are placed one after the other in the packet without additional signalling.

ECM Streams shall be signalled within an SDP file as defined in TS 102 472 [36], clause 10.1.

A.3.2.2 ECM stream binding

ECM Stream binding shall be signalled within an SDP file as defined in TS 102 472 [36], clause 10.3.

In order to facilitate implementations and the management of the descrambling contexts in the terminal, each encrypted media stream SHALL be assigned an identifier, unique within the scope of the SDP, and signalled using the "mid" attribute as specified in RFC 3388 [34]. The value of this parameter SHALL be a positive integer no greater than 256.

The following example shows a Service containing a video component and 2 audio components, where the second audio component has a separate ECM stream to that for the video and first audio component.

The SDP example is just a "skeleton" and does not include all required attributes.

```
v=0
o=CBMS 2890844526 2890842807 IN IP4 126.16.64.4
s=Scrambled TV Stream
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
a=IPDCKSMStream:10
m=data 12340 UDP ipdc-ksm
a=fmtp:ipdc-ksm IPDCStreamId=10; IPDCKMSId=1002; IPDCOperatorId=344
m=data 12341 UDP ipdc-ksm
a=fmtp:ipdc-ksm IPDCStreamId=13; IPDCKMSId=1002; IPDCOperatorId=344
m=video 51372 RTP/AVP 31
a=mid:1
m=audio 49170 RTP/AVP 0
a=mid:2
a=lang:en
m=audio 52002 RTP/AVP 0
a=mid:3
a=lang:ES
a=IPDCKSMStream:13
```

A.3.3 Key Management and IPsec

IPsec ESP [24] MAY be used to encrypt a media stream in conjunction with any KMS, as specified in clause 6.1. IPsec ESP makes use of a 128 bit AES key to perform encryption of the content. This key SHOULD change rapidly over time (typically every few seconds) and is identified by the Security Parameter Index (SPI) carried in the ESP header.

The KMS is responsible for securely carrying and delivering the encryption key (and the optional authentication key) and its identifier to authorized terminals using adequate means (typically, the ECM) in a timely fashion (i.e. before it is needed). Such means are outside the scope of the present document.

An IPsec protected media stream SHALL be signalled in the SDP description of that service by adding the following attribute to the list of attributes of the media stream:

```
a=IPsecESP
```

EXAMPLE:

```
m=video 0 RTP/AVP 96
a=rtpmap:96 H264/90000
a=IPDCKSMStream:10
a=IPsecESP
a=mid:3
```

The SDP example is just a "skeleton" and does not include all required attributes.

A.3.4 Key management and ISMACryp

ISMACryp [11] MAY be used to encrypt content in conjunction with any KMS, as specified in clause 6.2.

The use of ISMACryp is STRONGLY RECOMMENDED.

ISMACryp makes use of a 128 bit AES key to perform encryption of the content. This key SHOULD change rapidly over time (typically every few seconds) and is identified by the ISMACryp Key Indicator.

The KMS is responsible for securely carrying and delivering the content encryption key (i.e. the TEK) (and the optional authentication key) and its identifier to authorized terminals using adequate means (typically, ECMs) in a timely fashion (i.e. before it is needed). Such means are outside the scope of the present document.

A.3.5 Key Management and SRTP

SRTP [32] MAY be used to encrypt an RTP stream in conjunction with any KMS, as specified in clause 6.3. SRTP makes use of a 128 bit AES key (TEK), derived from a 128 AES Master Key (according to [32] and according to the limitations specified in clause 6.3) to perform encryption of the content. This key SHOULD change rapidly over time (typically every few seconds) and is identified by the Master Key Indicator (MKI) carried in the SRTP header.

The KMS is responsible for securely carrying and delivering the following information to authorized terminals using adequate means (typically, the ECM) in a timely fashion (i.e. before it is needed):

- the Master Key;
- the Master Key Indicator (MKI).

Such means are outside the scope of the present document.

The MKI SHALL be signalled in each SRTP packet, following the encrypted payload, as specified in [32]. The length, in bytes, of the MKI SHALL be greater than or equal to 1 and less than or equal to 4 and SHALL be signalled in the SDP description using an attribute of either the session (applicable to all SRTP-encrypted RTP streams) or of the media stream (applicable to this RTP stream only):

```
a=SRTPMKILength:n
```

Reference [38] defines different modes of the Rollover Counter Carrying (RCC) transform, which are implemented as SRTP authentication algorithms. The mode of RCC transform used SHALL be signalled in the SDP description using an attribute of either the session (applicable to all SRTP-encrypted RTP streams) or of the media stream (applicable to this RTP stream only):

```
a=SRTPAuthentication:n
```

where n is the SRTP authentication algorithm value for the mode of the RCC transform used, as specified in table 1 of [38].

The value of the ROC transmission rate parameter, R, SHALL be signalled in the SDP description using an attribute of either the session (applicable to all SRTP-encrypted RTP streams) or of the media stream (applicable to this RTP stream only):

```
a=SRTPROCTxRate:n
```

where n SHALL be an integer between 1 and 65535 inclusive.

EXAMPLE:

```
m=video 0 RTP/SAVP 96
a=rtpmap:96 H264/90000
a=mid:1
a=SRTPMKILength:4
a=SRTPAuthentication:2
a=SRTPROCTxRate:10
a=IPDCKSMStream:13
```

The SDP example is just a "skeleton" and does not include all required attributes.

A.4 SimulCrypt signalling for OMA-DCF files

In the context of DVB IPDC over DVB-H, content confidentiality is provided by encryption. Two categories of contents are considered: audio-visual content and generic content. The encryption of the first category is specified in clause 6.2.2, the encryption of the second category is specified below.

A.4.1 Overview

This clause specifies an encryption method and generic container that support protection of downloaded files. This encryption system is applicable to any finite-length file, irrespective of its internal format, that is to be considered as a whole by the receiver and for which a complete and error-free reception is required prior to consumption (e.g. a picture, ring tone, application etc). It SHOULD NOT be used to encrypt audio-visual content for which a dedicated encryption standard has been defined, even if this content is of finite length and meant to be downloaded in its entirety prior to consumption (e.g. audio-visual content that needs to be stored encrypted in a file and broadcast as a whole rather than streamed over RTP SHOULD NOT be encrypted as generic content, but SHOULD instead be encrypted using ISMACryp [11] for MP4 files as specified in clause 6.2.2).

The encryption system is based upon the OMA DRM Content Format (DCF) v2.0 specification [8]. It supports the use of different encryption keys for different parts of the file and it supports access control to those keys by multiple KMSs (SimulCrypt).

A.4.2 Encryption and signalling

The OMA DCF specification [8] SHALL be followed, except where stated in this clause.

The high-level overview of the DCF file structure is depicted in figure A.5. The mandatory parts of the file are shown as boxes with solid outlines. Optional structures are shown with dashed outlines.

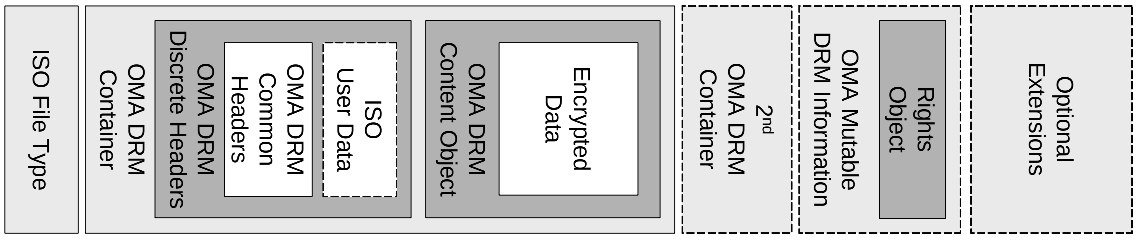


Figure A.5: DCF File Structure

A.4.2.1 OMA DCF adaptation

The value of the RightsIssuerURL[] in the OMA DRM Common Headers box SHALL be empty and RightsIssuerURLLength SHALL be set to 0.

At least one additional DVBECM box SHALL be included as an Extended Header within the OMA DRM Common Headers box, as shown in figure A.6. Additional DVBECM boxes MAY be added to support Simulcrypting of the protected content.

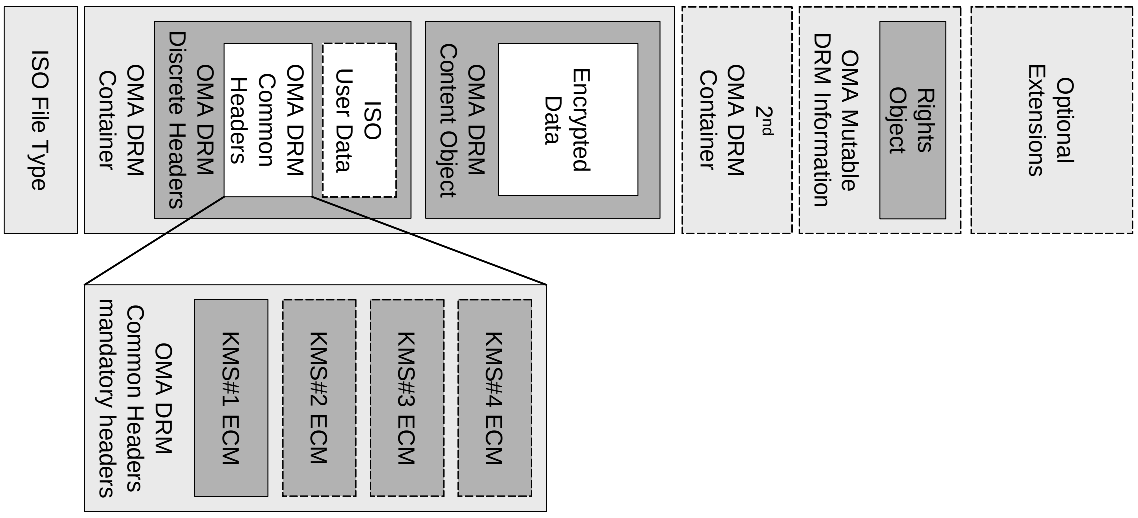


Figure A.6: ECMs as Extended Headers

The syntax of the DVBECM box is:

```
aligned(8) class DVBECM extends FullBox("decm", version, 0) {
   bit(16)      CA_system_id;
   bit(16)      OperatorId;
   byte[]       ECM;            // the actual value of the ECM
}
```

`CA_system_id` identifies the KMS applicable for the associated ECM, as defined in [39] and [14].

`OperatorId` identifies the operator using the associated ECM. Allocations of the values of this field are under the control of the KMS identified by CA_system_id, allowing for differentiation between operators using the same KMS.

`ECM` contains the actual ECM data for the corresponding KMS which is valid for the associated DRM content (stored in the OMA DRM Content Object box following the headers).

One or more DVBRightAcquisition boxes Boxes MAY be added at the end of the file shown in figure A.7. Each box contains the necessary information to obtain corresponding rights for the file.

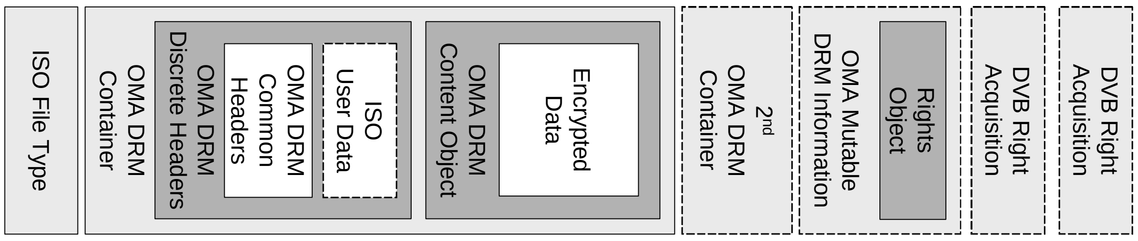


Figure A.7: Placement of DVBRightAcquisition boxes

The syntax of this box is:

```
aligned(8) class DVBRightAcquisition extends FullBox("dria", version, 0) {
   bit(16)              CA_system_id;
   bit(16)              OperatorId;
   unsigned int(16)     RightAcquisitionURLLength;
   char                 RightAcquisitionURL[];
   Box                  Extensions[];
}
```

`CA_system_id` identifies the KMS applicable for this box, as defined in [39] and [14].

`OperatorId` identifies the operator applicable for this box.

`RightAcquisitionURLLength` gives the length in bytes of the following `RightAcquisitionURL` field.

`RightAcquisitionURL` defines the Right Acquisition URL. It MAY be used by the Device to obtain rights for the protected content stored in this file. The mechanism by which these rights are obtained is outside the scope of the present document. The value of `RightAcquisitionURL` MUST conform to [3].

The `Extensions` field MAY be used to provide additional information relative to the protected file and the associated KMS.

A.5 Roaming

The Open Security Framework provides two mechanisms to allow Devices to roam from their Home Operator's network onto a Visited Operator's network.

The Open Security Framework inherently supports the commercially-used standard of DVB SimulCrypt [1], which provides the operator with a standard mechanism for interoperability between different KMSs. (This allows richer business models and better security as noted in clause A.1). In particular, this mechanism affords robust security for roaming as it utilizes the strength of all the KMSs without limiting the operator to a single one.

The second approach is to implement the generic roaming mechanism specified below, which provides for universal access to a baseline of services without requiring support on either the network or the Device for any specific KMS.

A.5.1 Roaming Overview

The Roaming mechanism and its support by operators is essential for interoperability between various IPDC over DVB-H networks. While the KMS chosen by an operator allows him to implement any business models, the Roaming mechanism ensures that visiting users may access at least a baseline of services without requiring support, in their Device or in their UICC, of the visited operator's KMS.

The players involved in the roaming scenario are the Home Operator, the Visited Operator and the receiving Device:

- The Home Operator (HO) is the operator that provides service in the Device's home network. The HO controls the KMS and KDA. If a UICC is present in the Device, the HO knows the secrets contained therein.
- The Visited Operator (VO) is the operator whose service the Device's owner wants to consume. Content services are unlike voice or data transport services since the channel package offered in any two markets is likely to be quite different.

A VO has a roaming service relationship with the HO, at least as far as billing is concerned. That roaming relationship is out of the scope of the present document.

A Device has a HO-specified KDA. When the Device owner roams to another market, there is a method in place by which they can choose services in the visited market. This can be automated if the Device has access to an interactivity channel; or alternatively, it may require a phone call.

A.5.2 Security Architecture

The roaming architecture supports authentication, confidential key distribution and data protection for roaming services. A protocol between the VO and HO is defined for Device authentication and to exchange keys used to protect the traffic between the VO and the Device.

The VO is responsible for applying the security mechanisms described in the following clauses, including the generation of the keys used to protect the traffic.

The HO is responsible for authenticating and authorizing a Device to access the roaming service.

An overview of the security architecture is depicted in figure A.8.

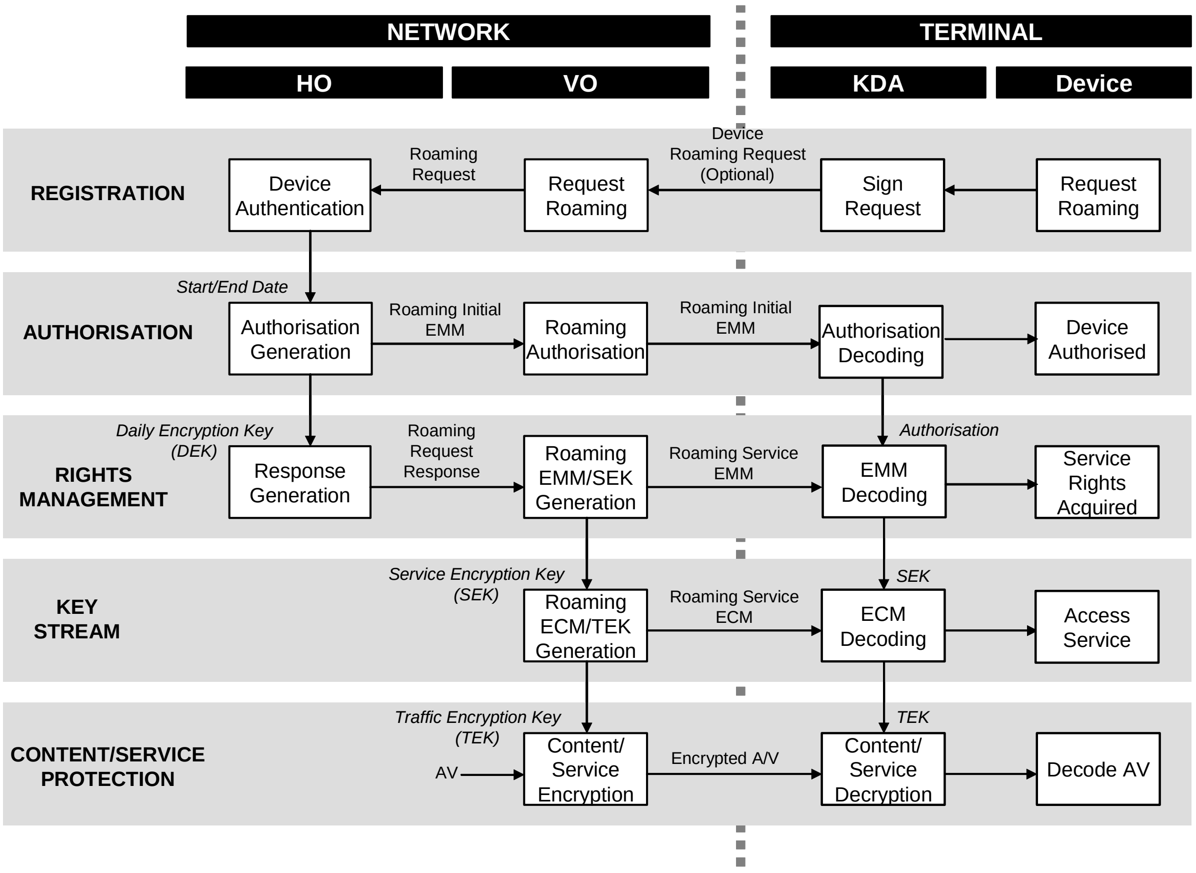


Figure A.8: Overview - Roaming Security Architecture

A.5.2.1 Registration

Service negotiation is carried out either via web pages or a phone call. A service package is selected by the subscriber and registration initiated.

Device registration shall take advantage of the interactivity channel, when available. The Device_Roaming_Request message is used by the Device to request roaming services from the VO. The Device_Roaming_Request message includes a Device signature, which is used by the HO to authenticate the Device.

The Device_Roaming_Request message is passed to the HO in a Roaming_Request message from the VO. The VO signs the Roaming_Request message to enable authenticated communications with the HO. The HO checks that the Device belongs to a valid subscriber and that the subscriber can be authorized for roaming services.

A.5.2.2 Authorization

The HO generates a Device authorization and series of Daily Encryption Keys (DEKs) for the requested roaming period and sends them along with the DEK Key Material to the VO in the Roaming_Request_Response message.

The VO sends the Device authorization and the certificate of the VOs public key in the Roaming_Initial_EMM message to the Device. This message includes a HO signature of the VOs public key and roaming date range, provided by the HO in the Roaming_Request_Response message. The VOs public key, now stored in the Device, enables authenticated communication between the VO and the Device and ensures signalling protection against unauthorized modification of the messaging.

The Roaming_Initial_EMM message authorizes the Device for roaming on the VO for a given period of time.

A.5.2.3 Rights Management

The VO is responsible for generating a SEK for each roaming service requiring service protection on a given day. The SEK is sent to the Device in the Roaming_Service_EMM message.

The key transfer is confidentiality protected by encrypting the SEK with the DEK for the specific day. The corresponding DEK Key Material is sent with the encrypted SEK within the Roaming_Service_EMM message. The VO signs this message with its public key to ensure data integrity and authentication of the received message by the Device.

The Roaming_Service_EMM provides the Device with the rights needed to access each authorized roaming service.

A.5.2.4 Key Stream

The VO generates TEKs for each roaming service, which are used by the scrambler to protect the content. The TEKs are sent to the Device in Roaming_Service_ECM messages. The VO creates a Roaming_Service_ECM message stream for each roaming service.

The VO operator ensures confidentiality of the TEKs by encrypting the contents of the Roaming_Service_ECM message with the SEK for the roaming service with which the message stream is associated.

The Roaming_Service_ECM includes an unencrypted roaming_service_id field to enable the Device to identify the SEK needed to decrypt the message.

The Roaming_Service_ECM messages are cycled to support random access of a roaming service. The message stream supports re-keying using the key synchronization mechanism of the scrambler.

A.5.2.5 Content protection

The VO encrypts the content with the TEKs. Re-keying of the TEKs by the VO provides protection against key-distribution attacks.

The details of the encryption schemes can be found in clause 6.

A.5.3 Key management and distribution

A.5.3.1 Roaming PKI infrastructure

The key management scheme is based on the assumption that there is a Public Key Infrastructure (PKI) in place, i.e. that any operator has a public/private 1 024 bit RSA key, and also has the public keys of all other operators with which it needs to interact. The methods by which those keys are distributed and authenticated are outside the scope of the present document. The operator keys are used for mutual authentication and encryption. In order to signal the public key itself , an ID of the public key is transferred in the protocol. The way to compute this ID would be the same as in RFC 3280 [9], section 4.2.1.2 (subject key identifier).

The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits).

The Public key signatures used in the protocol will all be RSA-PSS with SHA1 used as message digest (see [10]).

A.5.3.2 Daily encryption keys

The HO shall generate a set of DEKs for each Device requiring roaming. There shall be a DEK for each day that the Device is authorized for roaming. Those keys and all other symmetric keys following are 128 bit AES keys.

The DEK shall be generated by a KMS-specific one-way hash function:

DEK = HASH(roaming_random_data | roaming_date, device_id)

Where, roaming_random_data and roaming_date are provided to the VO with each DEK in the Roaming_Request_Response message and transmitted to the Device in the Roaming_Service_EMM message. In order to protect the DEKs in the Roaming Request Response they are sent encrypted using the VOs public key.

The security of the scheme is founded on the fact that the hash function is only known to the Device and the HO. Therefore it is strongly recommended that the hash function is implemented in the UICC secure hardware rather than the KDA in software.

A DEK shall be identified by a roaming_date in the Roaming_Service_EMM message. The roaming_date identifies the day during which the DEK is valid.

A.5.3.3 Service Encryption Keys

The VO shall generate SEKs for roaming services. One or more roaming services may share the same SEK.

The symmetric encryption scheme AES ECB shall be used to securely transmit the SEK to a Device using the DEK provided to the VO in the Roaming_Request_Response message: -

encrypted_SEK = AES-ECB.ENCRYPT(SEK, DEK).

The VO signs the Roaming_Service_EMM message used to transport the encrypted_SEK to the Device using their private key.

A SEK shall be identified by a roaming_service_id in the Roaming_Service_EMM and the Roaming_Service_ECM messages. The VO shall change the roaming_service_id each time there is a SEK key change.

A.5.3.4 Traffic Encryption Keys

The VO shall generate TEKs for roaming services. Each roaming service shall be encrypted by a sequence of time-varying TEKs.

The symmetric encryption scheme AES CBC mode shall be used to securely transmit the TEK to recipient Devices using the SEK generated by the VO.

The TEK shall be identified by meta information depending on the scrambling method in the Roaming_Service_ECM message. This meta-information shall be used to synchronize the TEK with the encrypted audio/video.

A.5.4 Key Generation and Validation at the Device

The Roaming_Initial_EMM is signed by the HO and authenticated by the KDA. This initial EMM enables the KDA to trust the VOs public key (It is essentially a certificate for the VO provided by the HO). The KDA then uses the VOs public key received in the initial EMM message to validate the Roaming_Service_EMM messages.

The KDA securely generates the DEK from the roaming_date, roaming_random_data and device_id transmitted in the Roaming_Service_EMM message. The KDA uses a KMS specific one-way hash function to generate the DEK.

The KDA securely generates the SEK from the Roaming_Service_EMM message using the DEK to decrypt the encrypted_SEK.

The KDA securely generates the TEKs from the Roaming_Service_ECM message using the appropriate SEK to decrypt the message. The required SEK is identified by the roaming_service_id within the Roaming_Service_ECM message.

The KDA sends the TEKs to the Device Descrambler.

A.5.5 Roaming messages

Before a Device is able to access content on a VO's network, the Device's HO needs to authorize it to roam onto the VO's network.

If the Device has interactivity channel capabilities, it can initiate this process by sending a Device Roaming Request message to the VO. The VO then forwards this request in a Roaming Request message to the Device's HO. Assuming the roaming is authorized, the HO transfers the Device authorizations and Daily Encryption Keys to the VO in a Roaming Request Response message. The VO then passes on the HO's authorizations to the Device in a Roaming Initial EMM. The Device is now able to establish authenticated communications with the VO and, thus, request and receive access to content on the VO's network.

If the Device does not have access to an interactivity channel, the user will need to contact the VO by other means (e.g. a voice call) to request authorization to roam. The VO will then send a Roaming Request message to the Device's HO based on information provided by the user, and the remainder of the authorization process is as described above.

A.5.5.1 Device Roaming Request

The Device Roaming Request message is used to request a service from the VO. This message is optional and is only used where the Device has interactivity channel capabilities. The purpose of the request is for the VO to prove to the HO that a Device has indeed requested service.

Table A.3

Syntax	No. of bits	Identifier
`ipdc_roaming_device_to_vo() {`		
`    device_id`	64	bslbf
`    ho_id`	32	bslbf
`    number_of_days_requested`	8	uimsbf
`    device_roaming_request`	var	bslbf
`}`		

Semantics:

device_id: ID of the Device requesting the service.

ho_id: ID of the home operator service provider.

number_of_days_requested: Number of days for which roaming services are requested.

device_roaming_request: opaque data for onward delivery to the HO by the VO, e.g. a signature verifying the request.

A.5.5.2 Roaming request

This message is used for the VO to ask the HO to authenticate and authorize the Device to access roaming services for the requested number of days.

Table A.4

Syntax	No. of bits	Identifier
`ipdc_roaming_vo_to_ho() {`		
`    device_id`	64	bslbf
`    ho_id`	32	bslbf
`    vo_id`	32	bslbf
`    number_of_days_requested`	8	uimsbf
`    request_type`	8	uimsbf
`    if request_type = 1 {`	32	
`        request_len`	16	bslbf
`        device_roaming_request`	var	bslbf
`    }`		
`    vo_pubkey_id`	see x509	bslbf
`    vo_signature`	1 024	bslbf
`}`		

Semantics:

device_id: ID of the Device requesting the service.

ho_id: ID of the home operator service provider.

vo_id: ID of the visiting service provider requesting the service for the Device.

number_of_days_request: Number of days for which roaming services are requested.

request_type: Is this a voice request (0) or a 2-way request (1).

request_len: Length of the appended request structure.

device_roaming_request: Roaming request from Device.

vo_pubkey_id: The ID of the VOs public key .

vo_signature: The VOs RSA-PSS signature on this request over the entire structure.

A.5.5.3 Roaming Request Response

The Roaming Request Response is used to transfer the Device authorizations and DEKs from the HO to the VO.

Table A.5

Syntax	No. of bits	Identifier
`ipdc_roaming_ho_to_vo() {`		
`    device_id`	64	bslbf
`    ho_id`	32	bslbf
`    vo_id`	32	bslbf
`    number_of_days`	8	uimsbf
`    encrypted message key`	1 024	bslbf
`    for (n=0; n < number_of_days; n++) {`		
`        roaming_date`	16	uimsbf
`        roaming_random_data`	112	bslbf
`        encrypted_daily_encryption_key`	128	bslbf
`    }`		
`    ho_signature_of_vo_pubkey_and_date_range`	128	bslbf
`    ho_pubkey_id`	160	bslbf
`    ho_signature`	1 024	bslbf
`}`		

Semantics:

device_id: ID of the Device requesting the service.

ho_id: ID of the home operator service provider.

vo_id: ID of the visiting service provider requesting the service for the Device.

encrypted_message_key: The 128 bit random message key, encrypted with the VOs public key. The RSA encryption method used shall be RSA PKCS v1.5 encryption as specified in RSA PKCS#1 standard [10].

number_of_days: Number of days for which keys are being given.

roaming_date: The date on which the key indicated in 'daily_encryption_key' is valid. The date format is the least-significant 16 bits of the MJD.

roaming_random_data: Random bits chosen by the HO to be used in the generation of the daily_encryption_key.

encrypted_daily_encryption_key: The DEK key corresponding to the roaming_date and roaming_random_data (see clause A.5.3.2), encrypted with the message key using AES-ECB.

ho_signature_of_vo_pubkey_and_date_range: The HO KMS-specific signature of the public key of the VO concatenated with the date range of validity. This will be communicated to the KDA/SIM to serve as a certificate for the HO's public key and the date range. This can be unique to a specific Device. The public key binary representation to be signed will be the standard ASN.1 encoding of a public key such as used in X.509 certificates.

ho_pubkey_id: ID of the HOs public key used for producing ho_signature.

ho_signature: The HOs RSA-PSS signature on this response over the entire structure.

A.5.5.4 Roaming initial EMM

This message is used for the HO to authorize a Device for roaming from a VO for a given number of days. A specially designated CA _system_id is used to show that this is a roaming EMM.

Table A.6

Syntax	No. of bits	Identifier
`ipdc_roaming_initial_emm() {`		
`    Length`	14	uimsbf
`    Type`	2	uimsbf
`    device_id`	64	bslbf
`    vo_pubkey_id`	160	bslbf
`    vo_pubkey`	see x.509	bslbf
`    start_date`	16	uimsbf
`    end_date`	16	uimsbf
`    ho_signature_of_vo_pubkey_and_date_range`	128	bslbf
`}`		

Semantics:

length: The length of the ipdc_roaming_initial_emm structure (including this field).

type: A value of 0 signals a Roaming_Initial_EMM. A value of 1 signals a Roaming_Service_EMM (see below). Other values are reserved. EMMs signalled with other values MUST be ignored.

device_id: This field identifies the Device for which this EMM is addressed.

vo_pubkey_id: The public key ID of the VOs public key.

vo_pubkey: The public key of the VO, binary ASN.1 encoded as in ITU-T Recommendation X.509 [37].

start_date: Start date that the Device can accept keys from this VO (expressed as least significant 16 bits of MJD).

end_date: End date for accepting keys from this VO (expressed as least significant 16 bits of MJD).

ho_signature_of_vo_pubkey_and_date_range: See above.

A.5.5.5 Roaming service EMM

This message is used to entitle the Device to access roaming services and deliver the SEKs. A specially designated CA_system_id is used to show that this is a roaming EMM.

Table A.7

Syntax	No. of bits	Identifier
`ipdc_roaming_emm() {`		
`    length`	14	uimsbf
`    type`	2	uimsbf
`    device_id`	64	bslbf
`    CA_system_id`	16	uimsbf
`    roaming_date`	16	uimsbf
`    roaming_random_data`	112	bslbf
`    number_of_services`	8	uimsbf
`    for (n=0; n < number_of_services; n++) {`		
`          roaming_service_id`	32	uimsbf

Syntax	No. of bits	Identifier
encrypted_service_encryption_key	128	bslbf
}		
vo_pubkey_id	160	bslbf
vo _signature	1 024	bslbf
}		

Semantics:

length: The length of the ipdc_roaming_emm structure (including this field).

type: A value of 1 signals a Roaming_Service_EMM. A value of 0 signals a Roaming_Initial_EMM (see above). Other values are reserved. EMMs signalled with other values MUST be ignored.

device_id: This field identifies the Device for which this EMM is addressed.

CA_system_id: This 16-bit field identifies the KMS. Allocations of the value of this field are found in [39] and [14].

roaming_date: The date the key indicated in 'encrypted_service_encryption_key' is valid. (Expressed as lower 16 bits of MJD).

roaming_random_data: A random 112-bit number.

number_of_services: Number of services contained in this EMM.

roaming_service_id: The service ID for which the key that follows is valid.

encrypted_service_encryption_key: The service encryption key encrypted using the Daily Encryption Key that corresponds to the roaming_date and roaming_random_data (see clause A5.3.2) using AES ECB mode.

vo_pubkey_id: The VOs public key ID.

vo_signature: The VOs RSA-PSS signature over the entire ipdc_roaming_EMM structure.

NOTE: The roaming_date, roaming_random_data and DEK for that Device were all passed to the HO in the Roaming Request Response.

A.5.5.6 Roaming Service ECM

The Roaming_Service_ECM message contains the time at which the ECM was built, which acts both as a trusted source of time for the UICC and as the date at which the ECM is valid. It also contains the identifier of the service it gives access to, and a pair of Synchronization Information/TEK couples. The nature of the synchronization information is dependent on the scrambling method selected (ISMACryp, SRTP or IPSec). Note that no indication of the chosen method is signalled in the ECM itself, such information is derived from the signalling in the SDP. The ECM structure contained within the encrypted_section of the message is encrypted with the Service Encryption Key using AES 128 in CBC mode.

A specially designated CA_system_id is used to show that this is a roaming ECM.

Table A.8

Syntax	No. of bits	Identifier
ipdc_sync_ISMACryp() {		
key_indicator	var	uimsbf
}		
ipdc_sync_SRTP() {		
master_key_identifier	var	uimsbf
}		
ipdc_sync_IPSec() {		
security_parameter_index	32	uimsbf
}		
ipdc_roaming_ecm() {		
length	12	uimsbf

Syntax	No. of bits	Identifier
`    type`	2	uimsbf
`    encryption_method`	2	uimsbf
`      roaming_service_id`	32	uimsbf
`      {`		
`          time`	32	uimsbf
`          roaming_service_id`	32	uimsbf
`          sync_info_1`	var	ipdc_sync_*
`          traffic_encryption_key_1`	128	bslbf
`          sync_info_2`	var	ipdc_sync_*
`          traffic_encryption_key _2`	128	bslbf
`      } encrypted_section`		
`}`		

Semantics:

length: The length of the ipdc_roaming_ecm structure (including this field).

type: A value of 0 signals a Roaming_Service_ECM, other values are reserved. ECMs signalled with a type other than 0 MUST be ignored.

encryption_method: This field signals which encryption method is used on the media streams:
0 signals ISMACryp
1 signals SRTP
2 signals IPsec ESP
3 is reserved

time: The time at which this ECM was generated, in seconds since January 1, 1970, UTC, and the date of validity of the ECM.

roaming_service_id: The service ID for which the keys that follow are valid.

sync_info_n: The synchronization information associated with Key n. This information can take one of three forms, depending on the scrambling method used on the media streams. For ISMACryp, ipd_sync_ISMACryp SHALL be used, for SRTP, ipdc_sync_SRTP SHALL be used, for IPSec, ipdc_sync_IPSec SHALL be used. The semantics of those structures are described below.

traffic_encryption_key_n: Traffic encryption key n, identified by the sync_info preceding it.

Semantics of ipdc_sync_ISMACryp:

key_indicator: The ISMACryp Key Indicator, identifying the key used to encrypt the content. The length of this field is signalled in the SDP (see the ISMACryp specification) [11].

Semantics of ipdc_sync_SRTP:

master_key_identifier: The Master Key Identifier, as specified in RFC 3711 [32] The length of this field is signalled in the SDP, see clause A.3.5.

Semantic of ipdc_sync_IPsec:

security_parameter_index: The identifier of the Security Association, and consequently of the key used to encrypt the traffic.

A.6 UICC

A.6.1 Application IDentifier (AID)

All UICCs offering KMS functionality are identified as such via the Application IDentifier (AID). The structure of the AID is given in TS 101 220 [5].

For any Open Security Framework KMS Application:

- the Registered application provider IDentifier (RID) SHALL be 'A000000009';
- the Application code SHALL be '0101';
- the Application provider code SHALL be the CA_system_id assigned to the KMS (see [39] and [14], encoded as 4 hexadecimal digits. The coding is right justified and padded with "FF" on the left;
- the Application provider field is reserved for the KMS provider's use.

A.6.2 KMS Application Selection

The application and logical structure for the UICC is defined in TS 102 221 [4].

All UICCs contain, in the EF_{DIR} file at the MF level, the list of AIDs installed in the cards. For a UICC offering KMS functionality, the EF_{DIR} file contains a record with an AID corresponding to the KMS application (i.e. with the Application code equal to '0101' and the Application provider code equal to the KMS provider's CA_system_id).

A UICC may contain several applications. Typically the UICC would contain a USIM application and one or several KMS applications. Each application will have a specific AID referenced in a record of this EF_{DIR} file.

After UICC activation, the KDA creates an APDU connection to communicate with the KMS Application using, as a parameter, the corresponding AID of the application found in the EF_{DIR}.

There can be one active selectable application session on a given logical channel. Therefore, in order to activate the new KMS application session in parallel with the USIM application selected on channel 0, a new channel bus has to be opened using the MANAGE CHANNEL command. On this new channel, the KMS application may be selected. The MANAGE CHANNEL command is described in TS 102 221 [4].

A.7 Secure Authenticated Channel Protocol

This clause describes the mechanisms used to establish a Secure Authenticated Channel (SAC) between the KMS application and the Descrambler. For a secure implementation, it is recommended that the KMS application should reside on secure hardware, such as a UICC.

The SAC ensures the secrecy of the communication between the KMS and the descrambler and prevents TEK extraction by an eavesdropper. The SAC also provides authentication of the descrambler by the KMS, thus preventing unauthorized Devices, such as a PC, from extracting TEKs from the KMS. The SAC does not, however, provide authentication of the KMS by the descrambler, as protection of the descrambler against misuse is not the primary goal of the SAC. The authentication of the KMS application may be done by the KDA.

A.7.1 High level description of the SAC

The SAC is based on asymmetric cryptography. Each descrambler is assigned a unique triplet:

- A Unique Identifier.
- A Private Key, kept secret in the descrambler.
- A Public Key, made available to the KMS.

The descrambler makes its Unique Identifier available to the KDA through a public interface. This Unique ID may then be used by the KMS to request from its Headend (server) the corresponding Public Key of the descrambler. The manner in which the KMS communicates with its Headend and retrieves the Public Key of the descrambler is out of scope of the present document, but it may involve using the interactivity channel or, for unconnected Devices, it may be based on the user contacting the service provider (Call center, Web, etc) to communicate the descrambler Unique ID and request that the public key be sent to its KDA.

Once in possession of the Public Key of the descrambler, the KMS generates a random session key based on the descrambler's public key. The KMS then securely sends the key material to the descrambler, which generates the session key from it (using its secret key).

Once both parties share this session key, all TEKs are encrypted by the KMS before they are passed to the descrambler, which decrypts them before use.

A.7.2 The cryptographic keys and parameters

The cryptographic protocol is based on the El Gamal Key Agreement (half-certified Diffie-Helman), specified in ISO/IEC 11770-3 [2] and described in TS 101 220 [5], clause 12.51.

A.7.2.1 The Descrambler's keys

For each Descrambler, the following values are generated:

p - a 1 536-bit prime modulus.

q - a 160-bit prime divisor of $p-1$.

g - a generator of order q.

x - a randomly or pseudo-randomly generated integer with $0 < x < q$.

The public key is composed of $(p, q, g, g^x \bmod p)$, and the secret key is x.

A.7.3 The SAC protocol

A.7.3.1 Session key establishment

The KMS picks a random $0 < y < \mathbf{q}$, sends the key material $g^y \bmod p$ to the descrambler and computes $s = (g^x)^y \bmod p$.

The descrambler computes $s'' = (g^y)^x \bmod p = (g^x)^y \bmod p = s$.

At the end of this protocol the KMS and the descrambler share s, a 1 536 bit value. This is hashed into a 160 bit value and the 128 high order bits of the hash are extracted to obtain the session key k:

$$k = [\, SHA\text{-}1(s)]_{MSB\text{-}128}$$

A.7.3.2 Secure key exchange

Following the session key establishment, the KMS may load keys in the descrambler, encrypting them with the session key k:

$$m = AES(TEK, k)$$

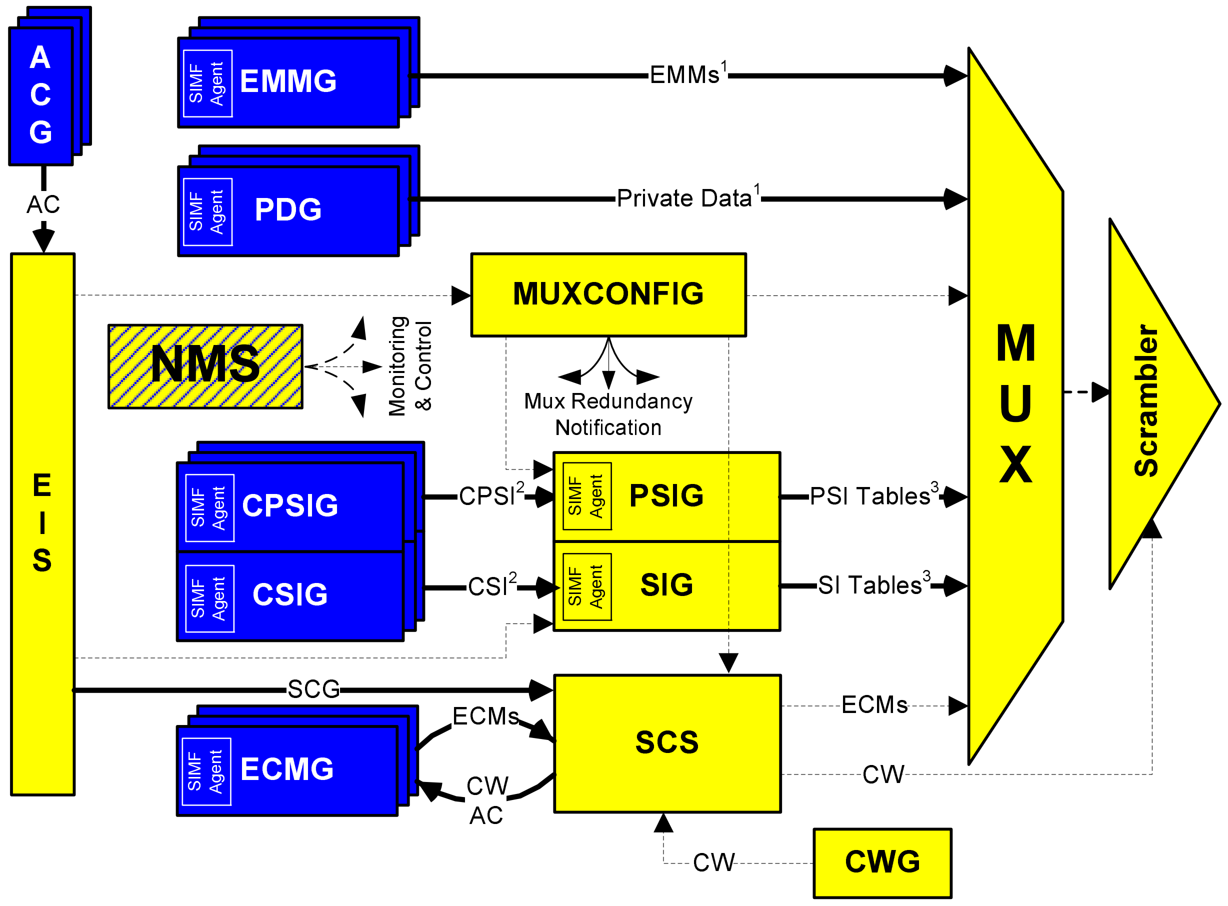
In the present document, both the TEK and the session key are 128 bit keys and AES is used directly on the TEK in ECB mode.

A.8 Adaptation of DVB Simulcrypt interfaces to the DVB-H Environment

This clause describes how the DVB SimulCrypt interfaces may be adapted to the DVB IPDC over DVB-H environment, according to the DVB IPDC over DVB-H specifications.

A.8.1 Reference DVB-Headend Architecture

Figure A.9 illustrates DVB-Headend reference architecture as described in TS 103 197 [1].



AC	= Access Criteria	**ACG**	= Access Criteria Generator
CPSIG	= Custom Program Specific Info Generator	**CSIG**	= Custom Service Information Generator
CWG	= Control Word Generator	**ECMG**	= Entitlement Control Message Generator
EIS	= Event Information System	**EMMG**	= Entitlement Management Message Generator
NMS	= Network Management System	**PDG**	= Private Data Generator
PSIG	= Program Specific Information Generator	**SCG**	= Scrambling Control Group
SIG	= Service Information Generator	**SIMF**	= Simulcrypt Integrated Management Framework

⟶	Defined in this specification
⇢	Proprietary; not defined
🟦	Simulcrypt CA components
🟨	Host head-end components
▨	Network Management System

NOTE 1: EMMG⇔MUX.
NOTE 2: C(P)SIG⇔(P)SIG.
NOTE 3: (P)SIG⇔MUX.

Figure A.9: Reference DVB Headend Architecture

NOTE: The TEK in the present document is synonymous with the CW in the DVB Headend specification.

A.8.2 DVB-H Headend Architecture and Interfaces

Figure A.10 illustrates how the reference architecture may be mapped to the DVB-H Headend.

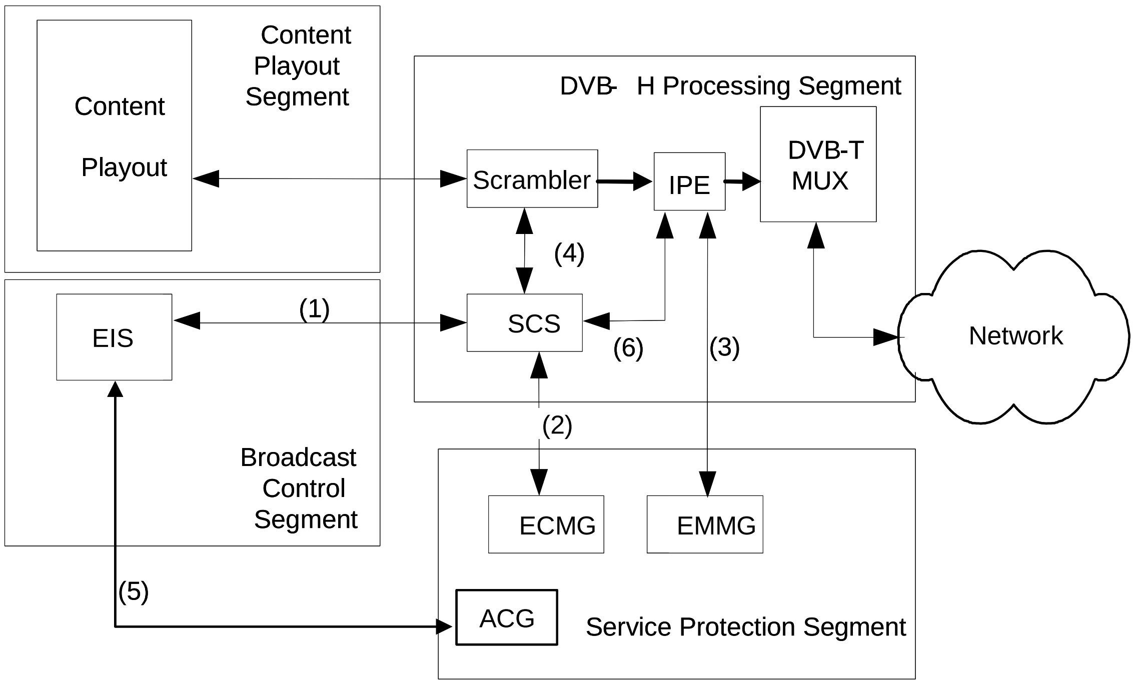


Figure A.10: DVB-H Headend Architecture

The numbered interfaces in Figure A.10 should be implemented as recommended below:

1) Interface EIS⇔SCS should be implemented according to TS 103 197 [1], clause 10.

2) Interface ECMG⇔SCS should be implemented according to TS 103 197 [1], clause 5.

3) Interface EMMG⇔IPE should be implemented according to TS 103 197 [1], clause 6.

4) Interface SCS⇔Scrambler is not defined in [1] and may be proprietary per ISMACryp Scrambler provider.

5) Interface ACG⇔EIS should be implemented according to TS 103 197 [1], clause 11.

6) Interface SCS⇔IPE is not defined in [1] and may be proprietary per IPE provider.

A.8.3 DVB-H Headend Architecture for Roaming Support

Figure A.11 depicts the recommended DVB-H Headend architecture enhanced to include roaming support.

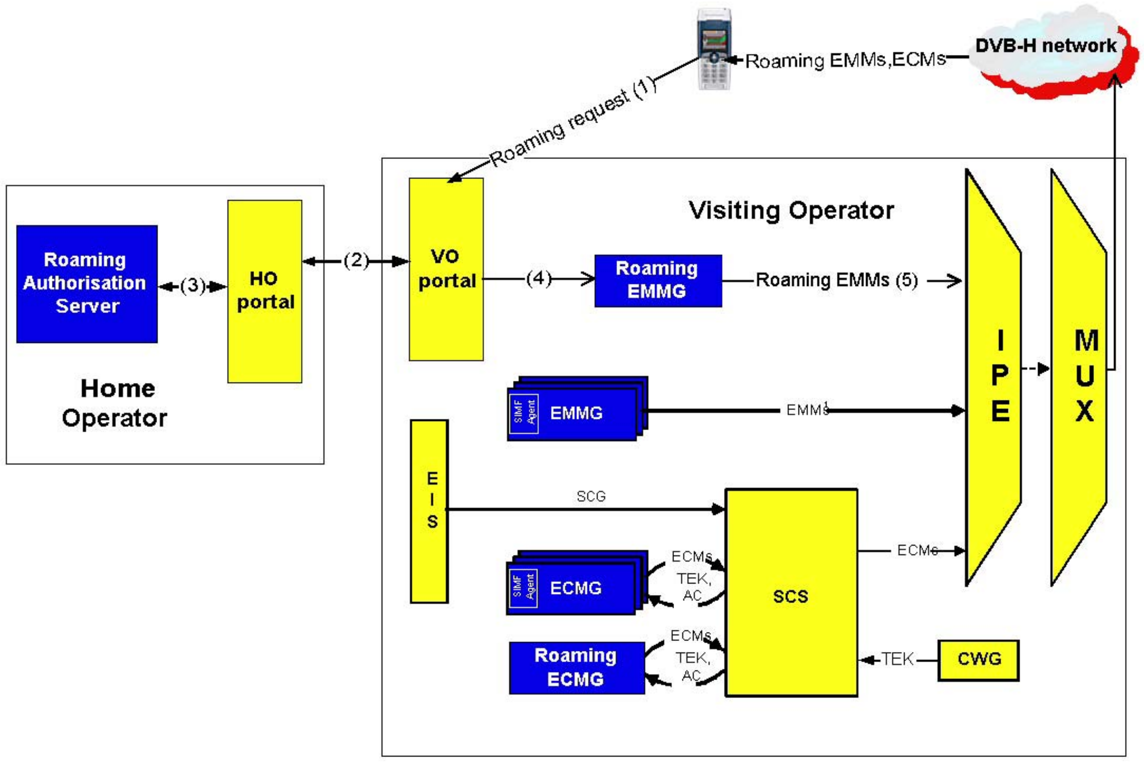


Figure A.11: DVB-H Headend Architecture including roaming support

A.8.3.1 Roaming ECMs

The special 'Roaming ECMG' should be supplied to generate Roaming ECMs. This ECMG should utilize the dedicated Roaming CA_system_id. (See [39] and [14]).

The Access Criteria for Roaming ECM generation should contain the roaming_service_id for each ECMG stream.

The Roaming ECMG should utilize the ECMG⇔SCS protocol defined in [1] to communicate with the SCS. In the Channel_setup message the Roaming ECMG should be configured to receive both the current and next TEK.

A.8.3.2 Roaming EMMs

The following interface reference numbers correspond to those in figure A.11.

1) This interface is used to transfer Device Roaming Request messages (see clause A.5.5.1) from roaming Devices to the VO. The protocol can be standardized as XML over HTTP.

2) This interface is used to transfer Roaming Request messages (see clause A.5.5.2) and Roaming Request Response messages (see clause A.5.5.3) between the VO and the HO. The protocol can be standardized as XML over HTTP.

3) This interface is used for communication between HO portal and the HO's Roaming Authorization Server. The protocol utilized is proprietary to the HO.

4) This interface is used to transfer information for generation Roaming Initial EMMs (see clause A.5.5.4) and Roaming Service EMMs (see clause A.5.5.5) from the VO portal to the Roaming EMMG. The protocol can be standardized as XML over HTTP.

5) This interface is used to deliver Roaming EMMs to the IPE for onward delivery to the roaming Device. The EMMG⇔MUX protocol defined in [1] should be utilized.

A.9 Mobile Device Security Framework

This clause provides a description of the Mobile Device Security Framework for IPDC over DVB-H which complements the Open Security Framework. Implementation of the Mobile Device Security Framework is an optional extension of the Open Security Framework. It specifies a set of standard security services for supporting varied KMSs, allowing for the horizontal deployment of Devices, supports KMS interoperability, and facilitates secure mobile Device implementations.

The same approach has been taken in defining the Mobile Device Security Framework as that governing the development of the Open Security Framework itself. According to this approach, a number of basic building blocks are standardized to provide interoperability. These basic building blocks must be included in all implementations of the Mobile Device Security Framework. Additional non-standardized blocks may be used where required for added value. These additional non-standardized blocks may be hooked into the system using a number of standard interoperability points.

A.9.1 Key Management System Device Agent

A.9.1.1 Overview

The security mechanisms employed by the KDA are summarized as follows:

1) Reception and processing of the Rights Management stream - Entitlement Management Messages (EMMs).

2) Service access control - decryption of the Encrypted TEKs - Entitlement Control Messages (ECMs).

3) Decryption of the encrypted media stream based on the three ciphers specified in clause 6.

The KDA is implemented as a Java application, which runs on the "KDA platform". A KDA is required by the KMS to operate with the IPDC application on the Device to protect access to IPDC services and support a wide range of purchase models, such as subscription, pay-per-view, rental etc, according to the operator defined business rules.

The KDA interacts with the descrambler and the KMS application on the UICC to realize the KMS-specific security functions. The KDA allows a Secure Authenticated Channel (SAC) to be established between the KMS application and the descrambler for secure delivery of the TEKs within the Device. As such, the KDA Platform supports interfaces to the KDA to allow communication with the descrambler and the UICC. Additionally, the KDA application may access the interactivity channel, if available, for direct communication with the KMS.

The interface that the KDA application exposes to the IPDC application is KMS-specific. However, a standard KDA interface, whilst beyond the scope of the present document, is not precluded. Such an interface could be defined to support additional interoperability between the IPDC and KDA applications (JSR 211 [42] defines a content handler API that may be suitable for this purpose). However, any such API will naturally restrict the functionality of the KDA that implements it and, as such, is not recommended for most applications.

The KDA platform supports the KDA application being deployed according to the following scenarios:

1) A library component integrated within a single IPDC Java application.

2) A standalone Java application that runs concurrently with an IPDC Java application.

3) A standalone Java application that runs concurrently with an IPDC native application.

Additionally, the present document does not preclude Device manufacturers from deploying native KDA implementations in addition to supporting KDAs via the KDA platform.

A.9.1.2 J2ME Mobile Information Device Profile (MIDP)

J2ME, the Java 2 platform Micro Edition, is the Java runtime that is optimized for portable Devices. It has become commonplace in almost all mobile phones and PDAs, and is thus a natural choice for the KDA Platform, allowing terminal vendors to reuse the existing J2ME platform already present on their Devices.

J2ME is available in the following configurations which detail the basic functionality of the virtual machine:

- CDC (Connected Device Configuration) [56].
- CLDC (Connected Limited Device Configuration) [54].

CLDC is the more prevalent configuration for portable Devices.

J2ME defines several profiles, which specify the higher level API and behaviour of the J2ME VM environment. The Mobile Information Device Profile - version 2 [40] is the common profile found in portable Devices today, and is the profile that the KDA Platform is built on.

A.9.2 KDA Platform

This clause describes a Java Platform required to support the KDA. The purpose of the Java platform is to allow different KDAs access to the security resources of the Device using standard Application Program Interfaces (APIs). The KDA Platform is specified as an extension to the widely deployed MIDP 2.0 profile of J2ME.

The KDA shall be implemented as a MIDlet according to this profile. The mobile Device shall provide a J2ME MIDP2.0 platform for running KDA applications.

Figure A.12 shows the KDA Platform.

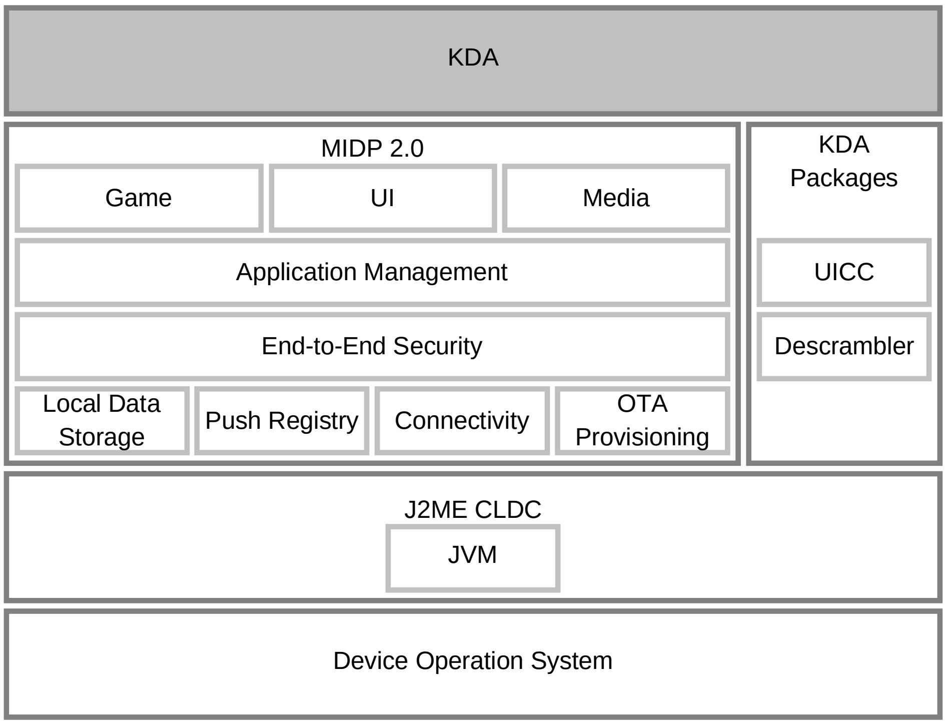


Figure A.12: KDA Platform

The KDA Platform extends the MIDP 2.0 profile (JSR 118 [40]) with several extension APIs to give access to the various system components that may be required by the KDA:

- The Descrambler API is provided as a means for controlling the Encrypted Media Descrambler component.
- The standard SATSA-APDU (JSR 177 [41]) is provided as a means of communication with Secure Hardware (SHW) such as a UICC.

KDAs may use other MIDP 2.0 resources. These will typically include:

- HTTP network access by means of the Generic Connection Framework (GCF) where available. It is important to note that certain Devices may have limited or no access to a two-way network connection. Therefore, implementers of KDAs are encouraged to enable the KDAs to support operation in a one-way broadcast environment.
- RMS storage for persistent data. RMS storage is typically not secure, and should be used for storing insensitive data only.
- MIDP 2.0 User Interface controls for OSDs and other user interaction.

The KDA Platform utilizes the J2ME over-the-air download capability for KDA provisioning, and extends this to support one-way delivery of KDAs over a broadcast network. All downloaded KDAs must be properly signed and provisioned with the necessary security permissions to access the KDA Platform extensions.

KDAs may appear in the normal J2ME application menu to allow users to delete them. KDAs may provide a user interface, which may be invoked from the normal startApp() method of the MIDlet.

A.9.2.1 Interactivity Channel

Where available, the MIDP 2.0 profile provides HTTP and HTTPS access for communicating via the interactivity channel over the two-way network. As such the KDA Platform provides the KDA with access to these protocols. Some Devices may not have an available back channel. KDAs should handle the exceptions generated by the Generic Connection Framework in such cases gracefully.

A.9.3 Security Background

The J2ME MIDP2.0 platform offers a restricted execution environment. Access to restricted APIs is granted according to the security policy on the Device. This policy defines protection domains, where each domain has a set of rules granting access to some or all of the restricted APIs. Each protection domain is typically associated with a root signing key-pair and certificate.

To provide a MIDlet with access to restricted functionality, the MIDlet must be put into the necessary protection domain. Putting a MIDlet into the protection domain is accomplished by signing the MIDlet with a private key for which the signer has a corresponding certificate chain whose root is the aforementioned protection domain root certificate. The signature and certificate chain are include with the MIDlet descriptor. When the MIDlet is installed the signature and the certificate chain are verified. If valid, the MIDlet is put into the protection domain associated with the root of the certificate chain.

Although KDAs may not be running in a completely secure environment, some security is necessary to prevent downloading of unauthorized KDAs. Unauthorized KDAs could enable denial of service attacks by stopping communication to the Key Management System or by not forwarding an EMM and an ECM to the UICC. The KDA has an interface for communicating with the UICC. Protecting the elements that interfaces with the UICC enables increasing the security of the KMS's secrets.

The KDA Platform APIs encapsulate access to all resources in the Device necessary for running a KDA MIDlet. All KDA specific extension APIs must be restricted to a KDA protection domain. All KDA specific extension APIs must be completely disallowed in ALL other protection domains. A user must not be prompted if a MIDlet in any other protection domain attempts to access KDA extensions.

The KDA protection domain root will be used to provide signed KDAs. This domain must also grant access to necessary standard restricted interfaces, such as SATSA (JSR 177 [41]), HTTP and HTTPS.

The J2ME MIDP2.0 platform provides for authentication of a KDA MIDlet during download. The KDA MIDlet shall be stored as a digitally signed JAR file (the format used to download them). Thus the same security model will be used when the KDAlets are downloaded or retrieved from the local storage. The security model is described in JSR 118 [40] RSP.

The KDA platform may support the option to receive encrypted KDAs. This can be useful where some semi-sensitive aspects of the KMS are contained in the KDA.

A.9.4 KDA APIs

This clause specifies the APIs provided to the KDA by the KDA Platform.

A.9.4.1 UICC

The KDA Platform provides an implementation of SATSA-APDU (JSR 177 [41]) package for communication with the KMS application residing on a UICC. The KDA utilizes the SATSA-APDU for communication with ISO/IEC 7816-4 [7] compliant UICCs.

The present document does not define the specific commands that the KDA may send the UICC. These commands are KMS specific.

A.9.4.2 Generic Connection Framework

The Generic Connection Framework (GCF), is a J2ME API that provides a straightforward hierarchy of interfaces and classes to create connections (such as HTTP, datagram, or streams) and perform I/O.

As the name implies, the GCF provides a generic approach to connectivity. It is generic because it provides a common foundation API for all the basic connection types - for packet-based (data blocks) and stream-based (contiguous or sequence of data) input and output.

Even though MIDP 2.0 defines a number of connection types, HTTP/HTTPS are the only connection type vendors must support. To ensure interoperability of KDAs, the following additional connection types are mandated by the KDA platform:

1) javax.microedition.io.SocketConnection;

2) javax.microedition.io.UDPDatagramConnection.

These APIs may be used by the KDA for accessing ECM and EMM streams on the broadcast feed, as well as for communication via the interactivity channel.

A.9.4.3 Descrambler

The Descrambler APIs provides methods for the KDA to securely load TEKs in the descrambler engine. The Descrambler APIs provides methods for negotiating an authenticated secure channel with either the KDA or the UICC. In most cases where UICC is present, the UICC will give out TEKs only over a secure channel with the Descrambler. These encrypted messages are delivered from the UICC to the Descrambler by the KDA, but the KDA cannot access or modify the keys.

The APIs define two java interfaces: the Descrambler interface and the DescramblerContext interface.

The first one represents the underlying descrambler and is used to establish a secure channel with it. As a rule, a platform will usually provide a single instance of a class implementing this interface and provide this instance to the KDA as part of the startup sequence of the IPDC platform.

The second represents a particular descrambler context, into which decryption keys will be loaded by the KDA using the secure channel previously established through the Descrambler interface. This allows for multiple DescramblerContext sharing the same secure channel to be instantiated in order to handle multiple descrambling sessions, each using different keys. One or more objects implementing the DescramblerContext interface are provided to the KDA by the platform, as part of the process of tuning to a protected service, and/or as the repository of the TEK contained in an ECM.

A.9.4.3.1 Descrambler interface

The Descrambler interface provides methods for establishing a secure connection with the underlying descrambler. This connection must be established before the descrambler may be used. The cryptographic protocol underlying the following API is described in clause A.7.

```
package dvb.cbms;

/**
 * Represents the descrambler engine, with which a secure connection must
 * be established before it may be used.
 */
public interface Descrambler
{
   /** Retrieves the Descrambler's unique identifier, which can, in turn
    *  be used to retrieve the descrambler's public key from a trusted
    *  source.
    *  @return A byte array containing the public ID
    */
   public byte[] getPublicID();

   /** Loads a new session key in the Descrambler
   *  @param keyMaterial is used to generate the session key.
    */
   public void loadSessionKey(byte[] keyMaterial);
}
```

The `getPublicID()` method returns the unique identifier of the descrambler. This identifier is assigned by the descrambler manufacturer and may be used by the KDA to retrieve the public key of the descrambler from a trusted source. The manner of this retrieval is outside the scope of the present document, but it must be noted that the establishment of the trust, by the KMS, in the descrambler resides entirely in the security of this operation and is the responsibility of the KMS.

The `loadSessionKey()` method is used to establish a session key between the descrambler and the KDA. The `keyMaterial` parameter is the value ***g^y mod p*** as defined in clause A.7.3.1. Once this method has been called at least once, the secure connection is considered as established and the DescramblerContext interface may be used to load keys in the descrambler. This method may be called at any time to establish a new session key.

A.9.4.3.2 DescramblerContext interface

The DescramblerContext interface provides methods for loading decryption keys in the descrambler.

The platform MUST support instantiating at least one DescramblerContext, and MAY allow instantiating more in order to support the descrambling of multiple streams.

Each DescramblerContext MUST support simultaneous loading of two keys with their associated key synchronization information in two separate memory slots. To allow the KDA to decide which key it needs to replace when loading a new key, the targeted slot is signalled in the loadKey() method.

```
package dvb.cbms;

/**
 * Represents a descrambler context.
 * Multiple descrambler contexts may be instantiated to handle decryption
 * of multiple streams with different keys.
 * Multiple instances of DescramblerContext may be related to the same
 * Descrambler instance, thereby sharing the same session key.
 */
public interface DescramblerContext
{
      /** Retrieves the identifier (mid) of the media stream this
   * context is associated with.
    */
   public byte mediaStreamIdentifier();

/** Loads a TEK into the Descrambler. The Descrambler must support the
    * simultaneous loading of a minimum of two keys to allow synchronized
    * switching of one key to the next.
    * Throws ArrayIndexOutOfBoundsException if the specified slot is not
    * available in this descrambler. Slots 0 and 1 MUST be supported.
    *
    * @param slot                The slot in which this key is stored
    * @param keyIndicator        The key indicator identifying this key
    * @param encKey              The TEK, encrypted with the session key
    *                            if a SAC is in use.
    * @param bypassSAC           True if SAC is not used.
    */
```

```
public void loadKey(byte slot, long keyIndicator, byte[] encKey,
                    boolean bypassSAC)
   throws ArrayIndexOutOfBoundsException;
```

The `mediaStreamIdentifier()` method is used to retrieve the unique identifier (`mid` parameter in the SDP) assigned to the media stream this context is associated with.

The `loadKey()` method is used to load a decryption key in the descrambler context.

The `encKey` parameter of the `loadKey()` method is the decryption key, encrypted with the current session key if a SAC is used (bypassSAC is false): the value ***m*** as defined in clause A.7.3.2.

The `keyIndicator` parameter of the `loadKey()` method uniquely identifies, for one particular DescramblerContext, the key to use to decrypt a block of data. This key indicator is signalled with the encrypted data. The same key indicator can be used in another DescramblerContext with a different key. The key indicator is not necessarily globally unique. Its nature depends on the descrambler being used: for ISMACryp it is the Key Indicator, for SRTP it is the Master Key Identifier, for IPsec it is the Security Parameter Index. In case the value to pass is encoded on less than 32 bits, the leading zeros are prepended to it in order to reach 32 bits, with the low order bits of the value stored in the low order bits of the `keyIndicator` parameter.

A.9.5 The KDA Life Cycle

The following methods of upgrading/replacing the KDA may be supported:

- A user may initiate a KDA upgrade from a Device menu.
- The KDA Platform may initiate a KDA download. This would happen, for example, when the platform determines that the KDA necessary for accessing requested content is not installed.
- The KDA may be replaced by firmware upgrade or application download via a PC (exactly the same as any other MIDlet).

A.9.5.1 Loading a new KDA

The KDA is a replaceable and upgradeable security component. This approach allows an operator to provide an OTA update for a compromised KDA or replace an active KDA with a KDA from another KMS vendor. Replacement of a KDA can be done by means of the broadcast stream or the interactivity channel (if present).

When accessible, the interactivity channel can be used for KDA installation/upgrade. An HTTP URL is provided to the KDA platform, which must return a JAD (Java Application Descriptor) that can be used to download the KDA using the standard OTA protocols.

The KDA may be delivered by means of the broadcast channel. This is the only way that one-way Devices can receive KDAs. A socket URL, such as socket://225.0.0.1:8001, must be provided by the KMS. The KDA Platform will then receive the new KDA MIDlet from the data carousel at the specified address.

Once the new KDA is downloaded, the KDA platform invokes the standard Java Application Manager to install the KDA, as specified in clause 6.3. During KDA upgrade, if the KDA name and protection domain has not changed, the Java Application Manager will leave all KDA persistent data stored in the RMS intact.

A.9.6 UICC - KMS Application Selection

After UICC activation (see TS 131 101 [6]), the KDA application creates an APDU connection to communicate with the KMS UICC application using the AID of the application as a parameter. The logical channel management is then handled by the JSR 177 [41] API implementation, which requests the UICC to allocate an unused logical channel.

The JSR 177 [41] API will perform the following commands when the APDUConnection method is invoked:

- MANAGE CHANNEL: There can be one active selectable application session on a given logical channel. Therefore in order to activate the new KMS application session in parallel with the USIM application selected on the channel 0, a new channel has to be opened using the MANAGE CHANNEL command.

- SELECT: On this new channel, the KMS application may be selected.
  The selection of the application is performed using the SELECT function with the AID of the selected KMS application as a parameter.

An APDUConnection supports exchange of APDU commands encoded in the format that conforms to ISO/IEC 7816-4 [7]. Each APDU connection has a logical channel reserved exclusively for it. Channel 0 is reserved for the USIM application.

Annex B (normative): 18Crypt

B.1 System Overview

B.1.1 General description of the system and elements

The present document describes a service protection system for services transmitted over an IP Datacast (IPDC) infrastructure. It enables access to services to be restricted to authorized users.

The solution is specific to IP Datacast channels. It can operate on either:

- The Internet Protocol (IP) level, based on the IPsec security standard, in which case it is transparent to IP based applications (such as video players); or
- The transport layer, using SRTP or ISMACryp. This allows direct storage of the content in encrypted form.

Additional application-specific content protection mechanisms could be freely combined with this solution, if desired, on top of the IP layer.

OMA DRM 2.0 is used as the default framework for rights management. In its most common form, OMA DRM 2.0 manages the rights to use files stored in a device; this solution extends that to the case of receiving streaming content over the broadcast channel. It also provides a means of performing rights management over a broadcast channel.

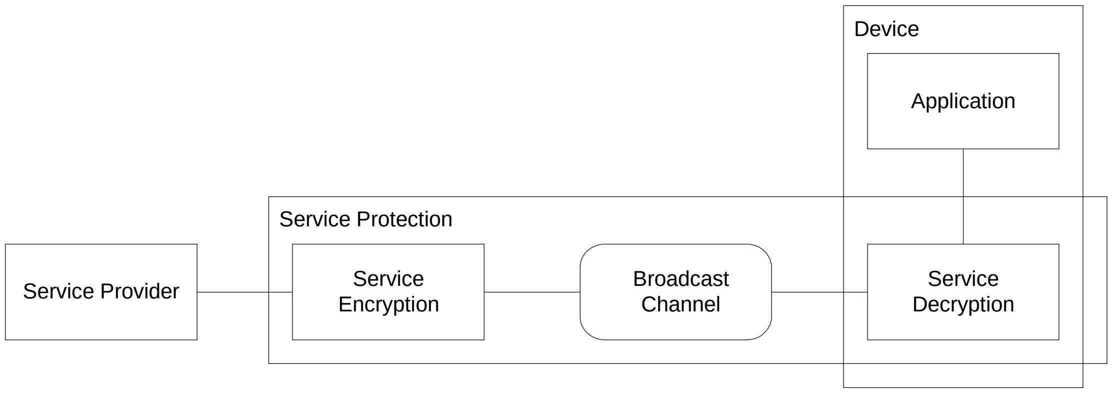


Figure B.1: System Overview

Figure B.1 shows the position of the service protection system within the overall architecture. IPsec allows the solution to be completely independent of the content format, while SRTP and ISMACryp provide alternatives for protecting content at the transport layer. Although IPsec, SRTP and ISMACryp are equally supported by the present document, the use of either IPsec or SRTP is strongly recommended.

At every level of the present document, special consideration has been given to reducing the power requirements of receiving devices.

B.1.1.1 Selected technologies

These are the main standards on which the solution is based:

- Advanced Encryption Standard (AES, see [17]) in the Cipher Block Chaining mode, for actual content encryption. Furthermore, OMA DRM uses AES-WRAP in its Rights Objects and optionally AES CBC-MAC.
- Secure Internet Protocol (IPsec, see [24]) using the Encapsulating Security Payload (ESP) protocol, for implementing service encryption and decryption as a function of the IP stack. Only transport mode is used.

- Secure Real Time Protocol (SRTP, see RFC 3711 [32]) as one option for implementing service protection at the transport layer. SRTP uses AES-CM (counter mode).
- ISMA Encryption and Authentication (ISMACryp, see [11]) as a second option for implementing service protection at the transport layer. ISMACryp also uses AES-CM (counter mode).
- A traffic key delivery protocol and management as specified in the present document.
- Open Mobile Alliance (OMA) Digital Rights Management version 2.0 (OMA DRM 2.0, see [49]) for managing rights to services, the associated service keys and the cryptographic protection of those keys themselves. The present document introduces some adaptations to OMA DRM 2.0 for IPDC Service Protection.
- Rights object delivery and device registration over a broadcast channel, without make any use of an interactivity channel, are also specified in the present document.
- Adapted Zero Message Broadcast according to [43] in 1-reselient mode.

The reasons for choosing these particular technologies as the basis of the solution include the following:

- AES is an efficient symmetric encryption method and an open standard which has hardware implementations. AES has many existing applications.
- IPsec/ESP is the standard way of keeping service decryption in receiving devices within the IP stack, invisible to the receiving applications, which thus remain independent of service protection and the carriers of the IP flows. Note that an IPDC specific broadcast channel is only one means to transmit such IP flows. IPsec/ESP has many existing applications.
- SRTP is a standard way of performing service decryption at receiving devices within the transport layer. It can be used to carry all common forms of streaming content, which can be stored by a device for later decryption, if required.
- ISMACryp is also a standard way of performing service decryption within the transport layer. It provides end-to-end protection for transport and storage of streaming content.
- The traffic key (i.e. the actual content encryption key) management framework and protocol are specified in the present document. The efficiency and robustness of the solution is achieved by a particular delivery protocol and management scheme for the frequently changing traffic keys.
- The developers of the 18Crypt Profile expect that OMA 2.0 DRM will be used for selling content and services in the cellular world and thus will be implemented by many devices.
- OMA DRM 2.0 however uses interaction over a two-way communication channel for device registration and guaranteed rights delivery. To adapt to broadcast devices, and to optimize the use of the broadcast channel, some new standardization is introduced.
- The main and foremost consideration for the system is network bandwidth efficiency, while maintaining realistic CPU requirements for the local device. By using zero message broadcast encryption, the network does not need to transmit keys to devices after registration. By choosing Fiat Naor there is a good balance between network bandwidth consumption and local device processing requirements.

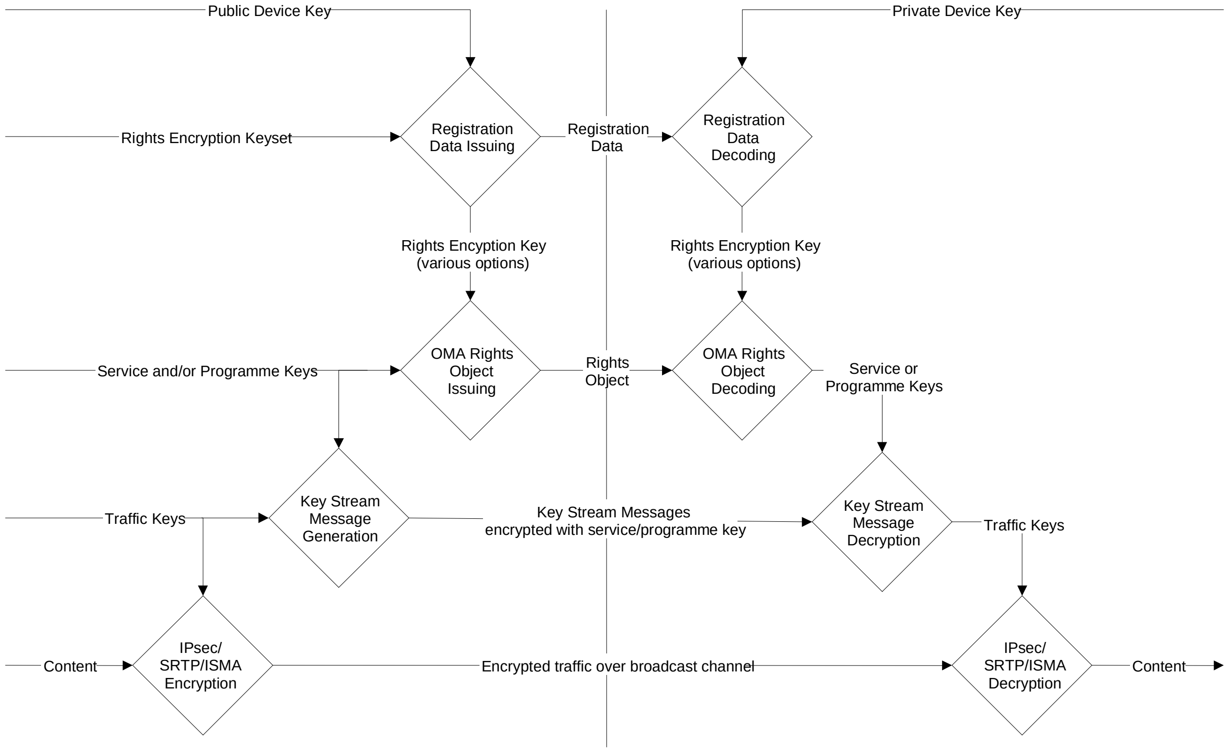
  The chosen group size is relatively small (with 256 or 512 devices), and depending on the group size(m) we will need:

  - device transmission and storage for 2log(m) Subscriber Group Keys (a.k.a. 'SGK's).
  - device computing power to derive (m-1) leaf keys.
  - device computing power to process, worst case, (m-1) leaf keys to create the 'IEK' key that covers the SEK.

Given the small size of the subscriber group, local device processing requirements are not considered as a problem.

Fiat Naor is used in 1-resilient mode, thereby making a collusion attack in theory possible. Making the scheme k-resilient, as discussed in [43] will increase the number of keys dramatically. The potential threat of 1-resilience is countered by measures described in clause B.2.6.3.

B.1.1.2 Overview of Operation



Public Device Key
Private Device Key
Rights Encryption Keyset
Registration Data Issuing
Registration Data
Registration Data Decoding
Rights Encyption Key (various options)
Rights Encryption Key (various options)
Service and/or Programme Keys
OMA Rights Object Issuing
Rights Object
OMA Rights Object Decoding
Service or Programme Keys
Traffic Keys
Key Stream Message Generation
Key Stream Messages encrypted with service/programme key
Key Stream Message Decryption
Traffic Keys
Content
IPsec/ SRTP/ISMA Encryption
Encrypted traffic over broadcast channel
IPsec/ SRTP/ISMA Decryption
Content

Figure B.2: Service protection via four layer model

As illustrated in figure B.2, the solution is based on a four-layer cryptographic architecture, with an optional optimization to provide both secure subscription and pay-per-view purchase options for a single service. Actual service encryption is carried out according to AES using 128 bit symmetric traffic keys.

Traffic keys are applied as part of standard IPsec security associations (SAs), as an SRTP master key, from which the session key is derived as per the SRTP specification, or as an ISMACrypKey, from which the session key is derived as per the ISMACryp specification. These are used by the IPsec, SRTP or ISMACryp layers to perform decryption automatically before passing the packets to the receiving application.

The traffic keys are not protected by IPsec, but instead encrypted with a service or programme key on the key stream layer, above the IP socket interface. These broadcast messages carrying traffic keys are called key stream messages.

Key stream messages can contain two levels of encryption. Separate programme and service keys have different lifetimes and can be used to provide, for a single service, different granularities of purchase periods to different users. This allows for the efficient implementation of both subscription and pay-per-view business models for the same service. Pay-per-view customers are provided with a programme key which is only valid for a single programme while subscribers are given a service key, valid for reception of the service for some longer period. Within the key stream message, the traffic key is encrypted with a programme key, and the programme key is also carried, encrypted with the service key. Thus, pay-per-view subscribers can directly decrypt the traffic key, while subscribers can decrypt the programme key using the service key, which can then be used to decrypt the traffic key.

Key stream messages contain extensions to content IDs, which are carried in the ESG, for the programme and/or service. Devices use this ID to identify which Rights Object contains the keys to use for key stream message decryption.

Where the two-level service and programme functionality is not required, the traffic key can be directly encrypted with either the service or programme key and the service-key-encrypted programme key omitted.

The service or programme key(s) are transmitted to each receiving device within OMA DRM 2.0 rights objects (ROs). Such transmission of ROs can be done in two different ways, depending on whether the receiving device can make use of a separate interactivity channel:

- via a broadcast channel; or
- by using the separate interactivity channel.

In both cases the ROs can be utilized by the customer device only, since the service or programme key sections are protected according to the OMA DRM 2.0 standard, or, in the broadcast case, by the variant of OMA DRM 2.0 described in the present document.

When delivering Rights Objects over the broadcast channel, bandwidth is a major constraint. The present document addresses this problem in two complementary ways. Firstly, a new binary form of the OMA DRM 2.0 Rights Object, called a Broadcast Rights Object (BCRO), is defined. Secondly, a method is described for securely delivering BCROs to groups of devices using a single broadcast message. Valuable portions of Rights Objects are protected by group or unit keys, and when necessary, Zero Message Broadcast encryption can be used to allow messages to be decrypted only by arbitrary sets of devices within a larger group.

An additional mechanism is available, as in OMA DRM 2.0, for Rights Objects to be issued to a group of devices known as a domain. It is expected that a domain will contain a number of devices belonging to the same user, and will be used by Rights Issuers to sell subscriptions allowing all devices within the domain to receive protected services.

Registration can be performed either via the interactivity or broadcast channels. In the case that the interactivity channel is used, the registration protocol is according to OMA DRM 2.0 and unit keys are delivered, protected with the public key of the device. The present document defines an efficient and user friendly process for the registration of devices which do not support an interactivity channel, and a new protocol, called the 1-pass ROAP, is defined for the delivery of unit, group and broadcast keys via the broadcast channel.

The service key protection thus is based, according to OMA DRM 2.0, on a public key cryptosystem where the public key of the customer device is registered at each Rights Issuer and the corresponding private key is kept within the customer device.

Note that the above assumes the use of OMA DRM 2.0. Should another DRM system be used, the Rights Management Layer and the Registration Layer would change accordingly.

B.1.2 The End-to-End System

This clause briefly describes the major roles within the system. For a more detailed description, see clause B.2.1.

Figure B.3 shows a simplified view of the major actors in the system and their relationship to each other.

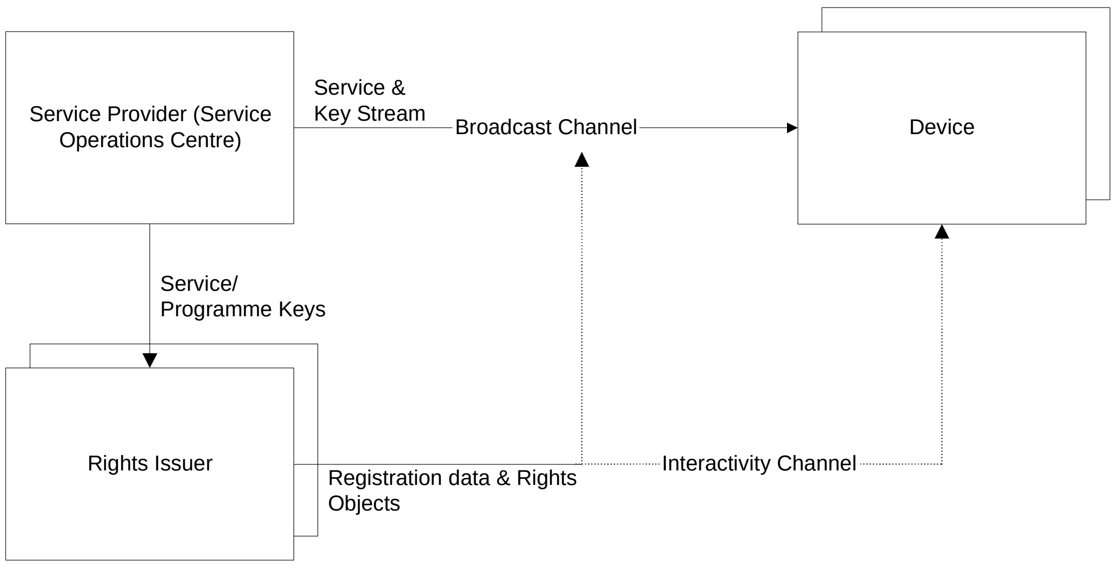


Figure B.3: Highly simplified view of the End-to-End system

The main actors are as follows:

- The Service Provider, also known as the Service Operations Centre, broadcasts and encrypts the service and the key stream. It provides service and programme keys to the Rights Issuers.
- The Rights Issuer registers devices and provides Rights Objects to devices allowing them to decrypt the services which they are entitled to receive.
- The device receives the service, decrypts it (if it has the necessary Rights Objects) and presents it to the user.

The interoperability point allows different Rights Issuers to use different DRM systems (or even the same Rights Issuer to use multiple DRM systems) to control access to the same broadcast service, without needing to broadcast the service or the key stream multiple times.

B.1.3 Modes of Operation and Types of Device

The present document supports populations of 'interactive', 'broadcast' and 'mixed-mode' devices.

- An *interactive device* supports some form of interactivity channel which can be used to communicate with Rights Issuers, e.g. GPRS. Messages from the device to a Rights Issuer and from the Rights Issuer to a device are sent via the interactivity channel. Services are received via the broadcast channel.
    - Reliable delivery of messages to interactive devices is possible. It is also possible to locate a device to a particular cell of an interactivity network, if this is required, although it should be noted that the location of the interactivity network cell to which a device is connected could be different to the location of the transmitter from which the device receives broadcast data.
- A *broadcast device* does not support an interactivity channel. Requests to Rights Issuers are made by the user via some out-of-band communication, such as a telephone call, an SMS message, a form on a website or some entirely different means. Messages from a Rights Issuer to the device are sent via the broadcast channel.
    - Reliable delivery of messages to broadcast devices may not always be possible, and it is typically not possible to locate a device. Therefore, messages to be delivered to broadcast devices are generally carried across the whole network, and regularly repeated.
- A *mixed-mode device* is also possible. Such a device supports operation both via the interactivity channel and via the broadcast channel.

Should a Rights Issuer use OMA DRM 2.0 to support the Rights Management and Registration Layers, interactive devices can use the existing 4-Pass ROAP from OMA DRM 2.0 [49] to register with a Rights Issuer and acquire Rights Objects. Some extensions to this scheme are provided in the present document to support the protection of broadcast services.

A new 1-Pass ROAP, called the 1-pass binary Push Device Registration Protocol, is defined for the delivery of registration data and Rights Objects to broadcast devices.

A Rights Issuer can choose to register mixed-mode devices to operate via the interactivity or broadcast channels, or both. It is also possible for the Rights Issuer to signal at any time which method will be used.

B.1.3.1 Unconnected Devices

[49] defines so called unconnected devices. These devices do not have direct access to an interactivity channel, but are capable of making a connection via an intermediary interactive device. [48] describes a method for these unconnected devices to register with a Rights Issuer as part of a domain, via an intermediary using the [47] protocol. The intermediary can then purchase domain Rights Objects and forward them to the unconnected device, embedded in a DCF. For Rights Objects referring to broadcast services, this DCF can otherwise be empty.

A broadcast device could potentially be able to use the OBEX facility to operate as a mixed-mode device. Furthermore, a device which would not otherwise be able to support the present document could potentially be able to become an interactive device. This leads to the following additional types of device.

- *Unconnected interactive device* (which can receive and decode protected services but lacks the ability to function as a full broadcast device as defined in the present document). As with any interactive device, the unconnected interactive device can be part of an OMA DRM 2.0 interactive domain and can use domain Rights Objects to receive protected services for those interactive domains.
- *Unconnected mixed-mode device* (which is a broadcast device, but can also act as an unconnected device for the acquisition of Rights Objects). The device can form part of two types of domains.
  - A broadcast domain which is supported by the BCRO and/or ICRO, in the usual manner.
  - An OMA DRM 2.0 interactive domain, which is supported by domain Rights carried in ICROs.

In both cases Rights Objects can be bought over an interactivity channel by an intermediary.

NOTE: For reasons of completeness: a mixed-mode device is, of course, also capable of buying rights out of band.

Figure B.4 shows Rights Issuer communication with various types of device.

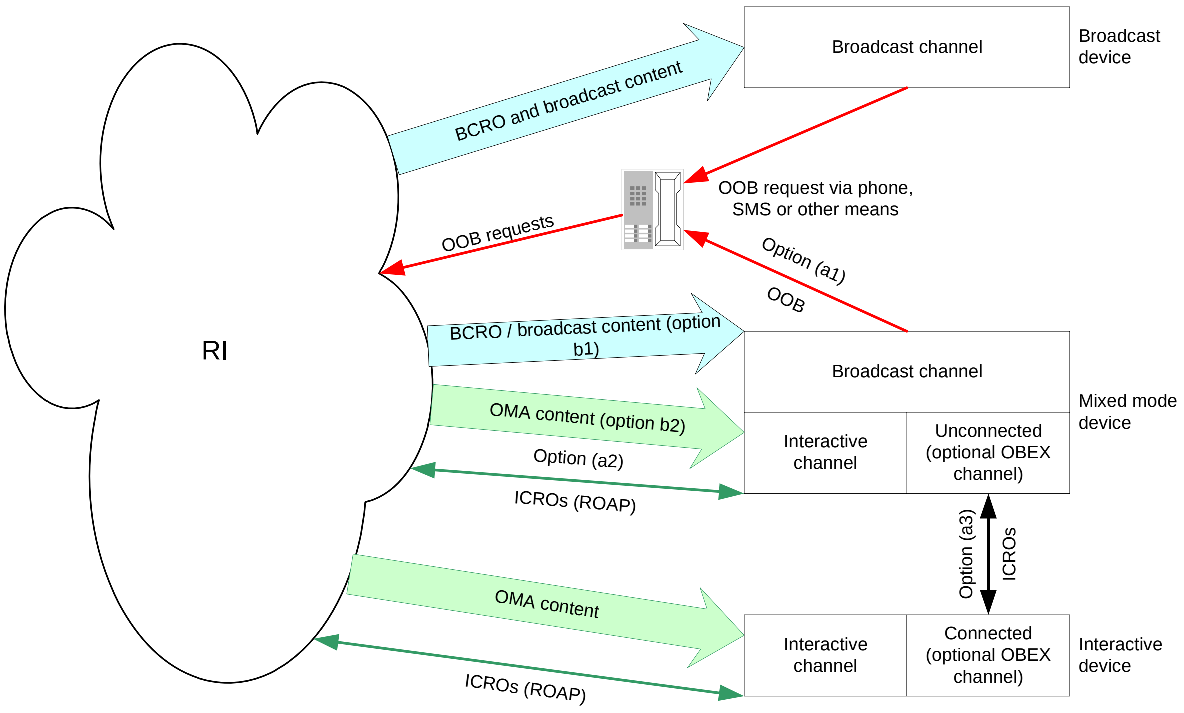


Figure B.4: Rights Issuer communication with various types of device

B.1.3.2 Scalability Considerations

The present document provides considerable scope for Service Providers to balance requirements for efficient bandwidth utilisation with the latency of delivering Rights Objects to a device, in order to build systems that are scalable to many millions of users. It also provides means for minimizing the power consumption of devices. The facilities provided for use with OMA DRM 2.0 include:

- The option to provide a service for the spontaneous, ad-hoc delivery of Rights Objects via the broadcast channel.
- The ability to broadcast the same Rights Objects to groups of devices in the same message.
- The ability to broadcast a carousel of Rights Objects, and to provide a schedule for this carousel, so that devices only need to listen to the carousel at the indicated times.

- The ability to broadcast Rights Objects in advance of when they will be used.
- The ability to make use of spare bandwidth that might be available at certain times to deliver Rights Objects.
- The ability for Rights Issuers to decide on appropriate service key life times.
- Rights Objects can be delivered via the broadcast or, when available, the interactivity channel.

B.1.4 Purchase steps

The customer device purchases a service protected by encryption by obtaining a Rights Object (RO) containing key material from a Rights Issuer. This is carried out with the help of an electronic commerce party, here called E-Commerce System. The fourth player of the purchase is the actual broadcaster of the service, especially its broadcast management which is responsible for generating the key material, which will be used for encrypting the actual service when broadcast.

Please note that for the sake of familiarity, in this clause vague, undefined terms like 'Broadcast Management' and 'E-Commerce System' are used for describing functions of the IPDC infrastructure. Later, in the normative clauses, they will be replaced by more specific and defined terms such as (respectively) 'Service Operation Centre (SOC)' and 'Customer Operation Centre (COC)'. The same applies to the term 'DRM Agent', which represents the implementation of the DRM system within the device.

The four parties involved in the purchase steps are shown in figure B.5, illustrating the purchase steps in case of an interactive device, capable of communicating with the E-Commerce System and the Rights Issuer across an interactivity channel.

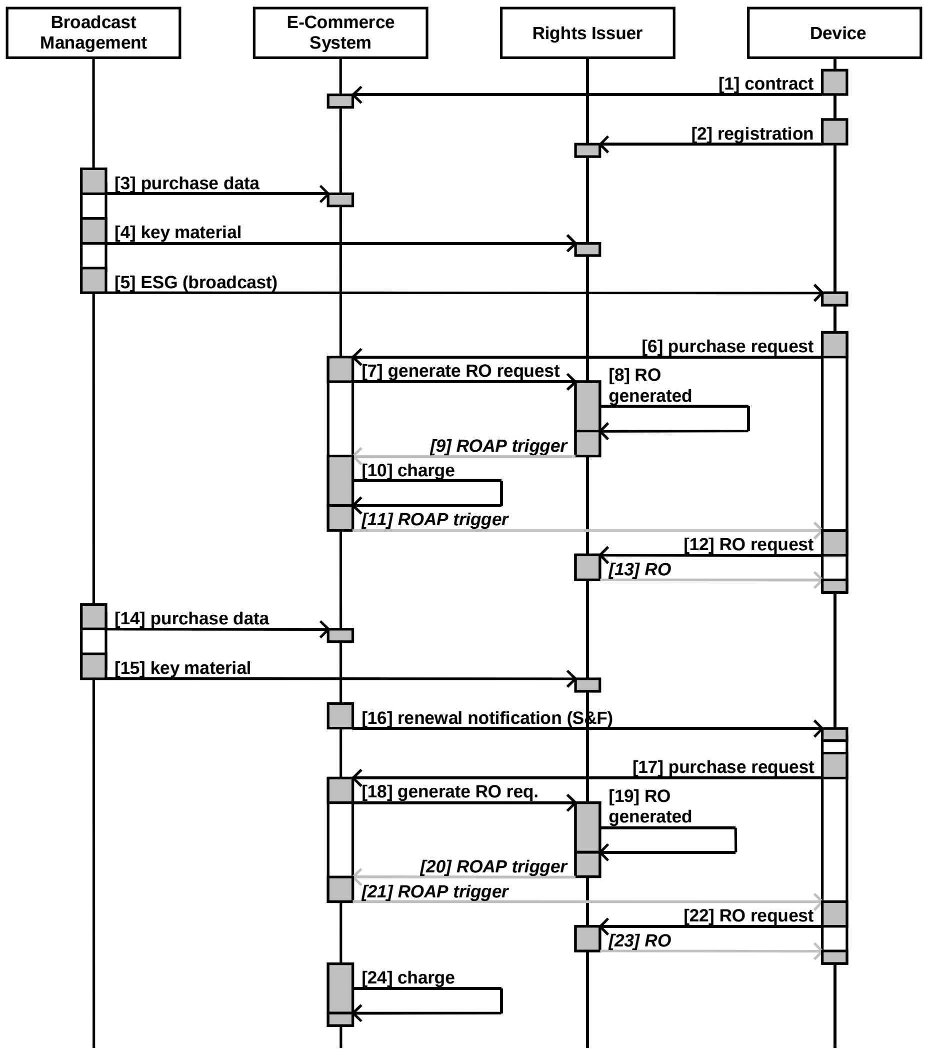


Figure B.5: Purchase steps in case of an interactive device

To enable purchase, device needs (1) a contractual relationship with the E-Commerce System, and (2) a DRM registration with the Rights Issuer. Thus both the E-Commerce system and Rights Issuer can authenticate the device whenever it interacts with them. Furthermore, the Rights Issuer knows the public key of the device, and can send key material to it via the interactivity channel in an RO which is encrypted by the public key - only the secure DRM agent in the device can decrypt the key material, using its secret private key.

Now, as the broadcaster prepares to broadcast a service, it distributes information about the service to the other parties: (3) data for purchasing the service to the E-Commerce System, (4) key material for decrypting the service (soon to be broadcast) to the Rights Issuer, and an (5) electronic service guide (ESG) to the device; the ESG is broadcast over the broadcast channel. Proper service identifiers in the distributed material make it possible to identify each service and service bundle, their purchase options, etc.

Based on the ESG, the human user of the device decides to purchase the service.

The device then (6) requests purchase of the service from the E-Commerce System, which (7) requests the Rights Issuer to (8) generate the RO and (9) return ROAP (Rights Object Acquisition Protocol) trigger data for retrieving the RO. The E-Commerce System then (10) charges the device and (11) returns the ROAP trigger to it. The device then uses the ROAP trigger to (12) request the RO from the Rights Issuer, (13) receiving the RO, encrypted by the public key of the device. All these interactions between the device and the network elements use the reliable interactivity channel, and (from the device) are carried out by a protocol like HTTP and HTTPS: the device issues requests and receives responses.

Please note that the present document does not specify the timing of customer charging by the E-Commerce System.

If the service is of a subscription type, eventually both (14) purchase data and (15) key material will change, and the device thus has to obtain the new key material in a renewed RO.

At this point, the E-Commerce System could notify the device of the renewal need by (16) sending a renewal notification to it, typically using some store-and-forward (S&F) mechanism of the interactivity channel, for instance the short message service in GSM networks. Use of the renewal notification is optional however, subject to the (1) contract with the E-Commerce System. Alternatively, the device could detect the need for RO renewal itself.

To renew the RO, the device repeats steps (6) through (13) of the original purchase, illustrated as steps (17) through (23) in figure B.5. Repeated charging of a continuous subscription as step (24) depends on the purchase in question: some continuous services could be charged repeatedly, while time limited services could be charged only once, despite the fact that key material is periodically changed in order to increase the security of service protection.

From the device point of view, due to the purchase steps, its secure DRM agent at all times holds an RO with the key material for decrypting the service, encrypted with the public key of the device, which only the DRM agent can decrypt using the secret private key it holds.

Let us now look at the differences between the above procedure, and the procedure for broadcast devices, illustrated in figure B.6.

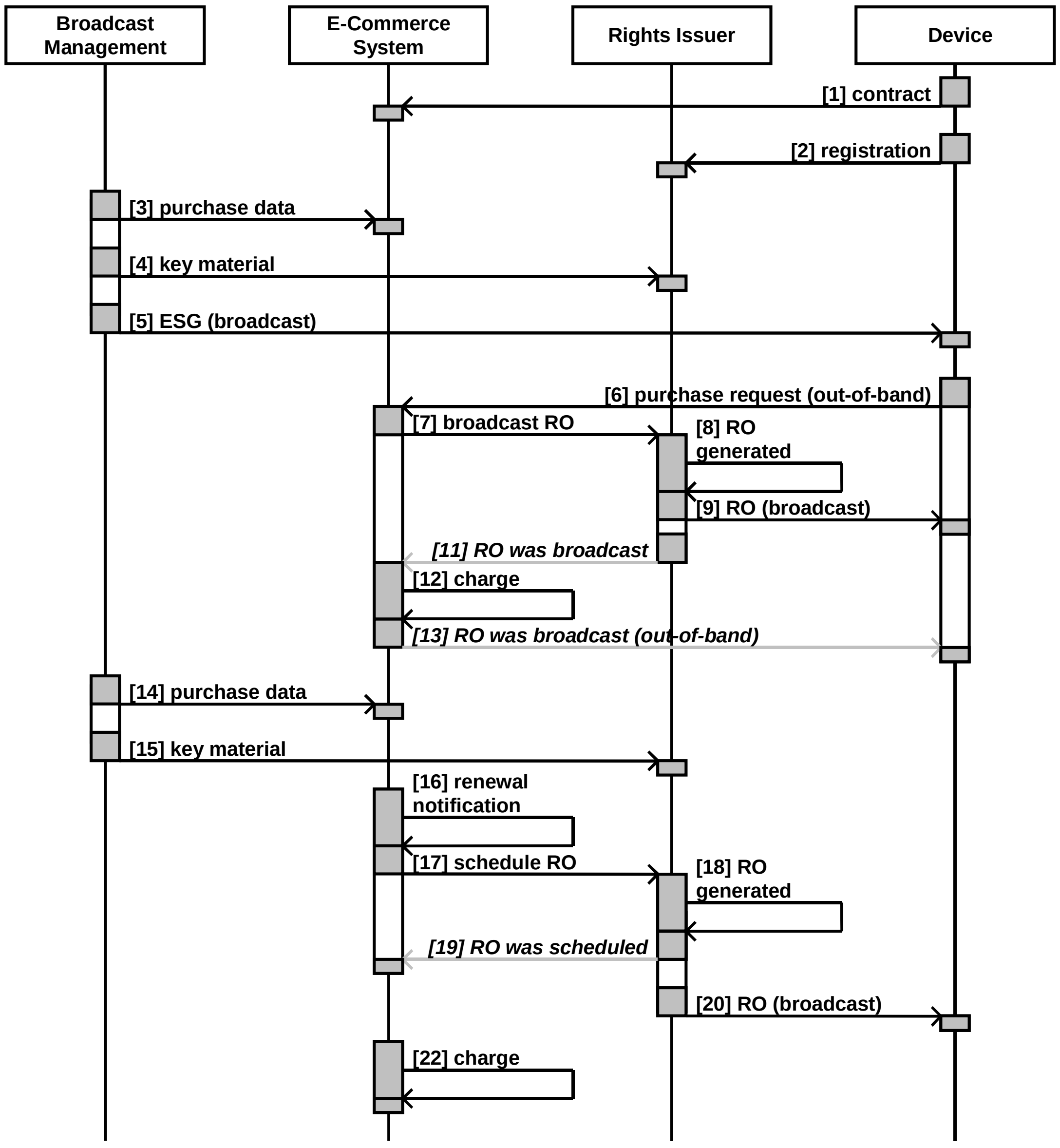


Figure B.6: Purchase steps in case of a broadcast device

A broadcast device does not have the interactivity channel for communicating with E-Commerce System and Rights Issuer; hence it has to make its purchase request (6) out of band and receive its ROs (9) and (20) over the broadcast channel.

The (1) contract with the E-Commerce System could be implicit in this case, since the E-Commerce System cannot authenticate the device; instead, the user of the device is authenticated to enable the payment. Still, the device has to be (2) registered to the Rights Issuer, in order to enable the production of ROs for it. In order to reduce the bandwidth required for RO broadcasts, Broadcast Rights Objects (BCROs) can be addressed to groups of devices instead of a single device. As the BCROs can be addressed to groups they cannot be encrypted with the public key of a single device. Instead they are encrypted with symmetric keys delivered to the devices at registration. In order to reduce the bandwidth even further the BCROs do not use the XML format but instead a binary format.

As before, service data is distributed to the parties in steps (3) through (5).

Now, the user of the device (6) requests the purchase out-of-band, possibly maintaining out-of-band connection until the confirmation (13) is received. The out-of-band connection could be a phone call to the Customer Care Centre of the broadcaster; a series of HTTP requests to a web shop (charging e.g. a credit card); or in some other way. The device could be listening to the broadcast channel at this point, to immediately receive the RO broadcast at step (9). However the RO broadcast can also be repeated in a carousel like manner giving devices the chance to receive the ROs at a later stage.

The following is one possible out-of-band purchase scenario, starting with the out-of-band purchase request (6). The E-commerce System (7) requests the Rights Issuer to (8) generate the RO, and to (9) broadcast it immediately. There is of course no (10) response about the device receiving it (and hence no interaction (10) in figure B.6 either). However, the Rights Issuer can (11) confirm the *attempt* of broadcast to the E-Commerce System, which will (12) charge the user, and could (13) pass the confirmation out-of-band to the human user of the device. At this point, the human user can verify that the device has indeed received the RO, and possibly request re-broadcast if that was not the case.

Thus, as *one possible* implementation, 'immediate broadcast' *can* be used when a new service is purchased.

Yet, when ROs are renewed, it is more convenient not to require out-of-band interaction, but to let the device receive the renewed ROs automatically. For this purpose, as a part of the broadcast channel air interface specification, delivery carousels for broadcasting ROs are specified, and a schedule for RO broadcast can be provided, allowing devices to determine the times when they listen for renewed ROs (and actually for first purchased ROs also, as an alternative to the 'immediate broadcast' scenario above).

Again, RO renewal is caused by (14) purchase data and/or (15) key material being changed.

Now (16) the renewal notification will again take place in the E-Commerce System, but it is not communicated to the device; instead, the E-Commerce System (17) requests Rights Issuer to (18) generate the renewed RO and to enter it into the broadcast carousel; the E-Commerce System then receives (19) confirmation thereof. In this case, the actual (20) ROs are typically broadcast continuously in a carousel format during their period of validity, since there is no (21) response (not in the figure either) for their success of failure.

ROs may be broadcast well before the time of the actual key change, so that as the programme or service key changes, access to the service is continuous.

As in the interactive device case, (22) charging of a continuous subscription may be repeated as well, by the E-Commerce System.

Should the device indeed fail to receive *any* of the broadcast attempts (20), then, as a fallback procedure, the human user of the device may again contact the E-Commerce System (represented by customer care, web shop or alike) out-of-band and request an 'immediate broadcast' of the renewed RO. Thus, as one implementation option, requested 'immediate broadcasts' and scheduled, repeated broadcasts using a broadcast carousel are alternative ways of achieving the same goal. From the device and air interface points of view, the essential thing is when and how to listen for the ROs.

Again the secure DRM agent of the device at all times holds an RO with the key material for decrypting the service, the RO being encrypted either by the device specific public key, or another key securely held by the DRM agent.

B.1.5 Consumption Steps

While the purchase steps may be applicable to both file download and streaming services, here we shall concern ourselves with streaming services only. What is achieved by the present document is that a generic data stream (not tied to any particular application) can be transmitted over a broadcast channel protected in such a way, that it can be consumed (i.e. received off air and presented to the user) by only those devices, which have purchased the proper Rights Object (RO) for the service.

To understand content consumption, we have to look at the key hierarchy which is applied for service protection.

- At the highest level, we have the keys applied by the standard DRM mechanism, used to protect the ROs in such a way that only the secure DRM agent in the correct device has access to the key material contained in each RO.
- Next, we have service keys, whose lifetime is relatively long, and which are typically used to protect continuous services such as television channels.
- Optionally, we may use programme keys, whose lifetime covers only a specific part of the service, such as a particular television programme. Programme keys are required in the case where rights associated with a service and indicated in a key stream message change frequently, with each programme event.
- At the lowest level, we have traffic keys, whose lifetime is relatively short, and which are used to encrypt the actual data stream.

To buy a pay-per-view RO for a particular television programme, the device purchases an RO containing a programme key, which decrypts traffic keys, which in turn decrypt the data stream.

To buy a television channel RO (for some time duration), the device purchases an RO containing a service key, which decrypts programme or traffic keys (where the programme keys, if used, decrypt the traffic keys), while the traffic keys decrypt the data stream. Let us look into this latter case in more detail, in case of one service. Our key hierarchy is then applied in the following fashion:

- The purchased RO contains a service key (which only the secure DRM agent of the device can reveal from the RO). As a consequence of the purchase steps, the DRM agent already holds the RO.
- Programme/traffic keys are broadcast as a separate key stream, encrypted by the service key. This is similar to the broadcast of OMA DRM 2.0 DCFs in the sense that by using the RO, the DRM agent can decrypt the programme/traffic keys from the key stream messages (but the key stream message format is different from DCF format).
- The data stream is then broadcast encrypted with the traffic keys. (If programme keys are present in the key stream, they decrypt the traffic keys.)

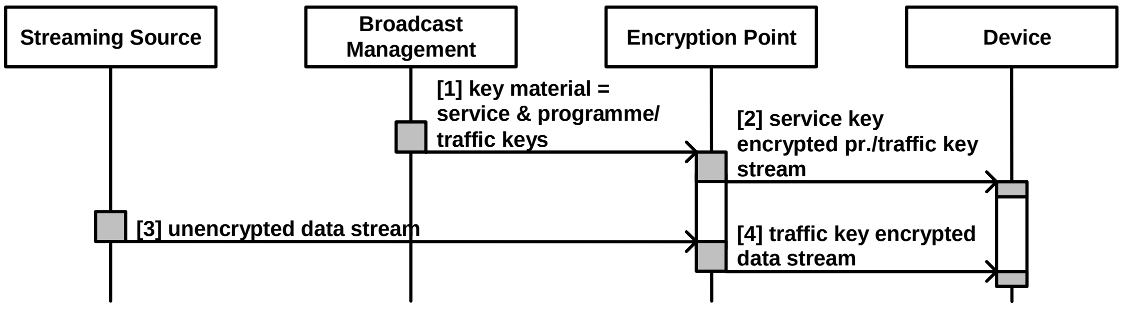

Streaming Source
Broadcast Management
Encryption Point
Device
[1] key material = service & programme/ traffic keys
[2] service key encrypted pr./traffic key stream
[3] unencrypted data stream
[4] traffic key encrypted data stream

Figure B.7: Consumption steps from the broadcaster point of view

Figure B.7 illustrates the broadcast steps. Broadcast management (1) provides the key material consisting of service and programme/traffic keys to an encryption point, (an abstract network function) which may reside at alternative places in the network. As the programme/traffic keys are to be used for data stream encryption, the encryption point (2) broadcasts the programme/traffic keys, encrypted with the service key, to devices as key stream messages. The (3) unencrypted data stream is then (4) broadcast to devices encrypted with the traffic keys.

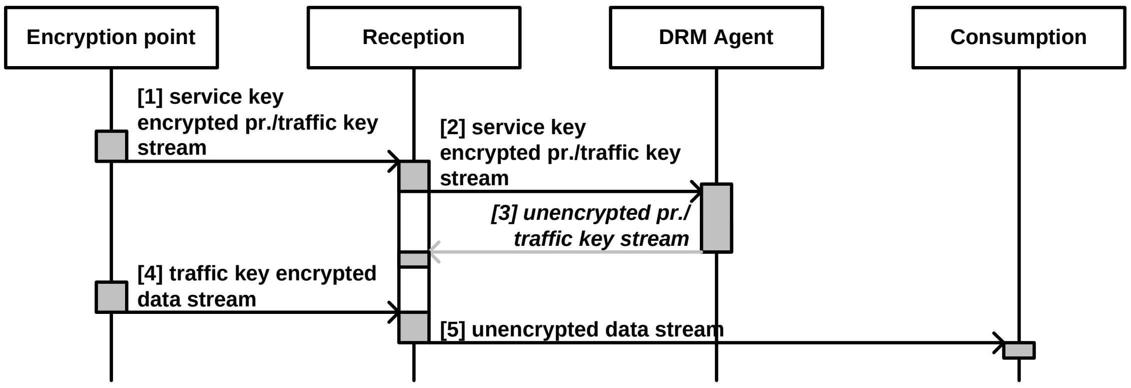

Encryption point
Reception
DRM Agent
Consumption
[1] service key encrypted pr./traffic key stream
[2] service key encrypted pr./traffic key stream
[3] unencrypted pr./ traffic key stream
[4] traffic key encrypted data stream
[5] unencrypted data stream

Figure B.8: Consumption steps from the device point of view

Figure B.8 then illustrates consumption steps at the device, in a generic way.

The (1) key stream messages are received at some reception function in the device and (2) passed to the DRM agent. The DRM agent then internally reveals the service key from the RO it holds; decrypts the programme/traffic keys from the key stream messages and (3) returns the traffic keys in plaintext to the reception function.

The reception function then (4) receives the encrypted data stream, decrypts it with the traffic keys and (5) passes the plaintext data stream to the consumption function of the device, such as a television display screen. (Yet the data stream may be anything, not just television programmes - the service protection solution specified herein is independent of the data stream format and purpose.)

Various realizations of the abstract figure B.8 are possible, and actually the data stream decryption can reside at two alternative protocol levels, both of which have to be supported by devices:

- Data stream decryption can be applied at the IPsec level, which naturally lends itself to an implementation where the reception process (with data stream decryption) is part of the IP stack of the device. In this case the application may be completely independent of the solution: it need not know anything about the IPDC service protection.
- Alternatively, data stream decryption can be applied using SRTP or ISMACryp, which lends itself to an implementation where the reception process (with data stream decryption) is integrated with the application itself. A choice between IPsec, SRTP and ISMACryp is determined by various factors; application independence versus integration being only one of them.

B.1.6 Service Protection vs. Content Protection

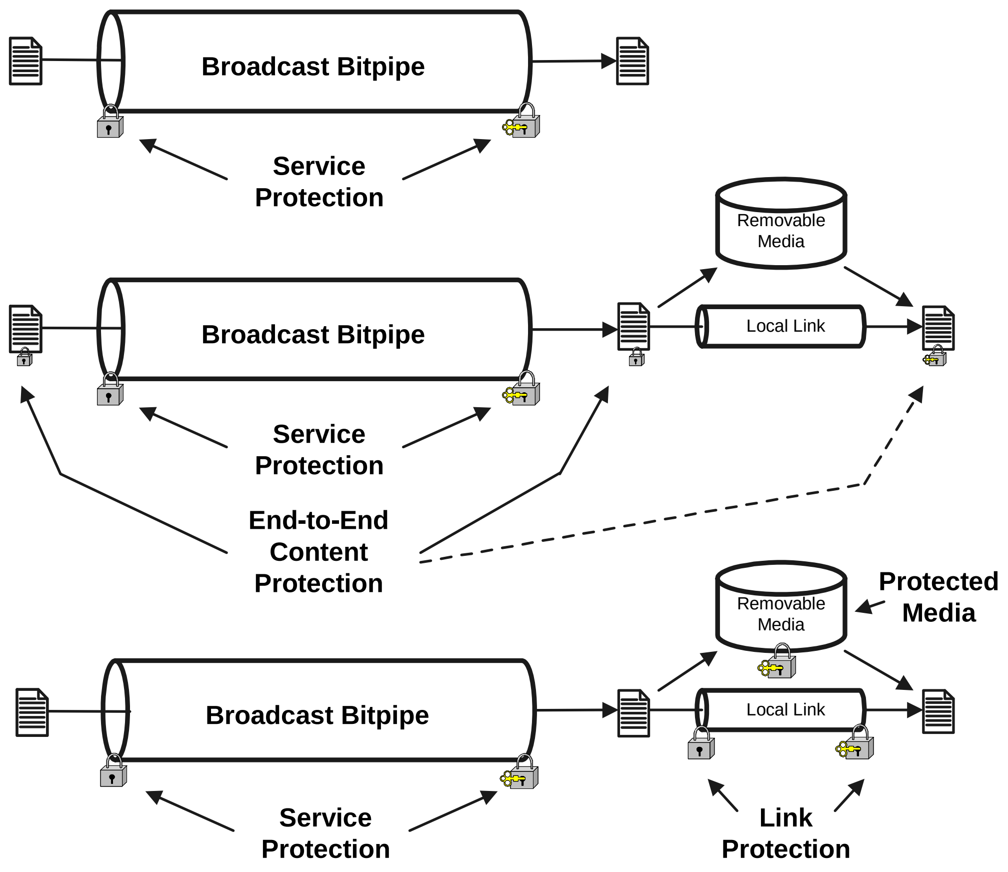


Figure B.9: Service protection vs. content protection

Figure B.9 illustrates the conceptual difference between service protection and content protection.

Service protection ensures that only those authorized to access the broadcast service can do so. Service protection is removed at reception time.

Content protection refers to protecting the content either throughout the content lifecycle (end-to-end content protection) or subsequent to delivery through the service protection system (post-delivery content protection).

The IPDC Services Purchase and Protection system defined in the present document provides service protection, and it can optionally also be used to facilitate end-to-end content protection. The access permission in Rights Objects controls access to the broadcast service and allows immediate rendering of the content. In order to be able to play recorded content, the play permission is required. If only service protection is needed, the export permission can be used to allow exporting in the clear.

In some cases, for instance to facilitate sharing of the protected content with different devices, it could be desirable to export the content to a separate post-delivery content protection system, such as DVB-CPCM. The Usage State Information needed by such a content protection system can be carried as a constraint for the export permission.

B.2 Theory of operation

B.2.1 End-to-end architecture

The figures in this clause can be used as a reference and overview when reading the present document. It contains a network-centric end-to-end architecture and a figure that shows how the Public Key Infrastructure relates to the end-to-end architecture.

The end-to-end architecture maps the elements that the present document refers to onto the entities defined in the tentative IPDC in DVB-H architecture.

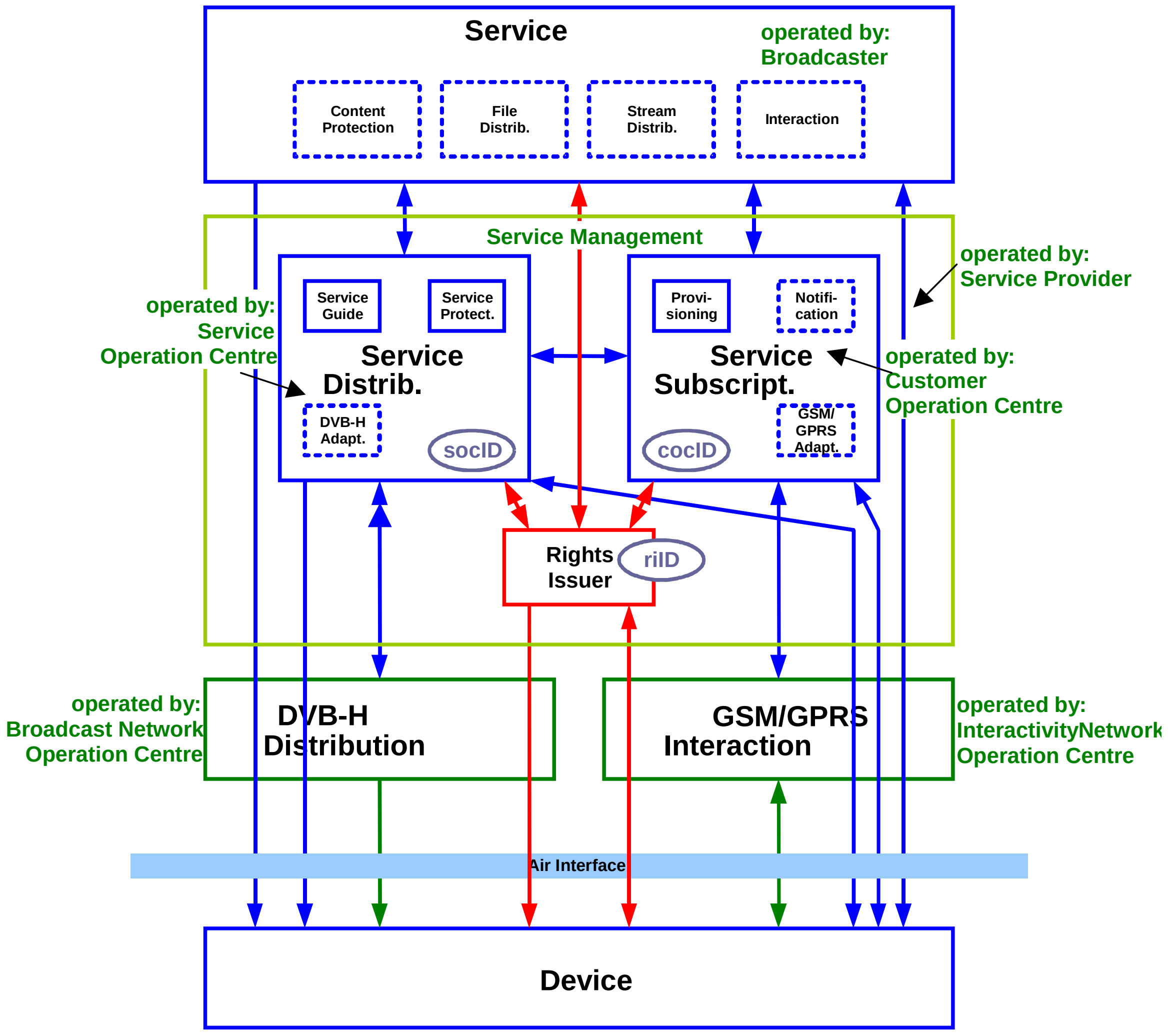


Figure B.10: Service Protection and Purchase Entities and names (Broadcast Architecture)

Figure B.10 above names several entities used in the present document. It also maps the entities used in the present document to the related architectural entities in the tentative IPDC in DVB-H architecture. The arrows in this figure represent information flow between the elements. All these entities should be considered as logical entities, i.e. actual deployment may be different and separate entities here may well be merged into larger systems. One should also note that there are some missing connections and entities, such as the off line communication of broadcast mode devices and Certification Authority needed for Public Key Infrastructure.

The service application represents any content that can be sent over a broadcast channel. Besides digital television it can be, for example, discrete files, streams or interactive services. The Service Distribution Management's role is to create and manage protected broadcasts as described in the present document. Service Subscription Management takes care of the purchase of services. The Rights Issuer takes care of device registration and rights issuing.

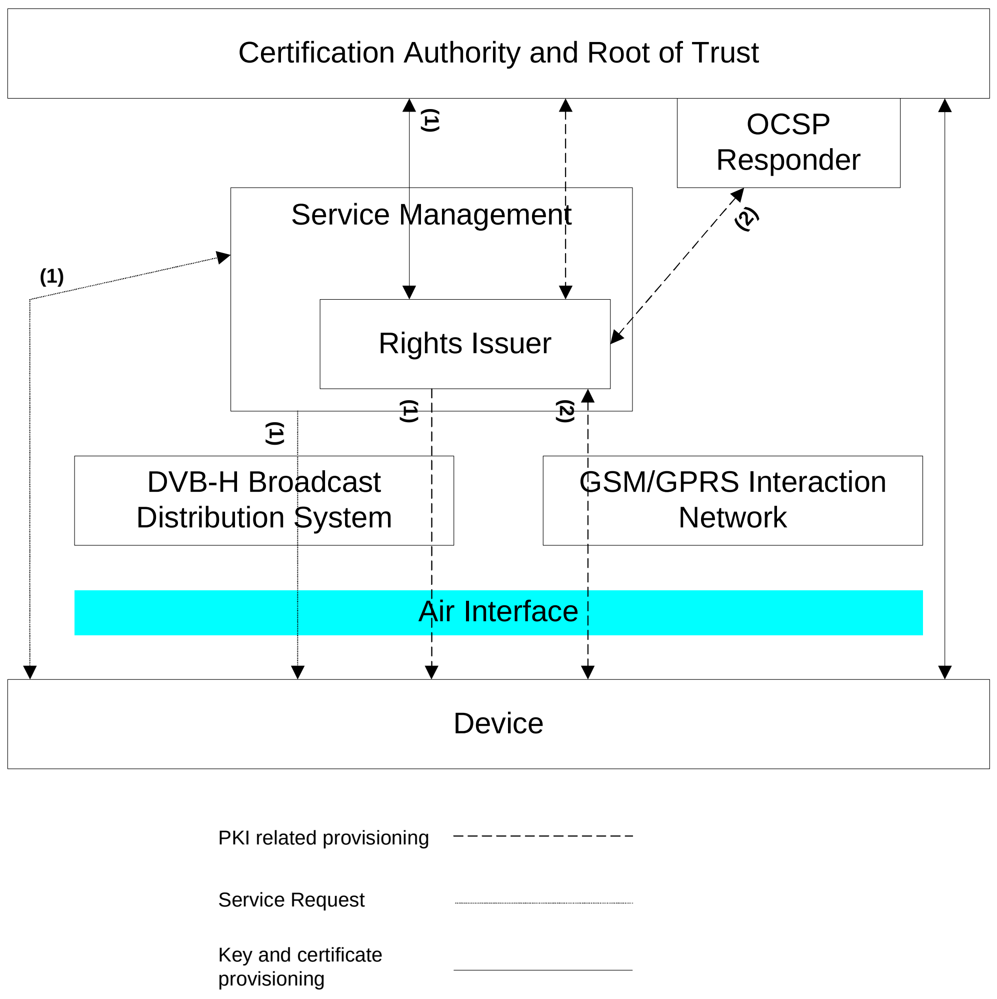

Certification Authority and Root of Trust
OCSP Responder
Service Management
Rights Issuer
DVB-H Broadcast Distribution System
GSM/GPRS Interaction Network
Air Interface
Device
(1)
(2)
PKI related provisioning
Service Request
Key and certificate provisioning

Figure B.11: Public Key Infrastructure

Figure B.11 above maps entities from the Public Key Infrastructure to the broadcast architecture. A Root Certification Authority (CA) provides keys and certificates to devices and Rights Issuers. The device and Rights Issuer SHALL have the same Root CA, otherwise the Rights Issuer cannot register the device or issue rights to the device.

Lines marked with (1) relate to broadcast mode of operation and lines marked with (2) relate to interactive mode of operation (please compare figure B.11 with figures B.13 and B.14).

B.2.1.1 Void

B.2.1.2 Special cases

B.2.1.2.1 Free-To-Air Services

Free-To-Air services are broadcast without any service protection. Any device can receive these services.

- Free-To-Air services SHALL NOT be broadcast within IPsec, SRTP or ISMACryp. In this case, a key stream will not normally be broadcast, but if one is broadcast, it SHALL be ignored by the device.

Otherwise, Free-To-Air services are beyond the scope of the present document.

B.2.1.2.2 Free-To-View Services

Free-To-View services are broadcast with service protection, but no charge is made to receive the services. However, reception can be restricted to certain users.

It is expected that Free-To-View services will be implemented by either:

- Requiring users to 'subscribe' to the service in the normal way, although no charge is made for the subscription. Rights Objects are delivered to 'subscribers'.
- Broadcasting Rights Objects to all devices in a population or to particular pre-existing Subscriber Groups or registered devices without requiring a specific 'subscription'.

However, Free-To-View services MAY be implemented in any way that is compliant with the present document.

B.2.2 Electronic Service Guide and Purchase

The Electronic Service Guide (ESG) is available to all DVB-H devices. It is a source of service and programme information. Service discovery and purchase of services is based on information transmitted in the ESG. For each service and programme, the ESG contains all the necessary information for making a purchase and for the device to find services and programmes.

Figure B.12 illustrates a high-level overview of an ESG and purchase system. Line (1) in figure B.12 represents the broadcast of an ESG over DVB-H. Lines (4) and (5) represent purchase and lines (2) and (3) represent the delivery of necessary Rights Objects.

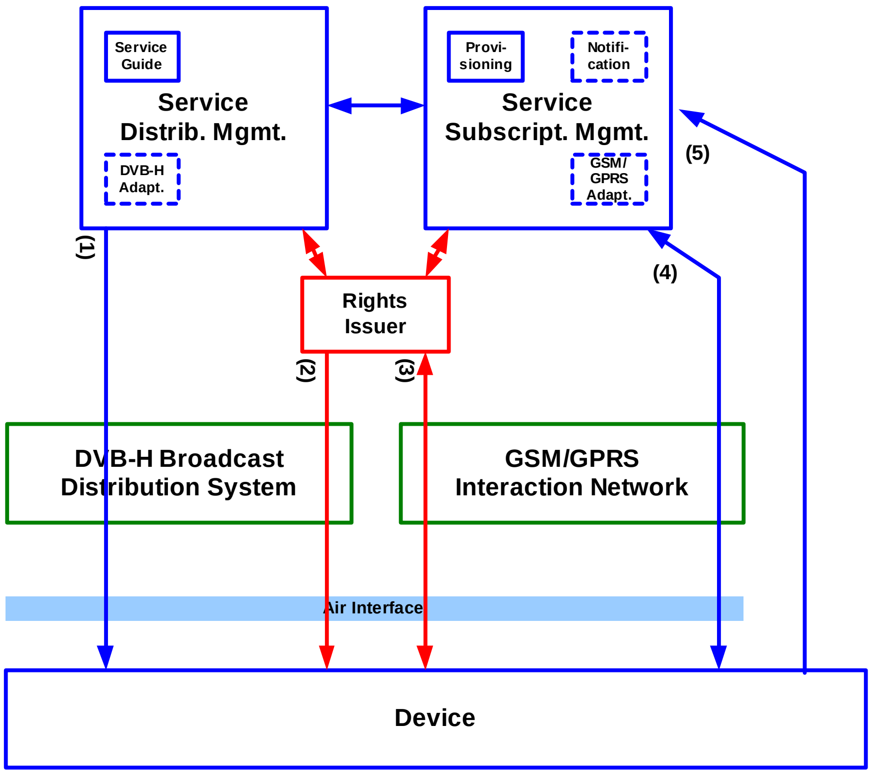


Figure B.12: Overview of ESG and Purchase

ESG data is carried over the broadcast channel. To be able to use the protected services described in the present document, a device needs to be able to receive and parse ESG data. Unless the service is free-to-air (see clause B.2.1.2.1) the user needs to purchase the service. The ESG contains all the needed information to make the purchase. Interactive and mixed-mode devices can find the URI and other necessary parameters from the ESG. For broadcast mode devices, the ESG contains a ServiceID (see clause C.3.3.2) that identifies a purchased service. Line (4) in figure B.12 represents purchase by interactive and mixed-mode devices over the interactivity channel and line (5) represents off line purchase by broadcast devices.

With interactive and mixed-mode devices, purchase initiates the sending of an ICRO over the interactivity channel; line (3) in figure B.12. With broadcast and mixed-mode devices, BCROs are sent over the broadcast channel. Before purchases can happen, devices need to be registered with the relevant Rights Issuer, see clause B.2.3.

B.2.3 Registration

B.2.3.1 Concept of the RI context

In order to communicate with a Rights Issuer (a.k.a. RI) and obtain rights objects from the RI, the device needs to have an RI context for that Right Issuer. To obtain an RI context the device notifies its device data to the RI. The RI will then contact a Root Of Trust (a.k.a. ROT) to request the certificate and capabilities matching this device data by checking the data against the Public Key Infrastructure (a.k.a. PKI) and a Certificate Revocation List (a.k.a. CRL) from that Root Of Trust. If the ROT certifies that the notified device data is valid, the RI can decide to send registration data to the device. The device will create an RI context from that registration data.

In the case that the device supports an interactivity channel which can be used to contact an RI, the device is called an 'interactive device'. In case the device does not have a direct interactivity channel to contact the RI, the device is called a 'broadcast device'.

The sending of registration data to the device is handled by a registration operation. There are several different types of registration operations, which are defined as:

Table B.1: Registration types

Registration type	Device types	clause
register device for interactive mode of operation.	Interactive	B.2.3.2
register device for broadcast mode of operation. (content and RO via broadcast channel)	Broadcast and Mixed-mode	B.2.3.3
register device for mixed-mode of operation. (both interactive and broadcast)	Mixed-mode	B.2.3.4

On successful execution the registration operations result in the creation of an RI context in the device. An RI context for broadcast mode of operation will differ from an RI context for interactive mode of operation.

B.2.3.2 Registration for interactive mode of operation

Registration for interactive mode of operation when OMA DRM 2.0 is used on Registration Layer is according to [49], using the standard 4-Pass ROAP.

This encompasses devices with interactivity channel, as well as unconnected devices, which have the capability to make a connection to an interactive device, as specified in [49] section 14, via the OBEX protocol and use the interactive device to report the device data to the RI.

B.2.3.3 Registration for broadcast (only) mode of operation

To register the device data has to be notified to the RI. There are two cases for the notification of device data to the RI:

Case 1: The device has never been registered before and is activated by the user.

There are two possibilities in which the device has no direct communication back channel to contact the RI but needs to report device data to the RI:

- The device has no interactivity channel or the interactivity channel is not able to make a connection to the RI, but the device is able to create another connection to a connected OMA device. This device is called an unconnected interactive/unconnected mixed mode device, and is covered in clause B.2.3.2.
- The device has no interactivity channel and is unable to make a connection to an interactive device. This device is called a broadcast (only) device. In this case the 1-pass binary push registered device protocol is used, as is specified in the present document.

Case 2: The device has been registered at the RI before and needs to be re-registered.

- In this case the RI uses the 1-pass binary inform registered device protocol to send a message ordering the device to re-register, as is specified in the present document.

Following sequence chart explains the registration for broadcast only mode of operation.

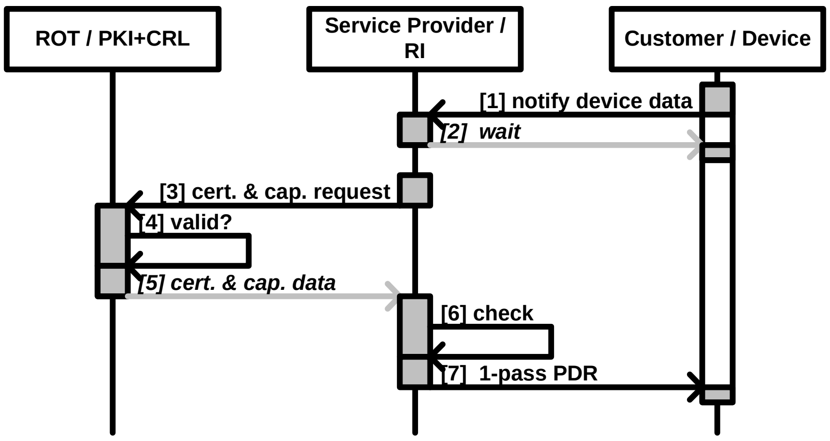


Figure B.13: Registration for broadcast mode of operation with one ROT

NOTE: Notification of device data to the Rights Issuer is performed off-line. Transmission of the registration data from the RI to the device is performed on-line via the broadcast channel.

Explanation of the protocol:

- Once the RI has the device data from the device (1) via the protocol described in clause B.3.4.3.2, the RI contacts the ROT (3), while the device is entered into registration mode and awaits the registration data (2).
- The ROT implements a Public Key Infrastructure (a.k.a. PKI). The PKI looks up the certificate and capabilities belonging to the device data in question (4). The ROT should have a Certificate Revocation List (a.k.a. CRL). In any case it is the responsibility of the ROT to decide whether the requested device data is valid or not and whether or not the requested certificate and capabilities data can be passed to the RI.
- Assuming the RI received the requested certificate and capabilities from the ROT (5), the RI will perform some last checks (6) and SHALL send back a registration data message to the device (7).
- The RI uses the 1-pass binary Push Device Registration data (a.k.a. PDR) protocol to send the registration data over the broadcast network. The PDR protocol is described in clause B.3.4.3.4. The registration data (in the format of the device_registration_response() message) is specified in clause B.3.4.3.7.2. The RI MAY decide to send an error status with the message or send valid registration data containing the data required to create an RI context.
- A device listening for device_registration_response() messages will look for messages with the corresponding message_tag. On every message with a matching message_tag the device will check the long_form_udn parameter. If this matches (any of) the device's local UDN(s), the device will process the message and will start trying to decrypt the secret data in it.
- If the device does not receive registration data within a timeout, the device leaves the registration mode and stops listening for device_registration_response() messages.

Subsequent distribution of Right Objects at regular intervals is done with a message send as an inform message using the 1-pass Inform Registered Device protocol.

B.2.3.4 Mixed-mode registration for interactive and broadcast modes of operation

This clause applies for devices supporting both communication via a broadcast channel and an interactivity channel. If such devices are registered for both interactive and broadcast mode of operation, the RI has the option of sending ROs either over the Broadcast Channel or the Interactivity Channel, whichever the RI finds better at the time of sending the ROs.

Since the registration of a broadcast only device involves more user interaction than the registration for an interactive device (see clause B.2.3.3), the registration of a mixed-mode device (with both support for broadcast channel and interactivity channel) SHALL start with the registration for interactive mode of operation, see figure B.14. Steps 3 to 10 in this figure are the 4-pass ROAP as specified in [49] with which the registration for interactive mode of operation is done when OMA DRM 2.0 is used on Registration Layer. In the figure, the registration for interactive mode of operation (steps 3 to 10) was triggered by the reception of a ROAP trigger (step 2), which was sent by the SoC when it received a purchase request (step 1) from a device that was not registered yet

After successful registration for interactive mode of operation, the device MAY also join a domain (steps 11 and 12), if so desired.

After step 12, a device that is capable of broadcast operation and uses OMA DRM 2.0 for registration SHALL put itself into registration mode, in which it waits for the registration data for broadcast mode of operation in the form of the device_registration_response() message (step 9 in the figure). If the device does not receive the registration data within a timeout the device leaves registration mode and stops listening for device_registration_response() messages.

As part of step 1, when OMA DRM 2.0 is used to initiate the registration, the RI obtains the capabilities of the device that wanted to register in the form of the signed XML data of the purchase request, see clause B.5.1.3.5. From this capabilities data, the RI finds out that the device is also capable of broadcast mode of operation. In such case the RI MAY decide to send this device the device_registration_response() message (step 14 in the figure), which message contains the registration data for the broadcast mode of operation.

In case the device indicates capability of mixed-mode operation and when the RI wants to include the domain registration in the device registration data as well, this registration data SHALL include the longform_domain_id().

Part of the registration data for the broadcast mode of operation is the longform_udn(), see clause B.3.4.3.6, which is stored in the device. In the mixed-mode registration outlined above, the RI will obtain in step 1 the longform_udn() of the device as part of the XML purchase data. Refer to clause B.5.1.3.5 for details on the use of this XML structure.

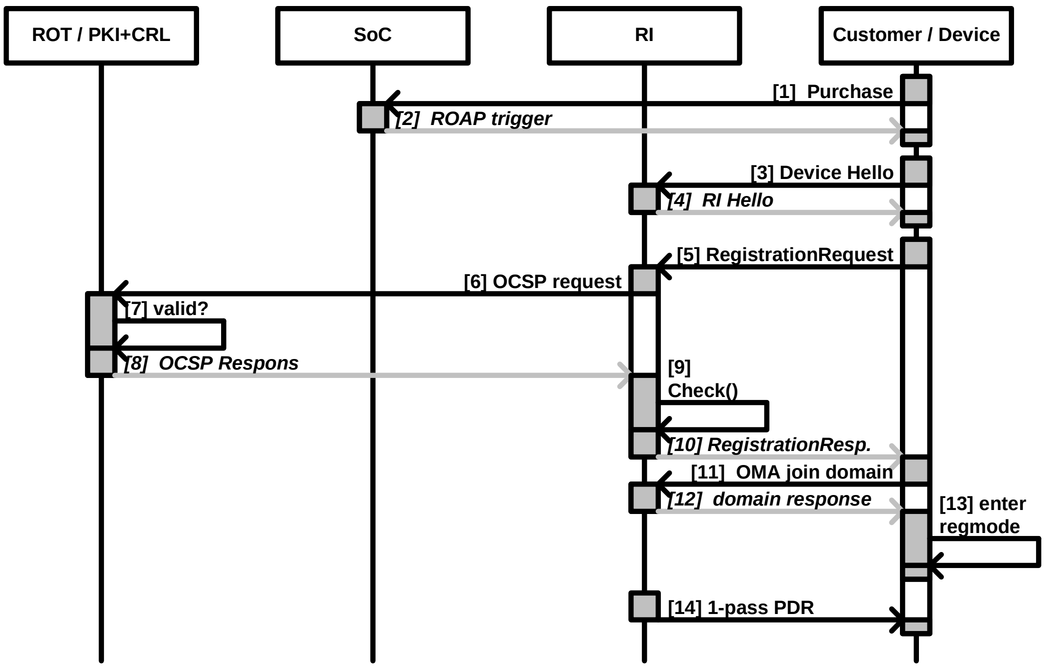


Figure B.14: Registration for mixed-mode operation with one ROT

B.2.4 The Four Layer Model

B.2.4.1 Key Hierarchy

A four-layer key hierarchy is used to implement service protection.

B.2.4.1.1 Keys on the Traffic Layer

On the lowest level, the data is encrypted using one of IPsec, SRTP or ISMACryp. This layer is called the Traffic Layer. The key used to encrypt the traffic on this layer is called Traffic Encryption Key, or TEK. The TEK changes frequently in the order of once per minute to once per second.

The encryption of the content can be written as:

$$E\{TEK\}(C)$$

with C being the data and E{TEK}() being the encryption function using the key TEK.

The data can be recovered by using the decryption function D() with the same key TEK:

$$C = D\{TEK\}\big(E\{TEK\}(C)\big)$$

As a symmetric encryption is used to encrypt the data, the data is recovered using the same key.

B.2.4.1.2 Keys on the Key Stream Layer

The TEK itself is transmitted on the key stream layer. The TEK will be encrypted using either a Programme Encryption Key (PEK) or a Service Encryption Key (SEK). The use of two different keys to protect the TEK allows for the models described in the following three clauses to be used.

B.2.4.1.2.1 Service based subscription

If the service is made available to customers by subscription only, then:

- If access rights change per programme, a programme key is used within the Key Stream Message, but is never delivered separately in a Rights Object. The scheme described in clause B.2.4.1.2.2 is used.
- If access rights do not change per programme, a programme key is not used and the scheme below is followed.

$$E\{SEK\}\big(TEK\big)$$

and

$$TEK = D\{SEK\}\big(E\{SEK\}(TEK)\big)$$

The SEK is transmitted to devices as part of the Rights Objects on the Rights Management Layer. These ROs can be normal OMA DRM 2.0 ROs in the case of an interactive device or BCROs for both mixed-mode and broadcast only devices.

Figure B.15 shows the key hierarchy for the case of a service based subscription.

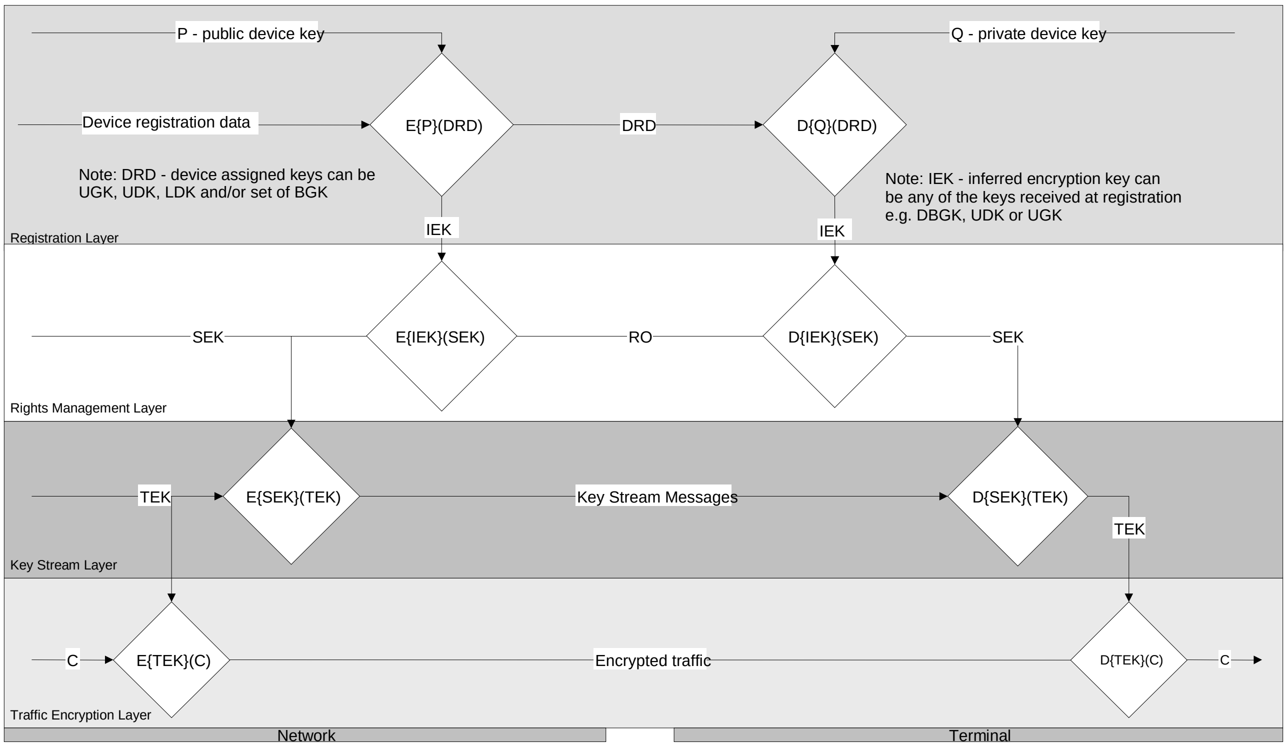


Figure B.15: 4-layer key hierarchy - use of SEK only

NOTE: DRD - device assigned keys and IEK - inferred encryption key are defined in clause B.2.4.1.5, Keys on the Registration Layer, for the broadcast mode of operation and by [49] for the interactive mode of operation.

B.2.4.1.2.2 Pay-per view based and service based subscription

If content is made available both via a service subscription and via a pay-per view based subscription then the TEK will be encrypted with the PEK:

$$E\{PEK\}(TEK)$$

Devices that do not have a service-based subscription to that service can acquire the entitlement for a specific pay-per view event. The RO for that pay-per view event will contain a PEK. This PEK can be used to decrypt the TEK:

$$TEK = D\{PEK\}(E\{PEK\}(TEK))$$

To allow devices with a service based subscription to access the service as well the PEK encrypted with the SEK is also carried in the Key Stream Message. So the KSM carries:

$$E\{SEK\}(PEK)$$

and

$$E\{PEK\}(TEK)$$

In order to decrypt the TEK given only the SEK the device has to do the following decryption

$$TEK = D\{PEK\}(E\{PEK\}(TEK))$$

with

$$PEK = D\{SEK\}(E\{SEK\}(PEK))$$

hence

$$TEK = D\{D\{SEK\}(E\{SEK\}(PEK))\}(E\{PEK\}(TEK))$$

The lifetime of a PEK is expected to last only for the duration of a specific pay-per view event while the SEK is expected to last for a longer period.

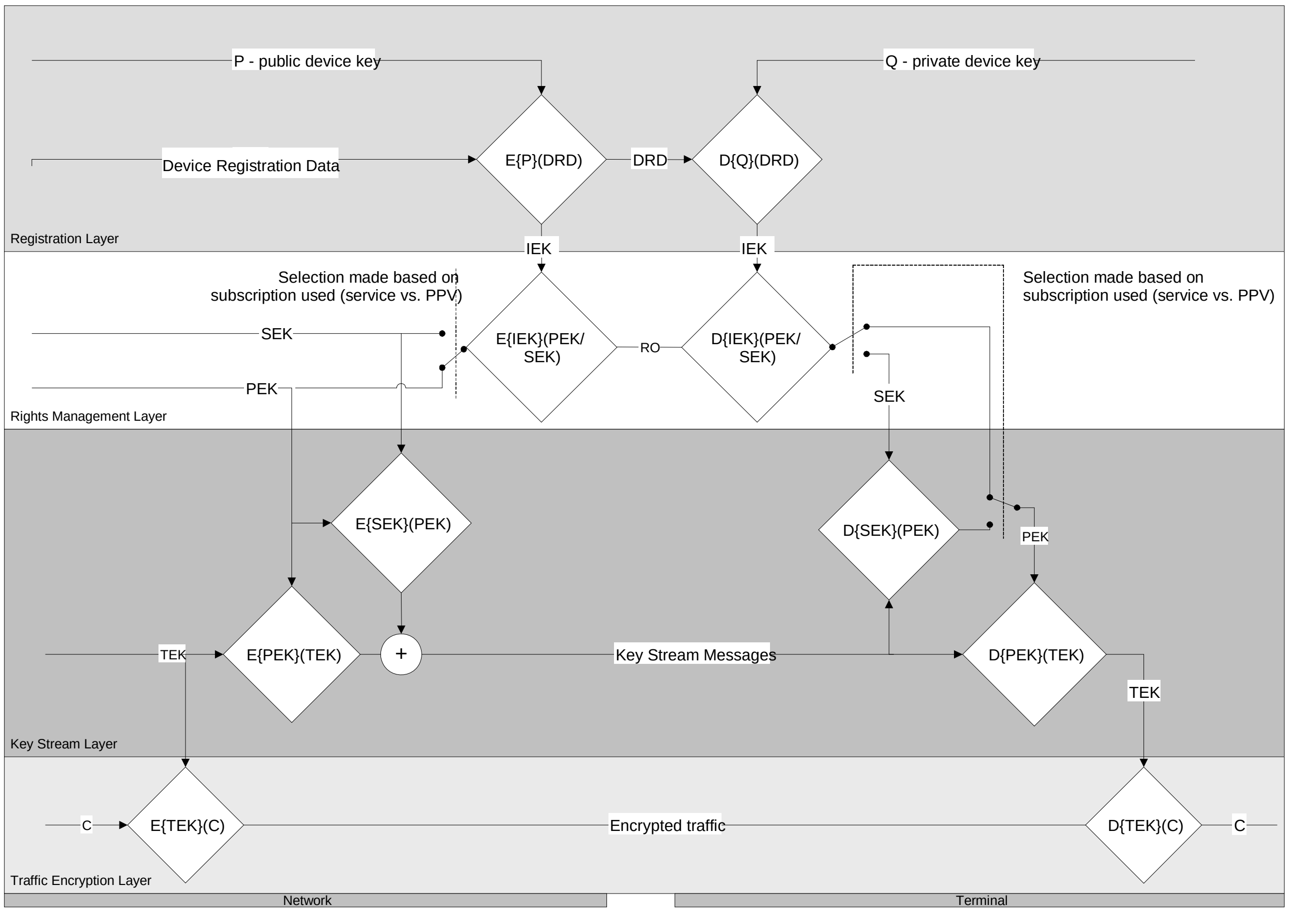


Figure B.16: 4-layer key hierarchy - use of PEK and SEK

Figure B.16 shows the four layer key hierarchy in the case of service subscription and pay-per-view.

B.2.4.1.2.3 Pay-per view based consumption

If content is consumed on a pay-per view only basis, that means that the content is not available via a service subscription, than the TEK MAY be encrypted with a PEK or the TEK MAY be encrypted with a SEK. Both PEK and SEK are interchangeable in this case.

B.2.4.1.3 Keys on the Rights Management Layer (Broadcast mode)

The SEK and PEK are transmitted to the device on the Rights Management Layer as part of a BCRO.

The keys used to encrypt and decrypt the SEK or PEK depend on the addressing mode of the BCRO (see clause B.2.6.2). The term 'Inferred Encryption Key' (IEK) is used to describe the key used to en-/decrypt the SEK/PEK without specifying which exact key is used.

- **RO addressed to a unique device:**
  In the case that an RO is addressed to a unique device, the IEK used to encrypt the SEK or PEK is the unique device key (UDK) which was delivered during device registration.

- **RO addressed to a subscriber group (subset of unique group):**
  In the case that an RO is addressed to a subset of a unique group (subscriber group), the IEK is deduced from the subscriber group keys (SGKs) by use of zero message broadcast encryption. Refer to clause B.2.6.3 for an explanation of the zero message broadcast encryption concept. Refer to clause B.13 for an explanation of how the IEK is deduced given the bit access mask and the SGKs.

- **RO addressed to a unique group:**
  In the case that an RO is addressed to all devices in a unique group, the IEK used to encrypt the SEK or PEK is the unique group key (UGK).

- **RO addressed to a domain:**
  In the case that an RO is addressed to a domain, the IEK used to encrypt the SEK or PEK is the broadcast domain key (BDK) which was delivered during device registration.

- **RO containing a CEK:**
  In the case an RO is for an OMA DRM 2.0 content format (e.g. a DCF), the asset carries a CEK object and an additional cipher value. Decryption of the key material is defined by [49].

B.2.4.1.4 Keys on the Rights Management Layer (Interactive mode)

OMA DRM V2 [49] governs how to process any keys contained within Interactivity Channel Rights Objects.

B.2.4.1.5 Keys on the Registration Layer (Broadcast mode)

The registration layer delivers the keys used for the broadcast mode of operation. There are several keys used for authentication and decryption purposes.

Table B.2: Keyset in the registration data

Name of key	Description	remark
UGK	unique_group_key	
SGK	subscriber_group_key	used for zero message broadcast
BDK	broadcast_domain_key	used for domains
UDK	unique_device_key	
RIAK	ri_authentication_key	used for authentication
UDF	unique_device_filter	Eurocrypt address, not a key
SBDF	shortform_broadcast_domain_filter	domain_id address, not a key
LBDF	longform_broadcast_domain_filter	domain_id address, not a key
TDK	token_delivery_key	not used on registration layer

The keyset itself is delivered to the device in a protected format as part of the device registration data (refer to clause B.3.4.3.7.2 for details).

The RI generates a session key (SK) to protect the keyset_block (UGK, SGK1..n, UDK, BDK, RIAK, UDF, SBDF, LBDF and/or TDK), which carries the keyset described in table B.2.

$$encrypted\_keyset\_block = E\{SK\}(keyset\_block)$$

The RI encrypts the SK and the encrypted_keyset_block (together called the SK+encrypted_keyset_block) into a sessionkey_block, such that:

$$sessionkey\_block = E\{DP\}(SK + encrypted\_keyset\_block)$$

where the sessionkey_block is encrypted with the public key of the device (DP).

NOTE: If the keyset_block would not fit into the size of the sessionkey_block the remainder is kept as surplus_block. Refer to clause B.3.4.3.7.2.2 for details.

The complete message (header, sessionkey_block and optional surplus_block) is protected by a single source authenticity check, such that:

$$signature\_block = A\{RIQ\}(message)$$

where the RIQ is the private key of the RI.

Upon reception the device follows the rules described above in reverse order:

$$V\{RIP\}(signature\_block)$$

$$SK + encrypted\_keyset\_block = D\{DQ\}(sessionkey\_block)$$

$$keyset\_block = D\{SK\}(encypted\_keyset\_block)$$

where:

The signature_block is verified with the RI public key (RIP).

The encrypted sessionkey_block contains the session key (SK) plus encrypted_keyset_block (together called the SK+encryped_keyset_block) and is decrypted with the device's private key (DQ).

NOTE: If the surplus_block is present, it is concatenated to the keyset_block from the sessionkey_block. Refer to clause B.3.4.3.7.2.2 for details.

The encrypted_keyset_block, decrypted with the session key (SK), produces the keyset_block, containing the keyset (UGK, SGK, UDK, RIAK, UDF), which never leaves the DRM agent.

The notation HMAC_SHA1{k}(s) is used denote the computation of HMAC [23] with SHA1 [44] as the hash function keyed by the key "k" over the string "s". HMAC_SHA1_128{k}(s)is used to denote the 128 most significant bits of HMAC_SHA1{k}(s) output.

A key IEK is 'derived' from the UGK, SGK, UDK or BDK to decrypt the BCRO, such that

$$IEK = HMAC\_SHA1\_128\{UGK\}(salt)$$

or

$$IEK = HMAC\_SHA1\_128\{NK_i \parallel ... \parallel NK_j\}(salt)$$

where the NKs are NK keys ordered according to the index (such that i < j) that are required for creating the key for the desired group. The keys NK are obtained using the scheme described in clause B.2.6.3. See clauses B.2.6.3 and B.13.

or

$$IEK = HMAC\_SHA1\_128\{UDK\}(salt)$$

or

$$IEK = HMAC\_SHA1\_128\{BDK\}(salt)$$

The IEK is used to decrypt the part of the BCRO containing keys, which carries CEK or SEK and/or PEK. The "salt" parameter is the BCI value in the asset structure of the BCRO. The BCI value from the first asset structure in a BCRO SHALL be used for all assets in a BCRO structure.

B.2.4.1.6 Authentication overview

Following picture explains the authentication 'hierarchy' of the system.

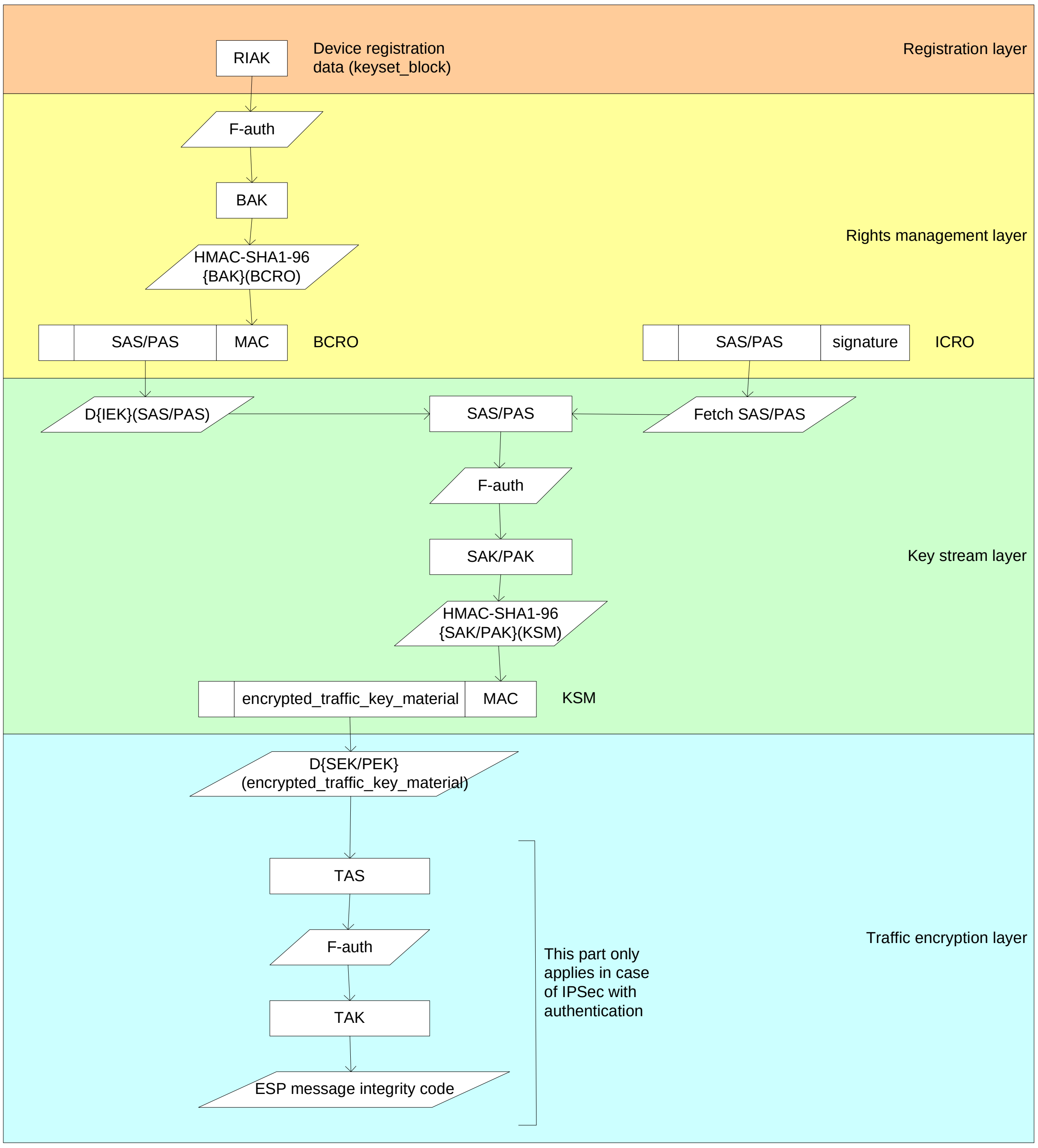

RIAK
Device registration data (keyset_block)
Registration layer
F-auth
BAK
HMAC-SHA1-96 {BAK}(BCRO)
Rights management layer
SAS/PAS
MAC
BCRO
SAS/PAS
signature
ICRO
D{IEK}(SAS/PAS)
SAS/PAS
Fetch SAS/PAS
F-auth
SAK/PAK
Key stream layer
HMAC-SHA1-96 {SAK/PAK}(KSM)
encrypted_traffic_key_material
MAC
KSM
D{SEK/PEK} (encrypted_traffic_key_material)
TAS
F-auth
TAK
ESP message integrity code
This part only applies in case of IPSec with authentication
Traffic encryption layer

Figure B.17: Authentication hierarchy

B.2.4.1.6.1 Authentication keys on traffic layer

When IPsec is used with authentication, the message SHALL be verifiable by the ESP integrity code. This SHALL be done by means of the Traffic Authentication Key (TAK) which is derived from the Traffic Authentication Seed (TAS) (see clause B.14.1).

B.2.4.1.6.2 Authentication keys on key stream layer

The KSM SHALL be authenticated and the integrity of the message SHALL be verified. This SHALL be done by means of the Program Authentication Key (PAK) and/or the service authentication key (SAK), which are derived from the Program Authentication Seed (PAS) and the Service Authentication Seed (SAS), which are delivered as part of the RO (see clause B.14.2)

B.2.4.1.6.3 Authentication keys on rights management layer (broadcast mode)

The BCRO SHALL be authenticated and the integrity of the message SHALL be verified. This SHALL be done by means of the BCRO Authentication Key (BAK), which is derived from the RI Authentication Key (RIAK), which is delivered during registration. (see clause B.14.3).

B.2.4.1.6.4 Authentication keys on registration layer (broadcast mode)

The RI Authentication Key (RIAK) is delivered during registration as part of the device_registration_response(). Refer to clause B.3.4.3.4 for details.

B.2.5 Deployment for interactive mode of operation

B.2.5.1 Concept of Domains - OMA DRM 2.0 Domains

Content in OMA DRM 2.0 can be bound to a device or to a domain, see [49].

A domain in OMA DRM 2.0 is a group of one or more devices which share a common secret, the so called Domain Key. In the case of content that is bound to a domain, the Content Encryption Key for that content (as stored in the RO associated with that content) is encrypted with the Domain Key of that domain. A device which receives an RO that is bound to a domain of which it is a member can freely pass that RO to other devices belonging to the same domain. The other devices in the same domain can then access that content as allowed by the RO.

OMA DRM 2.0 defines protocols with which a device can join and leave a domain: the ROAP Join Domain Protocol and the ROAP Leave Domain Protocol. In order to use domain keys the device needs to have joined the corresponding domain.

In the present document, an OMA DRM 2.0 domain as specified by [49] is referred to as an interactive domain.

B.2.6 Deployment for broadcast mode of operation

B.2.6.1 Concept of Domains - broadcast domains

Content in OMA DRM 2.0 can be bound to a device or to a domain, see [49]. In the present document, we refer to a domain as specified by [49] as an interactive domain.

A domain in [49] is a group of one or more devices which share a common secret, the so called Domain Key. In the case of content that is bound to a domain, the Content Encryption Key of that content as stored in the RO associated with that content is encrypted with the Domain Key of that domain. A device which receives an RO that is bound to a domain of which it is a member can freely pass that RO to other devices belonging to the same domain. The other devices in the same domain can then access that content as allowed by the RO.

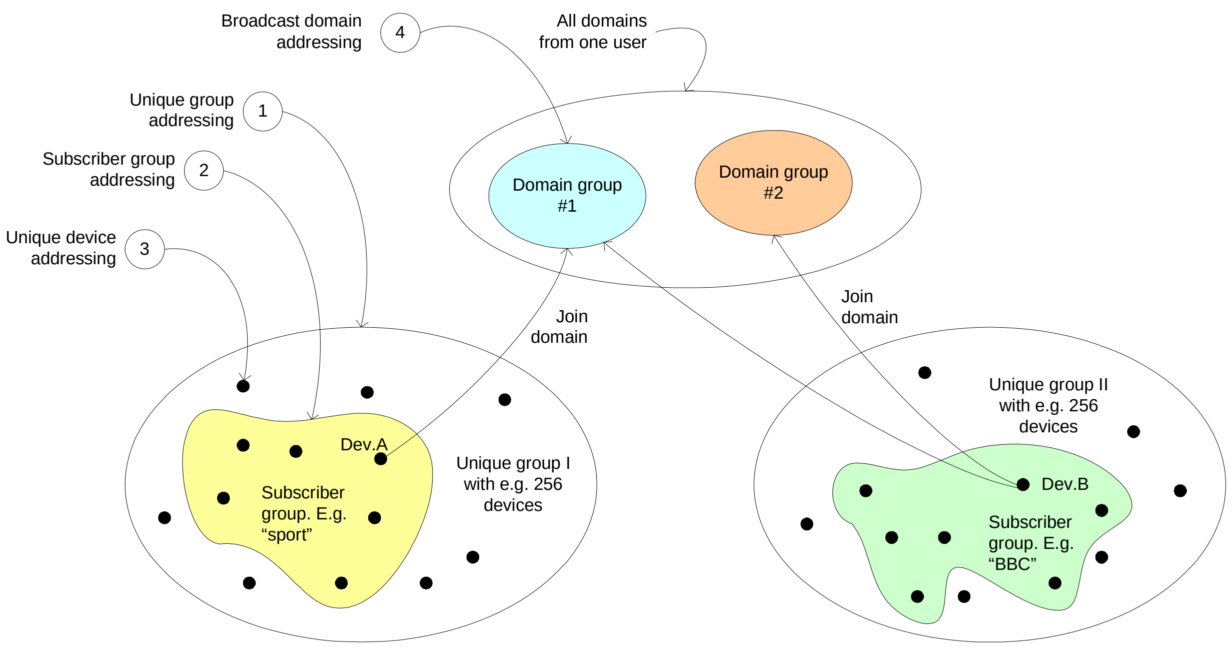
[49] defines protocols with which a device can join and leave a domain: the ROAP Join Domain Protocol and the ROAP Leave Domain Protocol. In order to use domain keys the device has to have joined the corresponding domain. The equivalent for these protocols in the case that there is no interactivity channel are defined in clause B.3.4.4.

The equivalent of the interactive domain in case there is no interactivity channel is the broadcast domain. In the BCRO, there is a facility to indicate that the BCRO is intended for a broadcast domain, using the address_mode. If the address_mode is set to 'domain', the domain key is used for the encryption of the key material in the BCRO. Furthermore, if the address_mode is set to 'domain', the domain_id is created by concatenating the address_field with the domain_id_extension.

Please note that while the domain ID and domain key MAY be the same in the case of the interactive and broadcast domain, the content for broadcast domains with BCROs is not interoperable with content for interactive domains with ICROs.

B.2.6.2 Addressing (group / subset / device / domain)

The registration data supports four methods of addressing devices, as is explained in figure B.18.


Broadcast domain addressing
4
All domains from one user
Unique group addressing
1
Subscriber group addressing
2
Unique device addressing
3
Domain group #1
Domain group #2
Join domain
Join domain
Dev.A
Subscriber group. E.g. "sport"
Unique group I with e.g. 256 devices
Unique group II with e.g. 256 devices
Dev.B
Subscriber group. E.g. "BBC"

key:
1 = addressing a unique group - use Unique Group Key (UGK).
2 = addressing < unique group - use Subscriber Group Key(s) (SGK).
3 = addressing only one device - use Unique Device Key (UDK).
4 = addressing a broadcast domain - use Broadcast Domain Key (BDK).

Figure B.18: Explaining the concept of addressing

A unique group contains all the devices in a group. A subscriber group can be smaller than, or as large as, the unique group. Alternatively a device can be addressed via a (broadcast) domain group. A unique device can be part of a unique group and/or a subscriber group and/or a broadcast domain. One or more unique groups form the population of devices. In this example the group size is 256 devices. Group sizes of 256 and 512 are supported as these are acceptable group sizes when it should be needed to revoke devices. Using a larger group size is not supported because of the fact that eventual revocation of such a group easily concerns too many devices.

The following clauses describe the relationship between the registration data and the Broadcast Rights Object. The registration data is sent to the device after successful registration of the device. At a later stage the device can receive a BCRO as a means to obtain the content encryption key, which in turn is used to decrypt the encrypted AV content. The content key may carry the SEK and/or PEK. The content key is encrypted with an Inferred Encryption Key (IEK) by the RI. The following clauses describe the different addressing modes, how the message is filtered and what type of IEK will be used by the device to obtain the content key.

A device supporting broadcast mode of operation SHALL support all four addressing modes. The network operator can choose to send registration data with a keyset of UGK and/or SGK and/or UDK and/or BDK to the device at registration time.

NOTE: Refer to clause B.2.4.1 for the general overview of the process used to construct the IEK.

B.2.6.2.1 Addressing the unique group

To access the whole group, following the (oversimplified) BCRO is used.

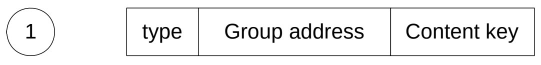


Figure B.19: (Oversimplified) group BCRO

- The group address was delivered with the registration data and is used to filter for the message (refer to clause B.3.4.3.7 for details).

- The content key (which can carry the SEK and/or PEK) is encrypted with an Inferred Entitlement Key (IEK). In this case the unique_group_key (UGK) is used as the IEK. The UGK used by the RI is identical to the key that was delivered with the registration data sent to the device (refer to clause B.3.4.3.7 for details).
- All the devices in the group can use the content key.

B.2.6.2.2 Addressing a subscriber group

The subscriber group is a privileged subset of the unique group. Two methods are available to address the subscriber group, but either one can be used:

- At registration, the registration data delivers a set of subscriber_group_keys (SGK) to the device. By using zero message broadcast encryption (refer to clause B.2.6.3) an Inferred Encryption Key (IEK) can be constructed from the SGKs sent to the device. The RI uses the this IEK to encrypt the content key and only the devices with the matching set of SGKs on board can construct the same IEK.
- For reasons of completeness it is mentioned that is also possible to deliver no SGKs with the registration data. With 0 (zero) SGKs, it is of course not possible to deduce an IEK. In this case, unique device addressing with a unique_device_key (UDK) as the IEK is used. Please refer to clause B.2.6.2.3 for more details.

NOTE: It is inefficient for large populations to use unique addressing instead of (broadcast) group addressing, since it quickly consumes a considerable amount of bandwidth.

To access the subscriber group, the following (oversimplified) BCRO is used.

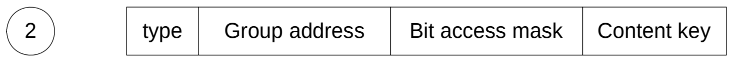


Figure B.20: (Oversimplified) subscriber group BCRO

For both methods, the group address was delivered with the registration data and is used to filter for the message.

- The group address is part of the unique_device_filter (UDF) address that was sent with the registration data to the device and is used to filter for the message (refer to clause B.3.4.3.7 for details).
- The Eurocrypt-style (see [16]) bit access mask prescribes which group members are 'entitled' to use the content key (which can carry the SEK and/or PEK). This bit mask indicates which group members of the subscriber group can deduce the broadcast key from the zero message broadcast keys in the device.
- The device can construct the Inferred Encryption Key (IEK) from the SGKs that were delivered to the device. The SGKs used in this process belong to the same RI context as the BCRO.
- Only members of the subscriber group can use the content key. If a member of the group succeeds in constructing the IEK, that member can decrypt the Content key. Any group member that tries to deduce the IEK but does not have the appropriate (zero message) SGK on board is unable to deduce the IEK to decrypt the content key.

B.2.6.2.3 Addressing a unique device

To access a unique device, the following (oversimplified) BCRO is used.

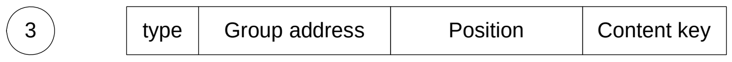


Figure B.21: (Oversimplified) unique device BCRO

A unique device is a privileged set of the total group.

- A unique device is addressed by a unique address, which is a group address plus a position / offset in the group. This address matches the unique_device_filter address that was sent with the registration data to the device (refer to clause B.3.4.3.7 for details).

- The content key (which can carry the SEK and/or PEK) is encrypted with an Inferred Encryption Key (IEK). In this case the Unique_Device_Key (UDK) is used as the IEK. The UDK used by the RI is identical to the key that was delivered with the registration data sent to the device (refer to clause B.3.4.3.7 for details).
- Only the unique device which is addressed can use the content key.

NOTE: In all cases, the head end infrastructure composes the IEK used to encrypt the content key and determines the access mask on the basis of the created key. For the subscriber group case the head end infrastructure creates the access mask based on the corresponding (zero message) SGKs. A 'type' field inside the BCRO SHALL indicate which addressing case is covered.

B.2.6.2.4 Addressing a broadcast domain

To access a broadcast domain, the following (oversimplified) BCRO is used.

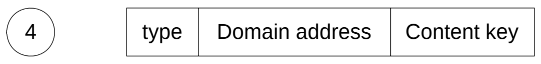


Figure B.22: (Oversimplified) broadcast domain BCRO

A broadcast domain is a privileged set of the total group.

- The domain address was delivered with the registration data (refer to clause B.3.4.3.7 for details) and/or via a join domain response (refer to clause B.3.4.4.4.1.1 for details) and is used to filter for the message. This address is the OMA DRM 2.0 domain ID.
- The content key (which can carry the SEK and/or PEK) is encrypted with an Inferred Encryption Key (IEK). In this case the broadcast_domain_key (BDK) is used as the IEK. The BDK used by the RI is identical to the key that was delivered with the registration data sent to the device (refer to clause B.3.4.3.7 for details).
- All the devices in a broadcast domain group can use the content key.

NOTE: Broadcast Domain addressing is included in the specification as an additional addressing option, that will potentially save some bandwidth over unique device addressing, because more devices belonging to one user might be registered into the same broadcast domain group. The "savings" in bandwidth with domain addressing are not as high as under subscriber group addressing. For completeness it is mentioned that the domain addressing can also be used in another mode: an option would be to use domain addressing with a population of millions of devices to allow large groups to access low value content. In this particular use of domains the "savings" in bandwidth might be considerable compared to any other of the mentioned addressing modes. The trade-off is that a security incident can affect more devices.

B.2.6.3 Zero Message Broadcast Encryption scheme

Zero message broadcast encryption was originally introduced in "Broadcast Encryption" by Fiat and Naor [43]. It solves the problem of creating a privileged set within a group of devices without the need for sending out messages to create such a set. During device registration the RI creates registration data for the device and is able to enter the correct subscriber_group_keys (SGK) into the registration data. After the device registration no more data needs to be transmitted to the unique devices, which is why it is called a Zero Message Broadcast (a.k.a. ZMB) scheme. This therefore preserves bandwidth on the network.

The present document uses an algorithm similar to the Fiat-Naor scheme, but with the following advantages:

- The algorithm is altered to support changing the IEK independently of the subscriber group keys.
- The IEK does not reveal information about the subscriber group keys.

Zero message broadcast encryption is based on a set of group keys that are provisioned to the terminal during registration. The number of group keys needed depends on the group size (n = log2(group size)). Each terminal needs (worst case) to derive m -1 keys (with m = group size). A device that implements zero message broadcast encryption will need, for group addressing, n+1 keys (the additional key is used when the privileged set is equal to the complete group).

Deriving the required keys is done as follows, illustrated with a group size of 8.

NOTE 1: Key material for the authentication of messages is omitted for readability reasons.

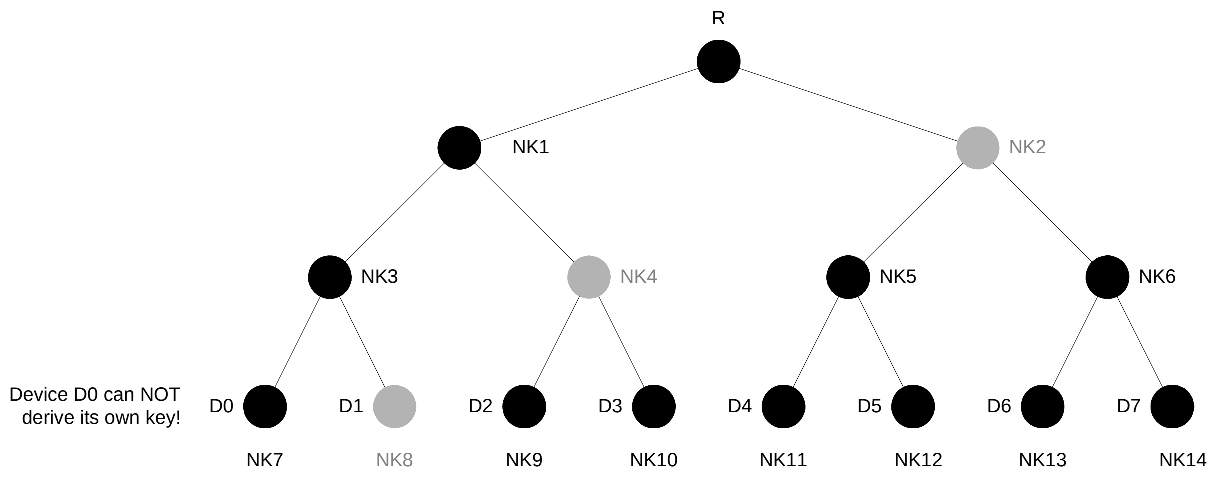


Figure B.23: Example of a zero message tree with 3 nodes (keys)

m = 8, the number of terminal keys will be log2(8) = 3

Assume the Terminal D0 has the following keys:

{NK2, NK4, NK8}

The terminal now derives, as specified in clause B.13 steps a to d, a set of leaf node keys {D4, D5, D6, D7} from NK2 and {D2, D3} from NK4. This is done using AES such that the left child keyg of key NKi is AES_K(T1+2i) and the right child key is AES_K(T2+2i). Using the mechanisms as set forth in clause B.13 the device can derive the leaves in the following order:

- From NK2 it derives NK5 and NK6.
    - From NK5 it derives NK11 and NK12.
    - From NK6 it derives NK13 and NK14.
- From NK4 it derives NK9 and NK10.

The || binary operator is used to denote concatenation. The notation HMAC_SHA1{k}(s) is used denote the computation of HMAC [23] keyed by the key "k" over the string "s" with SHA1 [NIST 180.2] as the hash function. HMAC_SHA1_128{k}(s) is used to denote the 128 most significant bits of HMAC_SHA1{k}(s) output. The IEK is constructed by computing HMAC_SHA1 keyed by the concatenation of the selected keys, with the keys ordered according to their index in the Eurocrypt [16] address. The HMAC_SHA1 is computed over a salt that is defined as follows. The IEK is specific to each encrypted SEAK or PEAK in the BCRO. The 96-bit BCI field from the BCRO asset() structure is used as the salt. The BCI value from the first asset structure in a BCRO SHALL be used for all assets in a BCRO structure in the case that there is more than one asset structure in a BCRO.

<salt> = BCI

IEK = HMAC_SHA1_128{NK8 || NK9 || NK10 || NK11 || NK12 || NK13 || NK14}(<salt>).

NOTE 2: To exclude a device from the so-called privileged set, its key is included in the decryption key. If terminal D1 and D5 are to be excluded, the rights decryption key would be constructed by computing HMAC_SHA1_128{D1 || D5}(<salt>). Given that terminal D1 and D5 are not able to derive their own keys (i.e. D1 or D5), these terminals are excluded from the so-called privileged set and cannot create the rights decryption key. Please refer to clause B.13 for an overview of the function F_{ZMB}.

NOTE 3: The best (feasible) group sizes will be 256 or 512. Because of the small size of the group the local processing requirements are limited on the device side, and the device needs limited resources for the Fiat Naor tree. The derivation process will have some computational overhead but can be implemented efficiently (e.g. it is not necessary to calculate all terminal keys before processing them into the broadcast key).

One of the reasons to keep the group size small is because the security of the Zero Message Broadcast Encryption scheme is based on the assumption that any two devices within the same group do not share or exchange their broadcast encryption keys. This means that if two members of a group hack their terminal and merge their subscriber group key set, they would be capable of a collusion attack which would enable them to decrypt any messages that are addressed to a subset of this group - even the ones for which they are not authorized. The RI would be forced to re-register the complete group (distributed over a number of groups to perform some sort of a stepwise refinement scheme to identify the culprits). However, when the group sizes are small enough (i.e. 256 or 512), collusion is not a big threat.

This scheme improves upon the current practice of relying purely on tamper-resistance, as breaking the tamper-resistance of a single device is not sufficient to obtain access to unauthorized content. A collusion attack where the SGK keys from at least two members of the same group is required. Even when the keys have been acquired these keys need to be distributed to another device, which is by no means trivial. Above that a distribution might be also be traceable.

In order to make the probability of such collusion attacks negligible, the following criteria SHOULD be followed when assigning devices to Broadcast Encryption Groups:

1) Devices owned by the same user are not assigned to the same group.

2) Each device is assigned to a group randomly.

It is the responsibility of the RI to make available and manage an appropriate number of groups. When the number of groups is in the order of thousands, collusion attacks become costly and impractical.

B.2.7 Interoperability with Alternative Implementations of the Functionality of Rights Management Layer and Registration Layer

All devices SHALL implement the rights management and registration layers as specified in clauses B.3.3 and B.3.4 of the present document. In addition, devices MAY contain alternative implementations of the functionality of these layers and Rights Issuers MAY make use of these alternative implementations, subject to the requirements below.

Alternative implementations of the functionality of the registration layer and rights management layer SHALL provide the following:

- A means to provide a collection of information called an RO from the RI to the device per service or programme that the device is entitled to consume. This collection of information consists of the following data:
  - The ID of the RI (riID).
  - The CID or BCI of the programme or service associated with the RO. If not apparent from the CID, the RO SHALL have an indication of whether the CID is associated with a programme or a service.
  - Optionally information regulating the consumption of the programme or service associated with the RO.
  - If the CID is for a programme, an RO associated with that programme SHALL contain key material for transferring the values of the PEK and PAK for that programme to the device. This key material SHALL be protected for confidentiality.
  - If the CID is for a service, an RO associated with that service SHALL contain key material for transferring the values of the SEK and SAK for that programme to the device. This key material SHALL be protected for confidentiality.
  - The combination of all of the above information SHALL be protected for integrity.
- A means to understand the permissions and constraints that are carried in the KSM, if any.
- A means to combine the permissions and constraints obtained from the KSM with the optional information in the RO for regulating consumption for making the decision to grant or deny a particular requested access to a particular programme or service.

- A means to provide from the RI to the device all information that is required for the key stream layer to decrypt the programme(s) or service(s) that the RI allows the device to access and providing this decryption information to the key stream layer for a particular programme or service if the combined permissions and constraints grant the requested form of access to this programme or service.

The device SHALL check all information it receives from the RI for integrity. In addition, it SHALL protect all cryptographic key information (e.g. all decryption and authentication keys) for confidentiality.

The specifications in this clause define an interoperability point between the key stream layer and the rights management layer. This interoperability point not only allows for horizontal competition between providers of the Service Protection System according the present document including clauses B.3.3 and B.3.4, but also with providers of arbitrary, e.g. proprietary, solutions in a standards-compatible way.

B.3 The Four-Layer Model for Service and Content Protection

B.3.1 Traffic Layer

For protection of media streams on the traffic layer, IPsec, SRTP or ISMACryp MAY be used, as defined in detail in clause 6. The use of EITHER IPsec OR SRTP is STRONGLY RECOMMENDED.

This clause explains how to use these protection methods together with the other layers defined in this annex.

B.3.1.1 IPsec

Figure B.24 shows the different objects and elements involved in instantiating IPsec security associations.

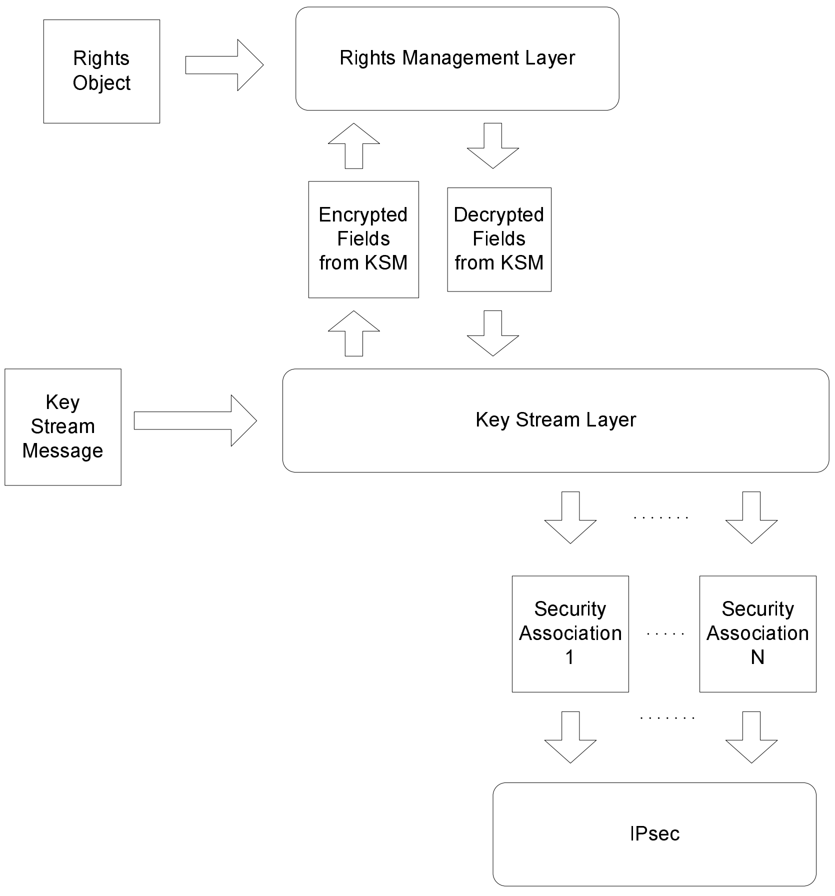


Figure B.24: IPsec Security Association Elements

The instantiation of security associations is performed by the key stream layer and is driven by key stream messages and Rights Objects. Given a key stream message, the key stream layer extracts the encrypted fields from that message. The key stream passes these and other relevant fields to the rights management layer. For each Rights Object stored in the device, the rights management layer examines if that Rights Object would be able to decrypt the fields in the key stream message. If the rights layer does find a suitable rights object, then it decrypts these fields using the appropriate rights management system and rights object. The decrypted fields are provided back to the key stream layer which - based on the key stream message and the decrypted fields - instantiates a set of security associations. If the rights management layer does not find a suitable rights object, then the key stream message SHOULD be silently dropped.

B.3.1.2 ISMACryp

Figure B.25 shows how ISMACryp is used with the Rights Management and Key Stream Layers.

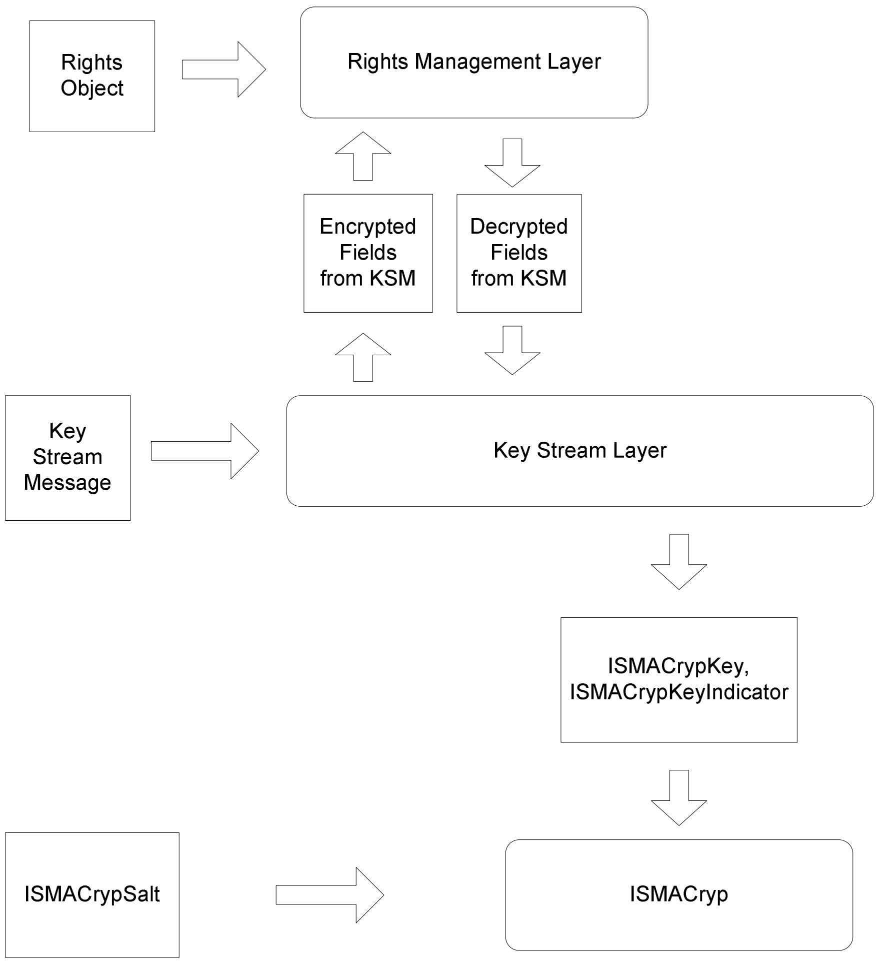


Figure B.25: ISMACryp Key Management

When ISMACryp is used, the key stream message carries encrypted ISMACrypKey in the traffic key material field, and ISMACrypKeyIndicator, which is used to associate the ISMACrypKey with a particular access unit of the content stream. Given a key stream message, the key stream layer extracts the encrypted fields from that message. The encrypted fields are sent to the rights management layer for decryption. If the rights management layer does find a suitable Rights Object, then it decrypts the fields using the appropriate rights management system and rights object. The decrypted ISMACrypKey is sent along with the ISMACrypKeyIndicator to the ISMACryp content decrypter. When an access unit with a matching ISMACrypKeyIndicator is received, the ISMACryp decrypter uses the ISMACrypKey to decrypt the content. The ISMACrypSalt parameter is signalled in the SDP attributes of the stream. If the rights management layer does not find a suitable Rights Object, then the key stream message SHOULD be silently dropped.

B.3.1.3 SRTP

Figure B.26 shows the different objects and elements involved in building SRTP cryptographic context.

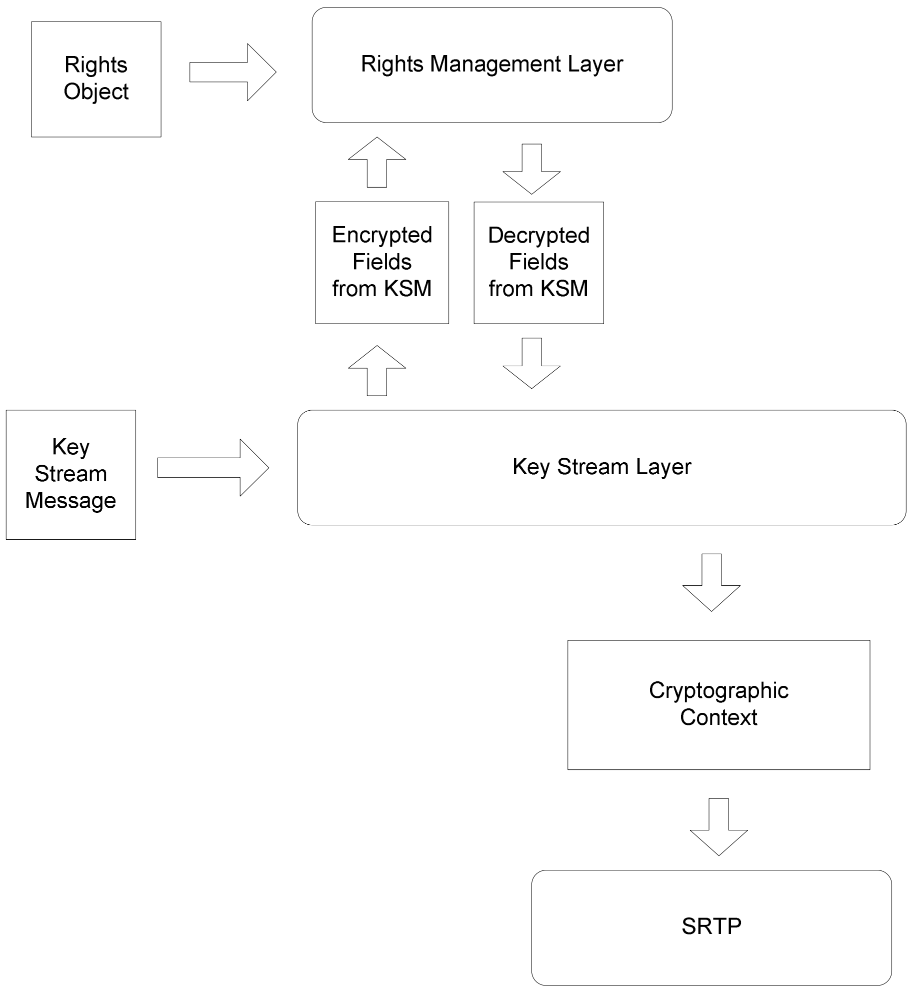


Figure B.26: SRTP Cryptographic Context Management

As with IPsec security associations, the instantiation of a cryptographic context is performed by the key stream layer and is driven by key stream messages and Rights Objects. A key stream message includes parameters that are specific to the selected content encryption layer. In the case of SRTP, a key stream message includes an MKI (Master Key Index) which is necessary to identify an SRTP cryptographic context. The SRTP ROC (Roll-Over-Counter) values for each component stream is communicated in the integrity transform of SRTP packets in order to keep track of how many times each RTP sequence number has wrapped around. This is required for keeping receivers synchronized with the current SRTP session. Because it is possible that a-received ROC value is already out-of-date by the time that a receiver starts decrypting packets, there is additional information provided in the SRTP packets that allows receivers to adjust the ROC value appropriately.

Just as it is the case with IPsec, an encrypted traffic key is extracted from a key stream message and is passed to the rights management layer for decryption. If the rights management layer finds a valid Rights Object for this traffic key, it will be decrypted and converted to an SRTP Master Key. The SRTP content decryption layer then creates an SRTP session with decryption and (optionally) authentication keys that are derived from the Master Key as required by SRTP. If the rights management layer does not find a suitable Rights Object, then the key stream message SHOULD be silently dropped.

B.3.2 Key Stream Layer

The key stream layer is made up of key stream messages, which are distributed in-band, together with the protected media streams.

B.3.2.1 Key Stream Message (KSM)

Each KSM SHALL be encapsulated in exactly 1 UDP packet.

In order to keep access times low for devices that start accessing a service, a KSM SHALL be transmitted periodically.

The KSM SHALL be transported in-band, in the same Elementary Stream together with the media streams that are protected with the traffic keys contained in the KSM.

Table B.3: Format of Key Stream Message

Key_Stream_Message_Description	Length	Type
key_stream_message() {		
selectors_and_flags {		
protocol_version	4	uimsbf
reserved_for_future_use	3	bslbf
access_criteria_flag	1	uimsbf
traffic_protection_protocol	3	uimsbf
traffic_authentication_flag	1	uimsbf
next_traffic_key_flag	1	uimsbf
timestamp_flag	1	uimsbf
programme_flag	1	uimsbf
service_flag	1	uimsbf
}		
if (traffic_protection_protocol == KSM_ALGO_IPSEC) {		
security_parameter_index	32	uimsbf
if (next_traffic_key_flag == KSM_FLAG_TRUE) {		
next_security_parameter_index	32	uimsbf
}		
}		
if (traffic_protection_protocol == KSM_ALGO_SRTP) {		
master_key_index_length	8	uimsbf
master_key_index	8 × master_key_index_length	uimsbf
reserved_for_future_use	7	bslbf
master_salt_flag	1	uimsbf
}		
if (traffic_protection_protocol == KSM_ALGO_ISMACRYP) {		
key_indicator_length	8	uimsbf
key_indicator	8 × key_indicator_length	bslbf
if (next_traffic_key_flag == KSM_FLAG_TRUE) {		
key_indicator	8 × key_indicator_length	bslbf
}		
}		
encrypted_traffic_key_material_length	8	uimsbf
encrypted_traffic_key_material	8 × encrypted_traffic_key_material_length	bslbf
if (next_traffic_key_flag == KSM_FLAG_TRUE) {		
next_encrypted_traffic_key_material	8 × encrypted_traffic_key_material_length	bslbf
}		
reserved_for_future_use	5	bslbf
traffic_key_lifetime	3	uimsbf
if (timestamp_flag == KSM_FLAG_TRUE) {		
timestamp	40	mjdutc
}		
if (access_criteria_flag == KSM_FLAG_TRUE) {		
reserved_for_future_use	8	bslbf
number_of_access_criteria_descriptors	8	uimsbf
access_criteria_descriptor_loop() {		
access_criteria_descriptor()		
}		
}		
if (programme_flag == KSM_FLAG_TRUE) {		
programme_selectors_and_flags {		

Key_Stream_Message_Description	Length	Type
`        reserved_for_future_use`	7	bslbf
`        permissions_flag`	1	uimsbf
`    }`		
`    if (permissions_flag == KSM_FLAG_TRUE) {`		
`        permissions_category`	8	uimsbf
`    }`		
`    if (service_flag == KSM_FLAG_TRUE) {`		
`        encrypted_PEK`	128	bslbf
`    }`		
`    programme_CID_extension`	32	uimsbf
`    programme_MAC`	96	bslbf
`  }`		
`  if (service_flag == KSM_FLAG_TRUE) {`		
`    service_CID_extension`	32	uimsbf
`    service_MAC`	96	bslbf
`  }`		
`}`		

B.3.2.1.1 Descriptors for access_criteria_descriptor_loop

Table B.4: Descriptors for access_criteria_descriptor_loop

Access_Criteria_Descriptor	Length	Type
tag	8	uimsbf
length	8	uimsbf
value	8 × length	bslbf

The access criteria descriptor loop is an extension mechanism to allow the addition of new access criteria in the future versions of the present document. The device SHALL ignore access criteria descriptors that it does not support.

A single access criteria descriptor can carry one or more access criteria.

The following access criteria descriptors have been defined:

parental_rating - is the parental rating of the programme. The descriptor tag for this descriptor is 1. The value for this descriptor is encoded as follows:

Table B.5: parental_rating Access Criteria Descriptor

parental_rating descriptor	Length	Type
`rating_type`	7	uimsbf
`country_code_flag`	1	uimsbf
`rating_value`	8	uimsbf
`if (country_code_flag == KSM_FLAG_TRUE) {`		
`  number_of_country_codes`	8	uimsbf
`  for (i = 0; i < number_of_country_codes; i++) {`		
`    country_code`	16	uimsbf
`  }`		
`}`		

The optional list of **country_code** specifies that the rating is for a specific list of one or more countries, which is analogous to the MPEG-7 definition of the ParentalGuidanceType. Each country code is a 2-character value that must be compliant with ISO 3166 [19].

The **rating_type** with values 0 through 8 specifies one of the content rating systems that are defined by MPEG-7 and **rating value** is an integer with the meaning that is dependent on the rating_type. The rating values for rating type 0 through 8 are exactly as they had been defined by MPEG-7. Rating type 9 is for the parental rating for the German system.

Table B.6

rating_type	Name	Description	rating_value
0	N/A	EN 300 468 [14] for the parental_rating_descriptor in DVB systems	Minimum allowable age.
1	JapaneseAdmCommMotionPictureCodeEthicsParentalRatingCS	Japanese Motion Picture Parental Rating	1=PG12 2=R-15 3=R-18 4=None
2	ICRAParentalRatingCS	Internet Content Rating Association Parental Rating	1=Level4 2=Level3 3=Level2
3	MPAAParentalRatingCS	MPAA Parental Rating	1=G 2=PG 3=PG-13 4=R 5=NC-17 6=NR
4	ICRAParentalRatingNudityCS	Internet Content Rating Association Parental Rating for Nudity	1=Level 4 2=Level 3 3=Level 2 4=Level 1 5=Level 0 6=None
5	RIAAParentalRatingCS	RIAA Parental Rating	1=Parental advisory 2=None
6	ICRAParentalRatingSexCS	Internet Content Rating Association Parental Rating for Sex	1=Level 4 2=Level 3 3=Level 2 4=Level 1 5=Level 0 6=None
7	MPAAParentalRatingTVCS	MPAA Parental Rating for TV	1=TVY 2=TVY7 3=TVG 4=TVPG 5=TV14 6=TVMA 7=None
8	ICRAParentalRatingViolence		1=Level 4 2=Level 3 3=Level 2 4=Level 1 5=Level 0 6=None
9	GermanyFSK	German Freiwillige Selbstkontrolle der Filmwirtschaft Rating System	1=0 (Freigegeben ohne Altersbeschränkung) 2=6 (Freigegeben ab 6 Jahren) 3=12 (Freigegeben ab 12 Jahren) 4=16 (Freigegeben ab 16 Jahren) 5=18 (Keine Jugendfreigabe)

B.3.2.1.2 Constants

Table B.7: Constants in Key Stream Message

Name	Value
KSM_ALGO_IPSEC	0
KSM_ALGO_SRTP	1
KSM_ALGO_ISMACRYP	2
KSM_FLAG_FALSE	0
KSM_FLAG_TRUE	1

B.3.2.1.3 Coding and Semantics of Attributes

protocol_version: indicates the protocol version of this key stream message.

The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).

NOTE: If set to 0x0 the format specified in the present document is used. If set to anything else than 0x0, then the format is beyond the scope of the present document.

traffic_protection_protocol: defines the protocol used for the encryption and optional authentication of traffic:

KSM_ALGO_IPSEC = IPsec ESP (transport mode; encryption: AES-128-CBC [key length 128]; authentication: HMAC-SHA1-96 [key length 160] or NULL);

KSM_ALGO_SRTP = SRTP (encryption: AES-128-CTR [key length 128]; authentication: HMAC-SHA1-80 [key length 160] or NULL);

KSM_ALGO_ISMACRYP = ISMACryp (encryption: AES-128-CTR [key length 128]; authentication is not used);

other values = reserved for future use.

Whether or not authentication is used depends on <traffic_authentication_flag>.

traffic_authentication_flag: defines whether or not the traffic is authenticated:

KSM_FLAG_FALSE = traffic authentication is not used;

KSM_FLAG_TRUE = traffic authentication is used, and the algorithm depends on <traffic_protection_protocol>.

next_traffic_key_flag: indicates whether or not the key stream message contains the next traffic key material:

KSM_FLAG_FALSE = the key stream message contains only the current traffic key material;

KSM_FLAG_TRUE = the key stream message contains both the current and the next traffic key material.

The next traffic key material SHALL be included at least 1 second before it becomes current. This is to enable the devices to process the traffic key material and put the necessary security associations in place before the media packets start arriving that are encrypted with the next traffic encryption key.

The next traffic key material SHALL NOT be included earlier than 1 minute before it becomes current. This is to limit the effect on pay-per-view enforcement that is caused by sending the next traffic key material, encrypted with the encryption key of a programme that may end before the next traffic key becomes current, to maximally 1 minute.

The above times SHALL be relative to the moment of transmission of the key stream messages.

timestamp_flag: indicates whether or not the key stream message contains a timestamp:

KSM_FLAG_FALSE = the key stream message does not contain a timestamp;

KSM_FLAG_TRUE = the key stream message contains a timestamp.

programme_flag: indicates whether or not the programme key layer is present in the key stream message:

> KSM_FLAG_FALSE = the programme key layer is not present, i.e. the optional programme key layer is not used for the service;
>
> KSM_FLAG_TRUE = the programme key layer is present, i.e. the optional programme key layer is used for the service.

<programme_flag> and <service_flag> SHALL NOT both be 0. All other combinations are allowed, indicating that either or both of the key layers are present.

service_flag: indicates whether or not the service block is present in the key stream message:

> KSM_FLAG_FALSE = the service key layer is not present, i.e. the optional service key layer is not used for the service;
>
> KSM_FLAG_TRUE = the service key layer is present, i.e. the optional service key layer is used for the service.

<programme_flag> and <service_flag> SHALL NOT both be 0. All other combinations are allowed, indicating that either or both of the key layers are present.

security_parameter_index: provides the link to the IPsec ESP header.

Upon reception of a protected IP packet, the device SHALL use the security parameter index (SPI) to identify (look up) the correct security association and thereby find the decryption and authentication keys to be used for the received IPsec ESP packet. The SPI value SHALL be in the range 0x00000100 to 0xFFFFFFFF. An incoming ESP packet containing the SPI value specified in this field SHALL use the key material provided in the encrypted traffic key material field as key material for the decryption operation.

next_security_parameter_index: provides the link to the IPsec ESP header.

This field is present in the packet only if next traffic key flag is set to true. This field then contains the IPsec SPI value corresponding to the next_encrypted traffic key material field. The value of the SPI SHALL be in the range 0x00000100 to 0xFFFFFFFF. An incoming ESP packet containing the SPI value specified in this field SHALL use the key material provided in the next encrypted traffic key material field as key material for the decryption operation.

master_key_index_length: provides the length of the master_key_index field.

This field gives the length of the master_key_index field in bytes.

master_key_index: provides the link to the SRTP header:

Upon reception of a protected RTP packet, the device SHALL use the master key index (MKI) to identify (look up) the correct security association and thereby find the decryption and authentication keys to be used for a received SRTP packet.

This field is a sequence of 8-bit values. The sequence consists of master_key_index_length bytes. The bytes are in the same order that they will be in an SRTP packet and SHALL be in SRTP [32] network byte-order when extracting the MKI value.

The MKI is associated with the current TEK. If the next traffic key flag is set to 1, the MKI associated with the 'next TEK' is implicitly defined as MKI+1.

master_salt_flag: specifies if the master salt is included in the SRTP parameters. For DVB-IPDC 18Crypt Profile, this flag is always false, thus indicating that the master salt is not provided in the message and that the terminal shall consider the master salt to be set to a NULL value consisting of 112 0-bits.

key_indicator_length: the length of key_indicator for ISMACryp in bytes (1…255).

key_indicator: is used by ISMACryp to associate ISMACrypKey with the access unit(s) of content encrypted with it.

encrypted_traffic_key_material_length: is the length in bytes of the encrypted traffic key material.

The length of the traffic key material depends on the encryption and authentication algorithm, and is obtained by adding the respective key sizes. Encryption MAY require the clear-text key material to be padded.

encrypted_traffic_key_material: is the key material currently used for encryption and optional authentication of the traffic, encrypted using AES-128-CBC, with fixed IV 0, and with 0 padding in the last block, if needed.

If <programme_flag> == KSM_FLAG_TRUE, the traffic key material is encrypted with the programme encryption key (PEK).

If <programme_flag> == KSM_FLAG_FALSE and <service_flag> == KSM_FLAG_TRUE, the traffic key material is encrypted with the service encryption key (SEK).

After decryption (and discarding any padding), the traffic encryption key (TEK) and the traffic authentication key (TAK) are obtained in a way that depends on the protocol used for traffic protection:

1) IPsec:
If no traffic authentication is used, the TEK is identical to the decrypted traffic key material (16 bytes).
If traffic authentication is used, TEK and traffic authentication seed (TAS) are obtained by splitting the decrypted traffic key material into two parts, where the TEK is identical to the first 16 bytes, and the TAS is identical to the second 16 bytes. The TAK (20 bytes) is derived from the TAS, as described in clause B.14.

2) SRTP:
The master key is identical to the decrypted traffic key material and SHALL always be a 16-byte AES key as required by SRTP. SRTP specifies how to derive session encryption and authentication keys from the master key using a derivation function based on AES in counter mode.

3) ISMACryp:
No traffic authentication is used. TEK is identical to the decrypted traffic key material (16 bytes).
TEK is the binary representation of the 'aes-key' part of the ISMACrypKey. The complete ISMACrypKey (as understood by an ISMACrypKey parser) can be constructed by the device using the ISMACrypSalt in the SDP file and the KSM parameter key_indicator as follows:

ISMACrypKey = (key)BASE64(TEK||ISMACrypSalt)|2^64|key_indicator

The optional ISMA lifetime parameter is not signalled, assuming 2^64 (the default).

next_encrypted_traffic_key_material - is the encrypted key material used for encryption and optional authentication of the traffic after the current crypto period is over and the next crypto period starts. The structure of this attribute is similar to encrypted_traffic_key_material attribute.

traffic_key_lifetime - denotes is the lifetime of the traffic key material, relative to the first occurrence of an SPI, MKI or key_indicator.

If <traffic_key_lifetime> is ***n***, then the actual lifetime is $\mathbf{2^{n}}$ seconds, as presented in table B.8:

Table B.8: Traffic Key Lifetime

value of traffic_key_lifetime attribute	0	1	2	3	4	5	6	7
actual lifetime of traffic key material (seconds)	1	2	4	8	16	32	64	128

The actual duration of the crypto period SHALL be strictly shorter than the defined lifetime of the traffic key material. Typically, an SPI or MKI appears for the first time implicitly, when the 'next' traffic key material is included in a KSM. Any safety margins to cope with network and transmission delays SHALL be added by the network. A typical value for the lifetime could be three times the crypto period.

The maximal value for the crypto period duration is in practice slightly shorter than the traffic key lifetime, because the KSM will include the 'current' and 'next' traffic key material before a change of crypto period, to allow the devices to set up the security associations.

After the lifetime has expired, the security association containing the traffic key can be safely deleted by the device. This may help managing the security association database in the device or enable other optimisations.

The maximum value for the traffic key lifetime is defined mainly in order to have a strict upper bound for the effect of the 'sneak post view' problem: the 'next traffic key' material is distributed under the current PEK, and allows viewers to view a programme during the next crypto period. Should this possibility still be of a concern, the network MAY choose a shorter crypto period than the maximum value, or, during the crypto period where the current programme ends and a new programme starts, choose to distribute the 'current' and the 'next' traffic key material in separate KSMs, encrypted with their respective PEKs.

timestamp - Field containing a timestamp at the point of sending the key stream message. The timestamp SHALL be used as a reliable time of reception of the associated media stream for post-acquisition permissions. The device SHALL not use the timestamp as a reliable source for DRM time.

The format of the 40-bit mjdutc field is specified in clause B.9. This 40-bit field contains the timestamp of the key stream message in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD).

EXAMPLE 1: 93/10/13 12:45:00 is coded as "0xC079124500".

access_criteria_flag: indicates whether or not access criteria are defined for the programme:

KSM_FLAG_FALSE = no access criteria are defined. Access to the programme is governed by RO(s) associated with this programme or with the service this programme is a part of.

KSM_FLAG_TRUE = access criteria are defined, implying that the device is allowed to access the programme only if the specified access criteria are met and if the device has an RO granting access to the programme.

permissions_flag: indicates whether or not permissions category is defined for the programme:

KSM_FLAG_FALSE = no permissions category is defined.

KSM_FLAG_TRUE = permissions category is defined.

number_of_access_criteria_descriptors: indicates the number of access criteria descriptors.

permissions_category: indicates the permissions category for the programme:

Table B.9

Value	Description
0x00	no permissions category, service RO applies as such
0x01…0x3F	permissions_category is included in the post-acquisition permissions lookup
0x40…0xFE	reserved for future standardization
0xFF	no post-acquisition content protection (export in plaintext is allowed)

If permissions_category is in the range 0x01 to 0x3F,

in case of ICRO, the device SHALL use as service_CID for post-acquisition permissions lookup the text string:

service_CID = socID || "#S" || serviceBaseCID || "@" || hex(service_CID_extension) || "_" || hex(permissions_category); and

then apply the permissions specified in the service ICRO for this asset;

in case of BCRO, the device SHALL look up the permissions specified in the service BCRO for the asset that has a matching permissions_category field.

If permissions_category is in the (reserved for future standardization) range 0x40 to 0xFF, and device does not support it, device SHALL drop (i.e. ignore) all post-acquisition permissions (like play, redistribute etc.) indicated in the service RO, or if device cannot do such permissions dropping, allow real-time rendering of the streaming content only (i.e. refuse to record the content, or to redistribute it in real time). Permissions_category has no impact on a Programme RO. The permissions delivered in a Programme RO apply as such.

encrypted_PEK: is the programme encryption key (PEK) used within the current key stream message to decrypt the traffic key material, encrypted using AES-128-CBC with fixed IV 0).

The programme encryption key is encrypted with the service encryption key (SEK).

programme_CID_extension: is the extension of the programme_CID which allows to identify the PEAK that has been delivered to the device within a Programme RO.

The CID/BCI of the programme key is constructed as:

```
programme_CID = socID || "#P" || serviceBaseCID || "@" || hex(programme_CID_extension)

programme_BCI = hash(socID || "#P" || serviceBaseCID || "@") || programme_CID_extension
```

The socID and serviceBaseCID are string values and are expected to be part of the service guide. Upon reception of a KSM, the device can assemble the programme_CID/BCI and look up the programme key (wrapped inside an RO). The device SHOULD check whether it has a Programme RO for the programme_CID/BCI.

The hex() function is a hexadecimal presentation of the parameter containing hexadecimal characters 0 to 9 and a-f (in lowercase) with possible preceding zeros.

EXAMPLE 2: For a 16 bit value 2 748, hex() returns "0abc". Note that two characters are always generated for each byte.

The hash function for the construction of programme_BCI is defined in clause B.10, where BCI is defined. It does not depend on the contents of the KSM, and can thus be pre-computed.

programme_MAC: is the HMAC-SHA-1-96 according to [23] and [25] calculated over all preceding fields of the key stream message. It is used to authenticate the relevant part of the key stream message with PAK in case of pay-per-view, where a PEK from a Programme RO is used to directly decrypt the traffic key material.

In case the device is accessing the key stream message with a Programme RO, the device SHALL compute the programme MAC, and drop the message if authentication fails. In this case, <programme_MAC> MAY also be used to detect and drop duplicates (it can be expected that a particular key stream message is repeated multiple times, in order to keep access times short for devices that newly start receiving a broadcast transmission).

In case the device is accessing the key stream message with a Service RO, it will not be able to compute the programme MAC, and there is no need for it to do so.

service_CID_extension: is the extension of the service_CID which allows to identify the SEAK that has been delivered to the device within a Service RO.

The CID/BCI of the service key is constructed as:

```
service_CID = socID || "#S" || serviceBaseCID || "@" || hex(service_CID_extension)
service_BCI = hash(socID || "#S" || serviceBaseCID || "@") || service_CID_extension
```

The socID and serviceBaseCID are string values and are expected to be part of the service guide. Upon reception of a KSM, the device can assemble the service_CID/BCI and look up the service key (wrapped inside an RO). The device SHOULD check whether it has a Service RO for the service_CID/BCI.

The hex() function is a hexadecimal presentation of the parameter containing hexadecimal characters 0 to 9 and a-f (in lowercase) with possible preceding zeros.

EXAMPLE 3: for a 16 bit value 2 748, hex() returns "0abc". Note that two characters are always generated for each byte.

The hash function for the construction of service_BCI is defined in B.10, where BCI is defined. It does not depend on the contents of the KSM, and can thus be pre-computed.

If the permissions_category field is present and has a nonzero value, the Service_CID of the service is constructed as specified above (at description of the permissions_category field).

service_MAC: is the HMAC-SHA-1-96 according to RFC 2104 [23] and 2 404 calculated over all preceding fields of the key stream message. It is used to authenticate the key stream message with SAK in case of subscription, where a SEK from a Service RO is used to decrypt the PEK and further decrypt the traffic key material.

In case the device is accessing the key stream message with a Service RO, the device SHALL compute the service MAC, and drop the message if authentication fails, i.e. if the computed MAC does not correspond to <service_MAC>. In this case, <service_MAC> MAY also be used to detect and drop duplicates (it can be expected that a particular key stream message is repeated multiple times, in order to keep access times short for devices that newly start receiving a broadcast transmission).

In the case that the device is accessing the key stream message with a Programme RO, it NEED NOT compute the service MAC.

B.3.2.2 Key Stream Discovery

The access description to a particular service which is distributed as part of the Service Guide is assumed to contain a media description for each IP flow of the media service itself.

Based on the basic assumption that the service cannot be consumed (because the used IP addresses, codecs, and other 'technical' parameters are not known) unless the access description is present in the device; the access description will also carry the static security-related parameters of the service or of a session of the service.

Devices SHALL be able to buffer the access description, in order to ensure quick service access without need for Service Guide acquisition.

Therefore, the access description can only contain parameters that are likely to change very infrequently for a particular service, so that it can be tolerated that in case of a change, the device performs service guide acquisition before accessing a service.

The following access information pertaining to the traffic key stream SHALL be added to the access description of the service:

port_of_key_stream: is the port number of the UDP stream carrying the KSM flow.

IP_address_of_key_stream: is the IP address on which the key stream is transported, which can be an IPv4 or and IPv6 address.

To the service entity itself, the following information SHALL be added:

serviceBaseCID: is the static part of all CIDs of Service Encryption / Authentication Keys (SEAKs) and Programme Encryption / Authentication Keys (PEAKs) pertaining to the service.

B.3.3 Rights Management Layer

All devices SHALL implement the rights management layer as specified in this clause. Devices MAY additionally implement alternative schemes for the functionality of the rights management and registration layers, see clause B.2.7 for details.

In case of **subscription**, the SEAK associated with the service is delivered to the authorized device in an RO that is bound to a device, a unique group, a subscriber group or to a domain. Such an RO is called a Service RO. In general, a Service RO will contain key material associated with more than one service (with a service bundle).

In case of **pay-per-view**, the PEAK associated with a pay-per-view event is delivered to the authorized device directly within an RO that is bound to a device, a unique group, a subscriber group or to a domain. Such an RO is called a 'Programme RO'.

The ID of ROs that contain SEAKs or a PEAK needs to be structured, to allow for the management of purchase transactions in the device, or more specifically, to create an association between the service guide (where the purchase item is expected to be announced) and the successful completion of the purchase transaction (when the RO related to the purchase has finally been received in the device). This is valid for both interactive and especially for broadcast mode of operation, where the RO may be received by the device much later than the purchase transaction is initiated.

Defining a structured ID will allow the device also to check later on whether ROs for all subscribed services are available (and have been renewed). The rekeying_period_number is an increasing number by which the roID related to the same purchase item can be made unique.

The ID of an OMA DRM 2.0 RO delivered over the interactivity channel (i.e. the ID of an ICRO) linked with subscription (Service RO) or pay-per-view (Programme RO), and bound to a device or to a domain, and SHALL be constructed respectively as follows:

```
deviceRoID = "E" || deviceID || "_S" || socID || "_I" || purchase_item_id || "_" ||
hex(rekeying_period_number)
domainRoID = "O" || domainID || "_S" || socID || "_I" || purchase_item_id || "_" ||
hex(rekeying_period_number)
```

The hex() function is a hexadecimal presentation of the parameter containing hexadecimal characters 0 to 9 and a-f (in lowercase) with possible preceding zeros. EXAMPLE: for a 16 bit value 2 748, hex() returns "0abc". Note that two characters are always generated for each byte.

In the case of BCROs, the link with the corresponding subscription (Service RO) or pay-per-view (Programme RO) is obtained by using the field purchase_item_id and rekeying_period_number (see clause B.3.3.4.2).

B.3.3.1 Requirements for Service ROs

A Service RO SHALL contain at least one asset, i.e. one (<service_period_CID/BCI >, <wrapped_SEAK>) pair. The CID/BCI pertaining to a particular re-keying period of the service SHALL be constructed as specified in clause B.10.

For BCROs, after unwrapping the SEAK contained in the RO the service encryption key (SEK) and the service authentication seed (SAS) are obtained by splitting the unwrapped key material into two parts as follows:

SEK = first part (128 bits, since AES-128 is used to encrypt the traffic key material or the PEK).

SAS = second part (128 bits).

For ICROs, SEK and SAS are carried as separate elements in the asset, as described in clause B.14.2.1.

The SAK (160 bits, since HMAC-SHA-1-96 is used to calculate the service_MAC) is derived from the SAS, as described in clause B.14.

The Service RO SHALL contain a rights expression defining a 'datetime' constraint for the 'access' permission. The time interval defined by this constraint SHALL correspond to the time interval during which the <SEAK> can be used to get access to the key stream message. The device SHOULD use this information in order to determine the appropriate time for re-keying the Service RO.

All SEAKs that are bundled in a single Service RO SHALL have the exact same lifetime, and the 'access' permission SHALL be applicable to all SEAKs. This allows the device to determine the time for RO renewal in an unambiguous way.

If and only if post-delivery content protection is used in a system, the Service RO MAY contain further rights expressions restricting post-acquisition usage, and such restrictions SHALL be enforced

The post-acquisition permissions MAY be different for different programmes, depending on the permissions_category value signalled in KSM (see clause B.3.2). The Service RO SHALL specify (as separate assets) the permissions for each of the categories in the range 0x01 to 0x3F that are used in the service.

The 'access' permission is a new extension to OMA DRM for the purpose of defining rights to service protection in a clean manner that is distinguishable from usage rules defined for content protection. For maximum compatibility with older OMA DRM implementations, it is recommended that the 'execute' permission be granted as well.

B.3.3.2 Requirements for Programme ROs

A Programme RO SHALL contain exactly one asset, i.e. a (<programme_CID/BCI>, <wrapped_PEAK>) pair. The CID/BCI pertaining to the programme SHALL be constructed as specified in clause B.10.

For BCROs, after unwrapping the PEAK contained in the RO, the programme encryption key (PEK) and the programme authentication seed (PAS) are obtained by splitting the unwrapped key material into two parts as follows:

PEK = first part (128 bits, since AES-128 is used to encrypt the traffic key material).

PAS = second part (128 bits).

For ICROs, PEK and PAS are carried as separate elements in the asset, as described in clause B.14.2.1.

The PAK (160 bits, since HMAC-SHA-1-96 is used to calculate the programme_MAC) is derived from the PAS, as described in clause B.14.

The Programme RO SHALL contain a rights expression defining a 'datetime' constraint for the 'access' permission. The time interval defined by this constraint SHALL correspond to the time interval during which the <PEAK> can be used to get access to the key stream message.

If and only if post-delivery content protection is used in a system, the Programme RO MAY contain further rights expression restricting post-acquisition usage, and such restrictions SHALL be enforced.

The 'access' permission is a new extension to OMA DRM for the purpose of defining rights to service protection in a clean manner that is distinguishable from usage rules defined for content protection. For maximum compatibility with older OMA DRM implementations, it is recommended that the 'execute' permission be granted as well.

B.3.3.3 Delivery of ICROs over Interactivity Channel

When the Interactivity Channel is used to deliver Rights Objects, all aspects of Rights Object delivery are governed by OMA DRM 2.0 specifications [49]. Rights Objects delivered over the interactivity channel are referred to as Interactivity Channel Rights Objects (ICRO).

B.3.3.4 Delivery of BCROs over Broadcast Channel

B.3.3.4.1 Broadcast of BCRO Objects

BCRO Objects SHALL be broadcast within an RI Service, as defined in clause B.4. The RI Service streams used by individual Rights Issuers are identified by the ESG.

B.3.3.4.2 Format of a Broadcast Rights Object (BCRO)

The following tables define the format of a BCRO. The *asset*, *permission* and *constraint* object correspond in their meaning to their counterparts in [49]. The *action* object corresponds to the allowed elements in the [50] permissions element.

BCROs from one rights issuer are normally carried in one stream and this stream does only contain BCROs from this one rights issuer. As the rights issuer id is thereby already given by the BCRO stream the BCRO does not have to carry this information. If however BCROs from different rights issuers have to be carried in one stream then the rights issuer id SHALL be signalled in every BCRO. This is done by setting the rights_issuer_flag to 1.

Table B.10: Format of BCRO

Field	Length	Type
`BCRO() {`		
`    /* MAC protected part starts here */`		
`    message_tag`	8	uimsbf
`    version`	4	uimsbf
`    bcro_length`	12	uimsbf
`    group_size_flag`	1	bslbf
`    timestamp_flag`	1	bslbf
`    stateful_flag`	1	bslbf
`    refresh_time_flag`	1	bslbf
`    address_mode`	3	uimsbf
`    rights_issuer_flag`	1	bslbf
`    Address`	32	uimsbf
`    if(address_mode == 0x1 && group_size_flag == 0){`		
`        bit_access_mask`	256	bslbf
`    }else if(address_mode == 0x1 && group_size_flag == 1){`		
`        bit_access_mask`	512	bslbf
`    }else if (address_mode&0x6 == 0x2){`		
`        position_in_group`	8	uimsbf
`    }else if (address_mode == 0x4){`		
`        domain_id_extension`	6	bslbf
`        domain_generation`	10	uimsbf
`    }`		

Field	Length	Type
`    if(rights_issuer_flag == 1){`		
`        rights_issuer_id`	160	bslbf
`    }`		
`    if(timestamp_flag == 1){`		
`        bcro_timestamp`	40	mjdutc
`    }`		
`    if(refresh_time_flag == 1){`		
`        refresh_time`	40	mjdutc
`    }`		
`    permissions_flag`	1	bslbf
`    rekeying_period_number`	7	uimsbf
`    purchase_item_id`	32	uimsbf
`    number_of_assets`	8	uimsbf
`    for(i=0;i<number_of_assets;i++){`		
`        asset()`		
`    }`		
`    if(permissions_flag == 1){`		
`        number_of_permissions`	8	uimsbf
`        for(i=0;i<number_of_permissions;i++){`		
`            permission()`		
`        }`		
`    }`		
`    /* MAC protected part ends here */`		
`    MAC`	96	bslbf
`}`		

message_tag: Tag identifying this message as a BCRO. The value for this filed is defined in clause B.19.

version: 3-bit flag which indicates the version of the BCRO message format. If set to 0 the original format is used. Devices SHALL ignore BCROs with versions it does not support.

bcro_length: 12-bit field indicating the length in bytes of the BCRO starting immediately after this field. The size of a BCRO including its header SHALL NOT exceed 4 096 bytes.

group_size_flag: 1-bit field indicating the group size used. 0 - a maximum group size 256 is used, 1 - a maximum group size of 512 is used.

timestamp_flag: 1-bit field indicating that the BCRO is timestamped.

stateful_flag: 1-bit flag indicating that when set to 1 the BCRO contains stateful information.

refresh_time_flag: 1-bit flag indicating that a refresh_time for the BCRO is contained in this BCRO.

address_mode: 3-bit field indicating the addressing mode used by this BCRO. Table B.11 lists all four possible address modes.

Table B.11: address_mode

address_mode	Description
0x0	addressing whole of unique group
0x1	addressing of subscriber group using a bit_mask size of 256 bit or 512 bit depending on group_size_flag (subset of unique group)
0x2 to 0x3	addressing of unique device
0x4	addressing of OMA domain. Address field concatenated with the domain_id_extension will be the domain id in this case
0x5 to 0x7	reserved

rights_issuer_flag: 1-bit flag indicating that the rights issuer id is listed in this BCRO. Normally this information is given via a dedicated BCRO stream. This flag will only be set if BCROs from different rights issuers are carried in the same stream.

address: 4-byte group address. Each rights issuer has its own address space. If the group_size is 512 then the group address is made of the first 31 bit of the address field. If the BCRO is addressed to a unique device in a group then the LSB of the address field is the MSB of the group position.

If the address_mode is set to 0x4 the address field contains the first 32 bit of the short form domain_id.

bit_access_mask: If the BCRO addresses a subset of a unique group (address_mode 0x1) than the bit_access_mask defines to which devices in the group this BCRO is addressed to. Devices not listed in the bit_access_mask cannot decrypt the key material in this BCRO as zero message broadcast encryption is used for the encryption of the key material. The size of the bit_access_mask is given by the group_size_flag.

position_in_group: If the BCRO addresses a unique device then this field specifies the position of the unique device in the given subscriber group. If group_size_flag is 0 than the position in the group is directly given by the position_in_group field. If group_size_flag is 1 then 9 bit are used to identify the position in the group. If group_size_flag is 1 then the LSB (bit 0) from the address field is used as the 9th bit, the MSB. The real position in the group is then given by:

```
int real_position_in_group;

if(address_mode&0x6==0x2){

  if(group_size_flag == 0){

     //maximum size of 256 devices in group.

     real_position_in_group = position_in_group;

  }else{

     //maximum size of 512 devices in group;

     real_position_in_group = ((address&0x1)<<8)| position_in_group;

  }

}
```

domain_id_extension: The domain_id is given by the address field concatenated with the domain_id_extension to form a 38 bit id:

```
domain_id = (address<<6)|domain_id_extension
```

domain_generation: This 10 bit field specifies the generation of the domain.

rights_issuer_id: The ID of the rights issuer. This is the 160-bit SHA-1 hash of the public key of the RI. See X509PKISHash in [49].

bcro_timestamp: Field containing a timestamp at the point of issuing of the BCRO. The format of the 40-bit mjdutc field is specified in clause B.9. This 40-bit field contains the timestamp of the BCRO in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD).

refresh_time: The refresh_time specifies the time when the device should acquire a new BCRO. It does not specify when the keys in the BCRO expire. This field is a hint to a device to acquire a new BCRO for the content listed in the BCRO before the keys in the BCRO expires. The format of the 40-bit mjdutc field is specified in clause B.9. This 40-bit field contains the expiry time of the BCRO in Universal Time, Co-ordinated (UTC) and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4-bit Binary Coded Decimal (BCD).

EXAMPLE 1: 93/10/13 12:45:00 is coded as "0xC079124500".

permissions_flag: 1-bit flag indicating that the BCRO contains at least 1 permission.

rekeying_period_number: 7-bit counter used to differentiate between different ROs with the same purchase_item_id.

purchase_item_id: 32-bit field specifying the purchase ID this RO is associated with.

number_of_assets: This field specifies the number of assets (see below) in this BCRO. Each asset listed in this BCRO has an internal id which is equal to the index of the asset in this BCRO. In other words the first asset listed in this BCRO has the internal asset id (index) of 0, the second of 1 etc. This internal id or index is used by permissions objects (see below) to identify the assets it addresses.

number_of_permissions: This field specifies the number of permissions (see below) in this BCRO.

MAC: This is the authentication code calculated over all bytes before this field in the BCRO using HMAC-SHA-1-96 (see [23]). The MAC is used for integrity check of the BCRO. The key used to create the MAC is the BCRO authentication key BAK as defined in clause B.14.3.

B.3.3.4.2.1 Format of the asset object

Table B.12: asset format

Field	Length	Type
asset() {		
BCI	96	bslbf
key_flag	1	
key_type	1	uimsbf
reserved_for_future_use	2	uimsbf
inherit_flag	1	uimsbf
asset_type	2	uimsbf
permissions_category_flag	1	uimsbf
if(inherit_flag == 1){		
purchase_item_id	32	uimsbf
reserved_for_future_use	1	uimsbf
rekeying_period_number	7	uimsbf
}		
if(permissions_category_flag == 1){		
permissions_category	8	uimsbf
}		
if(key_flag == 1){		
if(asset_type == 0){		
if(key_type == 0){		
encrypted_service_encryption_authentication_key	256	bslbf
}else if (key_type == 1){		
encrypted_program_encryption_authentication_key	256	bslbf
}		
}else if(asset_type == 0x1){		
encrypted_content_encryption_key	128	bslbf
}		
}		
}		

BCI: This 96-bit field is the Binary Content ID. See clause B.10. A BCRO can contain multiple assets with the same BCI but with a different permissions_category. Only one asset has to carry key material.

key_flag: 1-bit flag indicating that the asset does contain key material.

key_type: 1-bit flag indicating the type of the key material. If set to 0 the key material contains a service encryption key (SEK), when set to 1 it contains a program encryption key (PEK).

inherit_flag: 1-bit flag indicating whether inheritance is used. If set to 1 the asset inherits the rights setting from a parent rights object.

asset_type: 2-bit flag indicating the asset type as defined in table B.13. If the asset_type is set to 0 the asset MAY contain either a PEK or a SEK. If the asset_type is set to 0x1 then the asset MAY contain a CEK.

Table B.13: asset_type

asset_type	Description
0x0	Broadcast stream protected IPsec, SRTP or ISMACryp as defined in the present document
0x1	Downloaded file content as defined by OMA
0x2 to 0x3	Reserved

permissions_category_flag: 1-bit flag indicating that a permissions_category field is present in this asset object.

purchase_item_id: 32-bit id specifying the purchase ID of the parent rights object.

rekeying_period_number: 7-bit field specifying the rekeying_period_number of the parent rights object. The purchase_item_id and rekeying_period_number are used together with the socID and deviceID or domainID to uniquely identify the parent rights object.

permissions_category: For programme assets, the value of this field (if present) is always zero. For service assets, the following rule applies. If the value of this field is nonzero, it indicates that the permissions (see below) linked to this asset are only to be applied for streaming content whose KSM contains the same value in its permissions_category field. If the value of this field is zero, it indicates that the permissions (see below) linked to this asset are only to be applied for streaming content whose KSM contains the value zero in its permissions_category field, or has value zero for its permissions_flag bit (indicating that there is no permissions_category field in the KSM). Note that there MAY be multiple assets with the same Service_BCI, in which case typically only one of them contains authentication and/or encryption keys in it asset object(s). KSM permissions_category field value thus selects the one with the permissions to be applied among the service assets with the same Service_BCI. The one with the authentication and/or encryption keys is found among the BCROs via inheritance, or by lookup for a BCRO with key material in its assets.

encrypted_service_encryption_authentication_key: If key_type is set to 0 than this field contains the encrypted SEAK, the service encryption key (SEK) concatenated with the Service Authentication Seed (SAS). The field itself is protected using AES-128-CBC, with fixed IV 0 and with 0 padding in the last block if needed. The key used to decrypt this field depends on the addressing mode of the BCRO.

Table B.14: Mapping of address_mode to keys

address_mode	Keys used
0x0 (unique group)	UGK (Unique Group Key)
0x1 (subscriber group)	Deduced SEK decryption key (based on bit_access_mask and zero message subscriber group keys)
0x2 or 0x3 (unique device)	UDK (Unique Device Key)
0x4 (OMA domain)	BDK (Broadcast Domain Key)

encrypted_program_encryption_authentication_key: If key_type is set to 1 than this field contains the encrypted PEAK, the program encryption key (PEK) concatenated with the program authentication seed (PAS). The field itself is protected using AES-128-CBC, with fixed IV 0 and with 0 padding in the last block if needed. The key used to decrypt this field is depending on the addressing mode of the BCRO.

Table B.15: Mapping of address_mode to keys

address_mode	Key(s) used to decrypt field
0x0 (unique group)	UGK (Unique Group Key)
0x1 (subscriber group)	Deduced PEK decryption key (based on bit_access_mask and zero message subscriber group keys)
0x2 or 0x3 (unique device)	UDK (Unique Device Key)
0x4 (OMA domain)	BDK (Broadcast Domain Key)

encrypted_content_encryption_key: This field contains the encrypted content encryption key (CEK). The field is protected using AES-128-CBC, with fixed IV 0 and with 0 padding in the last block if needed. The key used to decrypt this field is depending on the addressing mode of the BCRO.

Table B.16: Mapping of address_mode to keys

address_mode	Key(s) used to decrypt field
0x0 (unique group)	UGK (Unique Group Key)
0x1 (subscriber group)	Deduced CEK decryption key (based on bit_access_mask and zero message subscriber group keys)
0x2 or 0x3 (unique device)	UDK (Unique Device Key)
0x4 (OMA domain)	BDK (Broadcast Domain Key)

B.3.3.4.2.2 Format of the permission object

Table B.17: permission format

Field	Length	Type
`permission() {`		
`    constraint_flag`	1	uimsbf
`    actions_flag`	1	uimsbf
`    number_of_assets`	6	uimsbf
`    for(i = 0;i<number_of_assets;i++){`		
`        asset_index`	8	uimsbf
`    }`		
`    if(constraint_flag == 1){`		
`        constraint()`		
`    }`		
`    if(actions_flag == 1){`		
`        number_of_actions`	8	uimsbf
`        for(i=0;I<number_of_actions;i++){`		
`            action()`		
`        }`		
`    }`		
`}`		

constraint_flag: 1-bit flag which indicates when set to 1 that a constraint object is present in this permissions object. The constraint object applies to all action listed in this permission object.

actions_flag: 1-bit flag. When set to 1, 1 or more actions are contained in this permission object.

number_of_assets: The number of assets this permission object links to. Assets linked to by this permission object are bound by this permission object.

asset_index: A list of number_of_assets links to assets in this BCRO. Assets are linked to by using the internal asset id (the index of the asset in this BCRO).

number_of_actions: Field specifying the number of actions (see below) contained in this permission object.

B.3.3.4.2.3 Format of the action object

Table B.18: action format

Field	Length	Type
`action() {`		
`    action_type`	7	uimsbf
`    constraint_flag`	1	uimsbf
`    if(constraint_flag == 1){`		
`        constraint()`		
`    }`		
`}`		

action_type: 7-bit field specifying the type of action as listed in table B.19.

Table B.19: action_type

action_type	Description
0x00	PLAY_ACTION
0x01	DISPLAY_ACTION
0x02	EXECUTE_ACTION
0x03	PRINT_ACTION
0x04	EXPORT_ACTION
0x05	ACCESS_ACTION
0x06 to 0x7F	reserved for future use

constraint_flag: 1-bit flag which indicates when set to 1 that a constraint object is present in this action object. The constraint object only applies to the action it is in.

B.3.3.4.2.3.1 Action definitions

The action_types 0x00 (PLAY_ACTION), 0x01 (DISPLAY_ACTION), 0x02 (EXECUTE_ACTION), 0x03 (PRINT_ACTION) and 0x04 (EXPORT_ACTION) and their semantics are defined in [50].

The action_type 0x05 (ACCESS_ACTION) and its semantics is defined below.

The ACCESS_ACTION grants the permission to create a transient representation of audio or video content directly from a broadcast stream during its reception. It contains an optional <constraint> object. If the constraint_descriptor is specified the device MUST grant access rights according to the constraint_descriptor child object. If no constraint_descriptor is specified, the device MUST grant unlimited access rights.

A system_constraint object contained in an ACCESS_ACTION object is used to specify target system that may be used for creating a transient rendering of the broadcast stream.

The ACCESS_ACTION has the semantics of rendering the broadcast stream into transient audio/video form, for example, audio/midi, video/quicktime. The device MUST NOT grant access according to an ACCESS_ACTION to content that cannot be rendered in this way.

Note that the device MUST NOT grant access to stored content, not even stored broadcast streams, based on the ACCESS_ACTION. In order to specify rights for stored content, the PLAY_ACTION MUST be utilized instead.

B.3.3.4.2.4 Format of the constraint object

Table B.20: constraint format

Field	Length	Type
`constraint() {`		
`    number_of_constraints`	4	uimsbf
`    constraints_descriptor_length`	12	uimsbf
`    for(i=0;i<number_of_constraint;i++){`		
`        constraint_descriptor()`		
`    }`		
`}`		

number_of_constraints: 4-bit number specifying the number of constraint descriptors (see below).

constraints_descriptor_length: length of all constraint descriptors in bytes which follow this field.

Table B.21: constraint_descriptor format

Field	Length	Type
`constraint_descriptor() {`		
`    constraint_tag`	8	uimsbf
`    length`	8	uimsbf
`    for(i=0;i<length;i++){`		
`        byte`	8	uimsbf
`    }`		
`}`		

constraint_tag: Tag identifying the specific constraint_descriptor as listed in table B.22.

Table B.22: constraint_tag

constraint_tag	Description
0x00	count constraint
0x01	timed-count constraint
0x02	date time constraint
0x03	interval constraint
0x04	accumulated constraint
0x05	individual constraint
0x06	system constraint
0x07	metering constraint
0x08 to 0xFF	reserved for future use

B.3.3.4.2.4.1 Count constraint descriptor

Table B.23: count_constraint_descriptor

Field	Length	Type
`count_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`count`	8 × length	uimsbf
`}`		

length: The number of bytes used for the count field. Length SHALL NOT exceed 4, hence the maximum size of the count field can be 32 bits.

count: The number of times the content can be played. The field can be of size 8, 16, 24 and 32 bits. See [50].

B.3.3.4.2.4.2 Timed count constraint descriptor

Table B.24: timed_count_constraint_descriptor

Field	Length	Type
`timed_count_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`timer`	16	uimsbf
`count`	8 × (length - 2)	uimsbf
`}`		

length: The number of bytes following this field. The count field is length-2 bytes long and SALL NOT exceed 32 bits.

timer: Specifies the number of seconds after which the count state is reduced starting from beginning to render the content.

count: The number of times the content can be played. The field can be of size 8, 16, 24 and 32 bits. See [50].

B.3.3.4.2.4.3 Date-time constraint descriptor

Table B.25: datetime_constraint_descriptor

Field	Length	Type
`datetime_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`start_flag`	1	bslbf
`end_flag`	1	bslbf
`reserved_for_future_use`	6	bslbf
`if(start_flag == 1){`		
`start_time`	40	mjdutc
`}`		
`if(end_flag == 1){`		
`end_time`	40	mjdutc
`}`		
`}`		

length: The number of bytes of the descriptor immediately following this field.

start_flag: 1-bit field. When set the descriptor contains a start time.

end_flag: 1-bit field. When set the descriptor contains an end time.

start_time: Time field with the semantics of "not before" time for a permission. The start_time must be before the end_time if present. See [50].

end_time: Time field with the semantics of "not after" time for a permission. The end_time must be after the start_time if present. See [50].

B.3.3.4.2.4.4 Interval constraint descriptor

Table B.26: interval_constraint_descriptor

Field	length	type
`interval_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`time_interval`	8 × length	uimsbf
`}`		

length: The number of bytes following this field. Length specifies the size of the time_interval field.

time_interval: Specifies the number of seconds starting from first receiving this BCRO that the permission is valid. The length of the field is given by the length field and SHALL NOT exceed 32 bit. See [50] for a more detailed description.

B.3.3.4.2.4.5 Accumulated constraint descriptor

The accumulated_constraint_descriptor specifies the maximum period of metered usage time during which the rights can be exercised over the DRM content.

Table B.27: accumulated_constraint_descriptor

Field	length	type
`accumulated_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`accumulated_time`	8 × length	uimsbf
`}`		

length: The number of bytes following this field. Length specifies the size of the accumulated_time field.

accumulated_time: Specifies the maximum period of metered usage time during which the rights can be exercised. The period is given in seconds. The length of the field is given by the length field and SHALL NOT exceed 32 bit. See [50] for a more detailed description.

B.3.3.4.2.4.6 Individual constraint descriptor

Constraint used to bind content to individuals. If the content should be bound to more than one individual multiple individual_constraint_descriptor(s) can be carried in one constraint object.

Table B.28: individual_constraint_descriptor

Field	Length	Type
`individual_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`reserved_for_future_use`	4	bslbf
`id_type`	4	uimsbf
`individual_id`	(length - 1) × 8	bslbf
`}`		

length: The number of bytes following this field. Length-1 specifies the size of the individual_id field.

id_type: Tag identifying format of the individual_id as listed in table 29.

Table B.29: id_type

id_type	Description
0x0	The individual_id field contains the IMSI number coded as 16 digit 4-bit BCD. The first digit SHALL be 0 and SHALL be ignored. The length of the individual_id field is 64 bit.
0x1	The individual_id field contains the PKC id of the WIM to which the content is bound.
0x2 to 0xF	reserved for future use.

individual_id: Individual ID. The format and length of this field is identified by the identifier_type and length field see table 29. See [50] for a more detailed description.

B.3.3.4.2.4.7 System constraint descriptor

Constraint used identify systems to which the content and rights objects are allowed to be exported to.

Table B.30: system_constraint_descriptor

Field	Length	Type
`system_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`system_id`	64	bslbf
`parameterbytes`	8 × length - 64	bslbf
`}`		

length: The number of bytes following this field.

system_id: The system id of the system the content and RO can be exported to. This is the HMAC-SHA-1-64 encoded hash of the system name as registered with OMNA. See [50] for a more detailed description.

parameterbytes: This is a string of bytes, containing parameters for the system. This may e.g. be required when exporting to a (possibly non-DRM) system and requiring that no more copies are to be made. The format of parameters is system specific and out of scope of this specification.

B.3.3.4.2.4.8 Metering constraint descriptor

The metering_constraint_descriptor specifies that the consumption of the DRM content involves the consumption of tokens. The device can receive tokens from each Rights Issuer and store them per Rights Issuer in a token store. The parameters in the metering_constraint_descriptor indicate how 'much' consumption of DRM content requires how many tokens need to be consumed from the token store.

Table B.31: metering_constraint_descriptor

Field	Length	Type
`metering_constraint_descriptor() {`		
`constraint_tag`	8	uimsbf
`length`	8	uimsbf
`token_constraint_type`	2	uimsbf
`token_unit_length`	3	uimsbf
`token_consumed_length`	3	uimsbf
`token_unit`	8 × token_unit_length	uimsbf
`for(i=0;i<token_consumed_length;i++){`		
`token_consumed`	8 × token_consumed_length	uimsbf
`}`		

length: The number of bytes following this field.

token_constraint_type: If the value of this field equals 0x0 (COUNT), the consumption of the DRM content must be counted and any consumption of the DRM content equalling the number of 'counts' as indicated by the token_unit field requires the consumption of the amount of tokens as indicated by the value of the token_consumed field.

If the value of this field equals 0x1 (DURATION), any consumption of the DRM content with a duration of the number of seconds as indicated by the token_unit field requires the consumption of the amount of tokens as indicated by the value of the token_consumed field.

All other values of this field are reserved for future use.

token_unit_length: Field defining the length in bytes of the token_unit field. The value SHALL NOT be bigger than 4.

token_consumed_length: Field defining the length in bytes of the token_consumed field. The value SHALL NOT be bigger than 4.

token_unit: If the token_constraint_type field equals 0x00 (COUNT), the token_unit indicates the amount of 'counts' of consumption of the DRM content that can be consumed for the amount of tokens as indicated in the token_consumed field.

If the token_constraint_type field equals 0x01 (DURATION), the token_unit indicates the number of seconds of consumption of the DRM content that can be consumed for the amount of tokens as indicated in the token_consumed field.

token_consumed: This field indicates the amount of tokens that must be consumed from the token store of the device if the amount of DRM content is consumed as indicated by the token_constraint_type field and the token_unit field.

B.3.4 Registration Layer

All devices SHALL implement the registration layer as specified in this clause. Devices MAY additionally implement alternative schemes for the functionality of the rights management and registration layers, see clause B.2.7 for details.

B.3.4.1 RI Context

There are two types of RI context, being:

- RI context for interactive mode of operation. This is specified in [49], whereas some details are listed in clause B.3.4.2.
- RI context for broadcast mode of operation. This is specified in clause B.3.4.3.

Please note that both types of RI context are different from each other.

B.3.4.2 Interactive mode of operation

Registration is according to [49] as described in clause B.2.3.2.

B.3.4.3 Broadcast mode of operation

B.3.4.3.1 Protocol overview

The theory of operation (refer to clause B.2.3) results in the specification of several protocols:

- offline protocols (from device to RI).

Table B.32

Protocol	Clause	Purpose
Offline Notification of Detailed Devicedata protocol	B.3.4.3.2	Registration
Offline Notification of Short Devicedata protocol	B.3.4.3.3	Inform RI by action request codes

- 1-pass protocols (from RI to device).

Table B.33

Protocol	Clause	Purpose
1-pass binary Push Device Registration protocol	B.3.4.3.4	Transmit registration data to device
1-pass binary Inform Registered Device protocol	B.3.4.3.5	Inform device via messages

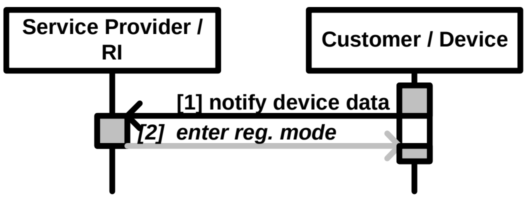
- supporting protocols for registration.

Table B.34

Protocol	Clause	Purpose
Unique Device Number (UDN) protocol	B.3.4.3.6	Make registration data robust (part of offline notification of detailed device data)

B.3.4.3.2 Offline Notification of Detailed Devicedata protocol

NOTE: This protocol is also known as the 'offline NDD protocol', short for offline Notification of Detailed Data protocol.



NOTE: Notification of device data is performed off-line. The device data (device_data_inform() message) is defined in clause B.3.4.3.7.

Figure B.27: offline NDD protocol

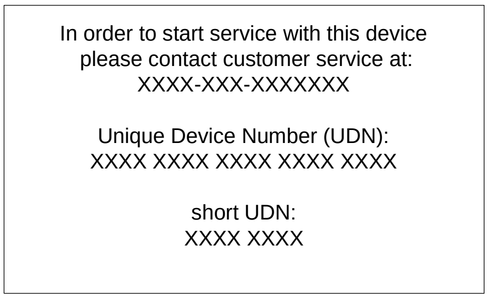
Explanation of the protocol:

- The device SHALL notify (1) its device data via some means to the RI. After user interaction the device SHALL produce the device_data_inform() message (refer to clause B.3.4.3.7.1 for details) and make this data available to the user.

- The device MAY display a dialogue with instructions. Notifying the device data can be done in various ways, for example by showing the user of the device a dialogue on the screen of the device, displaying the device data and a telephone number to call for vocal notification of the device data. Another example is to display instructions to send an SMS message via a mobile phone to the RI, or else.

An example of a displayed message follows, where the following information is reported back to the RI.

NOTE: It is the sequence of the defined values that is specified. The use of dashes as the delimiter is shown with an example placement to be consistent with the examples used elsewhere in this specification. The text portion of this screen is shown as an example only; there is no implied requirement to duplicate the exact wording or formatting shown. The numeric fields SHALL be included as defined above (please note: the short UDN will only be displayed after the first registration, when that data MAY available for display).

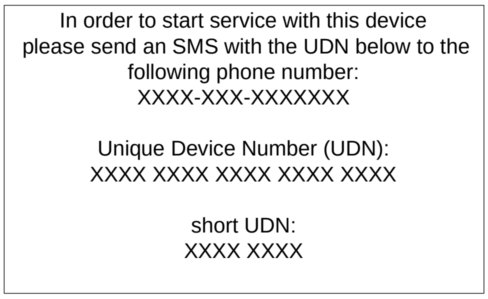


In order to start service with this device please contact customer service at:
XXXX-XXX-XXXXXXX

Unique Device Number (UDN):
XXXX XXXX XXXX XXXX XXXX

short UDN:
XXXX XXXX

An example dialogue showing instructions for vocal notification of UDN to callcenter



In order to start service with this device please send an SMS with the UDN below to the following phone number:
XXXX-XXX-XXXXXXX

Unique Device Number (UDN):
XXXX XXXX XXXX XXXX XXXX

short UDN:
XXXX XXXX

An example dialogue showing instructions for notification of UDN per SMS to callcenter

Figure B.28: Samples of notification displays

- If the device does not support a return channel to the RI, the device data (device_data_inform() message) SHALL be notified off-line, using the offline Notification of Detailed Devicedata protocol. The device data to notify SHALL be reduced by a special protocol (refer to clause B.3.4.3.6).

- After the notification of the device data, user needs to put the device into registration mode (2). When put into registration mode, device SHALL start to listen for the device registration data for a limited time.

B.3.4.3.3 offline Notification of Short Devicedata protocol

NOTE 1: This protocol is also known as the 'offline NSD protocol', short for offline Notification of Short Data protocol.

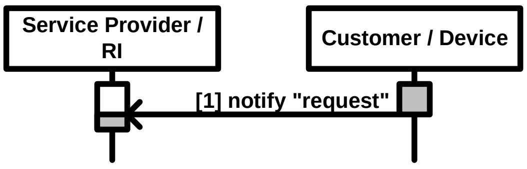


Figure B.29: Offline NSD protocol

NOTE 2: Notification of device data is performed off-line. Refer to table B.35b for an overview of the possible 'requests'.

Explanation of the protocol:

- The user may notify a short decimal code called the action request code (ARC) to the RI via offline methods (e.g. telephone call or SMS or else). The code SHALL be constructed as follows.

Short_udn	Action_code	Checksum

Figure B.30: Action Request Code (ARC)

Note that for some of the ARCs (e.g. the ARC token_consumption_report), the user MAY have to notify more digits to the RI than the ones of the ARC.

Table B.35a: NSD action request code fields

ARC fields	Length (digits)	Supporting up to
short_udn	8	100 million devices
action_code	2	99 action codes
checksum	2	

This totals to 12 digits. The fields are explained below:

***short_udn*:** The offline notification can be performed faster if the long form UDN is not used, but a shorter form instead. After first time notification of the device data to the RI, the RI MAY issue a short version of the full UDN (called short_form_udn) that is carried in the device_registration_response() message. The short_form_udn number is used to speed up the offline interaction with the RI. If this number is stored into the device, subsequent 'requests' by the user of the device can be notified offline much quicker by using the short_form_udn number concatenated by a standardized action code.

Please note: In cases where the device needs to be identified uniquely in another network than its home network where it was registered, the short_udn cannot be used because the (new / different) RI does not have the short_udn in its database. In this case the only possibility for the hosting RI to identify the device uniquely would be via the long_udn. It is the responsibility of the device to decide when it is appropriate to use the long_udn instead, for example by comparing the Service Operations Centre (SOC) ID received with the SOC ID remembered from registration.

***action_code*:** Following the short_udn the user of the device can notify an action code to the RI. The NSD protocol defined in the present document SHALL use following action_codes to construct the ARC:

Table B.35b: NSD action types

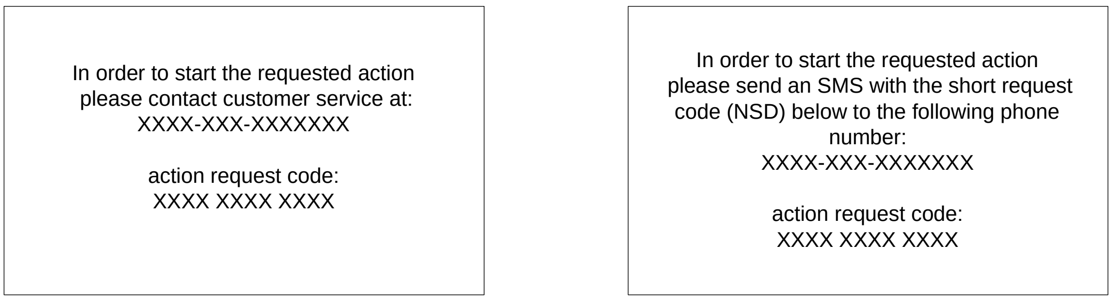
Action type	Action code (d)	Described in clause
re-registration (only at same RI)	{0d01}	B.3.4.3.3.1
resend BCRO	{0d02}	B.4.9
reserved for future use	{0d03,..,0d09}	
join domain	{0d10,..,0d19}	B.3.4.3.3.2
leave domain	{0d20,..,0d29}	B.3.4.3.3.3
purchase	{0d30}, whereas content identification is supplied by ESG.	B.5.3.2
token_consumption_report	{0d31,..,0d39}	B.3.4.3.3.4
reserved for future use	{0d40,..,0d49}	
token_request	{0d50,..,0d59}	B.3.4.3.3.6
notify DRM time drift	{{0d7}+{0d0,..,0d9},..,{0d8}+{0d0,..,0d9}}	B.3.4.3.3.
reserved for future use	{0d90,..,0d99}	

***Checksum*:** The constructed short_udn and action_code is appended by checksum digits. Please refer to clause B.15 for an explanation of the algorithm.

An example: In order to request to re-register, a sample NSD action request code could look like:

'1660 8731 0112'. An example of a displayed message follows, where the following information is reported back to the RI.

NOTE: It is the sequence of the defined values that is specified. The use of dashes as the delimiter is shown with an example placement to be consistent with the examples used elsewhere in this specification. The text portion of this screen is shown as an example only; there is no implied requirement to duplicate the exact wording or formatting shown. The numeric fields SHALL be included as defined above (please note: the short UDN will only be displayed after the first registration, when that data MAY available for display.



In order to start the requested action
please contact customer service at:
XXXX-XXX-XXXXXXX

action request code:
XXXX XXXX XXXX

An example dialogue showing instructions for vocal notification of ARC to callcenter

In order to start the requested action
please send an SMS with the short request code (NSD) below to the following phone number:
XXXX-XXX-XXXXXXX

action request code:
XXXX XXXX XXXX

An example dialogue showing instructions for notification of ARC per SMS to callcenter

Figure B.31: samples of notification displays showing an ARC message

B.3.4.3.3.1 Request re-registration (only at same RI)

After sending this ARC the user will wait until he receives the confirmation of the RI in the form of a device_registration_response() message. (refer to clause B.3.4.3.4).

B.3.4.3.3.2 Request join domain

The Action Request Code (ARC) for the NSD protocol is formed according to following rules:

- The first digit is used to notify the join domain action.
- The second digit is used as a device_nonce to help the device to keep track of join domain requests.

After notifying the ARC to the RI the user MAY notify a particular domain group number identifying a domain where the device is to be entered. The RI SHALL incorporate the device_nonce from the request in the response message.

B.3.4.3.3.3 Request leave domain

The Action Request Code (ARC) for the NSD protocol is formed according to following rules:

- The first digit is used to notify the join domain action.
- The second digit is used as a device_nonce to help the device to keep track of leave domain requests.

After notifying the ARC to the RI, the user needs to notify a particular domain group number identifying a domain where the device is to be removed from. The device SHALL display a domain ID. The RI SHALL incorporate the device_nonce from the request in the response message.

B.3.4.3.3.4 Token consumption report

The Action Request Code (ARC) for the NSD protocol is formed according to following rules:

- The first digit is used to notify the token consumption report.
- The second digit is used as a device_nonce to help the device to keep track of token consumption reports.

After notifying the ARC to the RI the user should notify the token consumption data. The device SHALL display the token consumption data e.g. to the left of or below the digits of the ARC for the token consumption report. The RI SHALL incorporate the device_nonce from the request in the response message.

An example of a displayed message follows, where the following information is reported back to the RI.

NOTE: It is the sequence of the defined values that is specified. The use of dashes as the delimiter is shown with an example placement to be consistent with the examples used elsewhere in this specification. The text portion of this screen is shown as an example only; there is no implied requirement to duplicate the exact wording or formatting shown. The numeric fields MUST be included as defined above (please note: the short UDN will only be displayed after the first registration, when that data MAY available for display.

```
 In order to start the requested action
  please contact customer service at:
          XXXX-XXX-XXXXXXX

         action request code:
           XXXX XXXX XXXX

        token consumption data:
      XXXX XXXX XXXX XXXX XXXX
```

Figure B.32: Sample of token consumption reporting notification display

B.3.4.3.3.4.1 Token consumption data definition

The token consumption data are defined below.

Table B.36: token consumption data

Field	Length (digits)	Supporting up to
tokens_consumed	4	9 999 tokens to be reported
report_authentication_code	13	
checksum	3	

This totals to 20 digits. The fields are explained below:

tokens_consumed: This field contains the amount of tokens the device wished to report as consumed to the RI. See clause B.21 for more information.

report_authentication_code: This field contains the authentication code for the value in the tokens_consumed field and the value of the device_nonce (second digit of the action_code of the ARC of this message). See clause B.20 for the computation of the report authentication code.

Checksum: The final digits of the device ID number are check digits, akin to a checksum. The 3 digits allow 1 out of 10^3 possible errors to remain undetected. The checksum algorithm used is the UDN checksum, see clause B.15.1.

B.3.4.3.3.5 Notify DRM time drift

Time drift is expressed in minutes and rounded up to next multiple of 5 minutes. The range is 0..100 minutes, whereas value 69 will decode as timedrift >= 100. Some examples of valid ARC values are given below:

E.g.1: Device notifies 4 minutes timedrift from newly received DRM time message: action code is 70.

E.g.2: Device notifies 38 minutes timedrift from newly received DRM time message: action code is 78.

E.g.3: Device notifies 235 minutes timedrift from newly received DRM time message: action code is 89.

B.3.4.3.3.6 Token request

The Action Request Code (ARC) for the NSD protocol is formed according to following rules:

- the first digit is used to notify the token request;
- the second digit is used as a device_nonce to help the device to keep track of token requests.

After notifying the ARC to the RI the user SHOULD notify the number of tokens desired. The RI SHALL incorporate the device_nonce from the request in the response message.

B.3.4.3.4 1-pass binary Push Device Registration Protocol

NOTE 1: This protocol is also known as the '1-pass PDR protocol', short for Push Device Registration protocol.

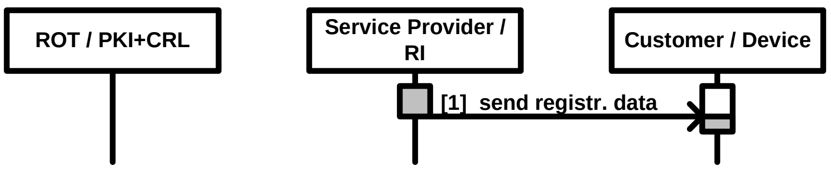


Figure B.33: 1-pass PDR protocol - (first) device registration

NOTE 2: Transmission of registration data is performed on-line via the broadcast channel. The registration data (device_registration_response() message) is specified in clause B.3.4.3.7.2.

Explanation of the protocol:

- The RI SHALL use the 1-pass binary Push Device Registration data (a.k.a. PDR) protocol to send registration data over the network (1). The registration data can be the device_registration_response() message (refer to clause B.3.4.3.7.2) or the domain_registration_response() message (refer to clause B.3.4.4.4.1). The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device. The RI SHALL include a valid keyset in the message.
- A device listening for device_registration_response() (or domain_registration_response()) messages SHALL look for messages with the corresponding message_tag. On every message with a matching message_tag the device SHALL check the long_form_udn parameter. If this matches (any of) the devices local UDN(s) the device SHALL start validating the signature and check the RI certificate (chain.). If both (UDN and signature) are valid the device detects this message is really addressed to it. The device SHALL start processing the message and SHALL start trying to decrypt the secret data in it. If the message is correct, the device SHALL store the new keyset with key(s). The devise SHALL delete the old keyset (if applicable).
- After a timeout the device SHALL leave the registration mode and stops listening for device_registration_response() messages.

B.3.4.3.5 1-pass binary Inform Registered Device Protocol

NOTE: This protocol is also known as the '1-pass IRD protocol', short for Inform Registered Device protocol.

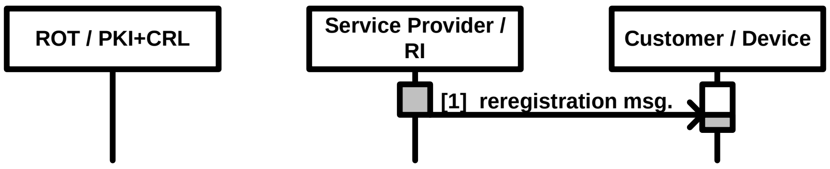


Figure B.34: 1-pass IRD protocol - RI initiated message to device (here re-registration).

Explanation of the protocol:

- The 1-pass IRD protocol is designed to meet the messaging push case. Its successful execution assumes the device to have an existing RI context with the sending RI.
- Several messages are defined for the IRD protocol.

Table B.37: Messages of the 1-pass IRD protocol

Message name	For message syntax refer to clause	Remark
force to re-register	B.3.4.3.7.3	
update RI certificate	B.3.4.3.7.4	in BCRO carousel
update DRM Time	B.3.4.3.7.5	in BCRO carousel
update contact number	B.3.4.3.7.6	in BCRO carousel
update domain	B.3.4.4.4.2	
force to join domain	B.3.4.4.4.3	
force to leave domain	B.3.4.4.4.4	
token delivery	B.3.4.5.4.1	

NOTE 2: The processing of each message will be discussed in following clauses.

B.3.4.3.5.1 Force re-registration

In this case the RI is sending a message to the device to get it into registration mode.

- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- The device SHALL filter on the message_tag to identify the message. Then it SHALL filter for the UDN and compare it to the local UDN of the device. If those match the device SHALL start validating the signature and check the RI certificate (chain.). If both (UDN and signature) are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.

- If the message is correct, the reception of this message SHALL start the (re-) registration process. The device will be rendered inoperable, but only in relation with the associated RI (context) as described below:
    - Accessing an ESG for purchase is still allowed, as this will require a registration first.
    - The device SHALL be rendered inoperable for any purchase protocol or playback of future content. The device MAY use stored BCROs to play old content for which the device obtained ROs, but SHALL NOT use these BCROs for new content received after the re-registration request until the device is re-registered with the RI.
- Depending on the implementation a dialogue will be shown to the user and the offline NDD protocol will be executed, using the RI_ID stored in the RI Context. Depending on the implementation a dialogue MAY be shown to the user and the offline NDD protocol SHALL be executed.

B.3.4.3.5.2 Update RI certificate

The RI can use this message to update the RI certificate in one or more devices.

- The RI SHALL enter a valid RI certificate in the message.
- The RI MAY enter a rooted RI certificate chain in the message. The root certificate is to be excluded.
- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- The device SHALL filter on the message_tag to identify the message. Then it SHALL filter for the UDN and compare it to the local UDN of the device. If those match the device SHALL start validating the signature and check the RI certificate (chain.). If both are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.
- If the message is correct, the device SHALL save the new RI certificate in the message after the signature of the message has been verified correctly. The old RI certificate SHALL be made obsolete.

B.3.4.3.5.3 Update DRM_Time

The RI can use this message to update the DRM time.

- The RI SHALL enter a valid DRM time in the message.
- The RI MAY put a time offset in the message. The time offset SHALL be valid.
- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- The device SHALL filter on the message_tag to identify the message. Then the device SHALL start validating the signature and check the RI certificate (chain.). If both are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.
- If the message successfully validated and the RI certificate is valid, the device SHALL save the new DRM time into the device.

B.3.4.3.5.4 Update contact number

The RI can use this message to update the contact number that the device should contact during the offline notification processes (both for use with the NDD or NSD protocols).

- The message SHALL contain (a) valid telephone number(s) to contact.
- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- The device SHALL filter on the message_tag to identify the message. Then the device SHALL start validating the signature and check the RI certificate (chain.). If both are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.
- If the message is correct, the device SHALL store the new contact number(s) and delete the old one(s).

B.3.4.3.5.5 Force to join a domain

In this case the RI is sending a message to the device to get it into join domain mode, which MAY be followed up by the matching action in the NSD protocol.

- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- A device listening for device_registration_response() messages SHALL look for messages with the corresponding message_tag. On every message with a matching message_tag the device SHALL check the long_form_udn parameter. If this matches (any of) the devices local UDN(s) the device SHALL start validating the signature and check the RI certificate (chain.). If both (UDN and signature) are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.

B.3.4.3.5.6 Force to leave a domain

In this case the RI is sending a message to the device to get it into leave domain mode, which MAY be followed up by the matching action in the NSD protocol.

- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- A device listening for device_registration_response() messages SHALL look for messages with the corresponding message_tag. On every message with a matching message_tag the device SHALL check the long_form_udn parameter. If this matches (any of) the devices local UDN(s) the device SHALL start validating the signature and check the RI certificate (chain.). If both (UDN and signature) are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.

B.3.4.3.5.7 update a domain

The RI can use this message to inform the device that he left a particular domain.

- The message SHALL contain a valid domain id.
- The RI SHALL use the mechanisms described in clause B.4.6 to address the message to a device.
- A device listening for device_registration_response() messages SHALL look for messages with the corresponding message_tag. On every message with a matching message_tag the device SHALL check the long_form_udn parameter. If this matches (any of) the devices local UDN(s) the device SHALL start validating the signature and check the RI certificate (chain.). If both (UDN and signature) are valid the device detects this message is really addressed to it, and the device SHALL start to perform the intended action.
- If the message is correct, the device SHALL delete the associated domain context.

B.3.4.3.6 Unique Device Number (UDN) protocol

To reduce the amount of data that is to be notified to the RI, the device data protocol takes care of data reduction. To ease the detection of errors during the registration process, the device data protocol will also allow detection of errors in the notified device data.

Following algorithm SHALL be used to construct a Unique Device Number (a.k.a. UDN).

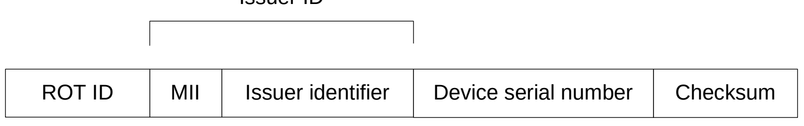


Figure B.35: Unique Device Number

Table B.38: UDN explanation

Field	Length (digits)	supporting up to
rot_id	3	1 000 ROT
mii	1	9 Major Industries
issuer_identifier	4	100 000 Issuers (10 000 in 10 industries)
device_serial_number	9	1 Billion
checksum	3	

This totals to 20 digits. The fields are explained below:

Every of 1 000 ROT can issue 100 000 issuer ranges, from which every unique issuer can have 1 Billion devices issued.

rot_id: The first 3 digits in the UDN identify the ROT. Every ROT has an own unique ID.

mi:i: The first digit of the device ID number is the Major Industry Identifier (MII), which represents the category of entity, which issued the device_serial_number. Different MII digits represent the following issuer categories:

Table B.39: major industry identifier

MII Digit Value	Issuer Category	Remarks
0	Root of Trust	
1	Telecom	
2	Consumer Electronics	
3	Network Equipment	
4	Reserved for future use	
5	Reserved for future use	
6	Reserved for future use	
7	Reserved for future use	
8	Reserved for future use	
9	National Assignment	

For example: Philips is in the Consumer Electronics Category and Nokia is in the Telecom sector. If the MII digit is 9, then the next three digits of the issuer identifier are the 3-digit country codes defined in [19], and the remaining final two digits of the issuer identifier can be defined by the national standards body of the specified country in whatever way it wishes.

issuer_identifier: The issuer_identifier identifies which issuer created the devices serial number. Together with the MII digit this forms the issuer ID.

device_serial_number: The device_serial_number is unique inside a range of an issuer. Each issuer therefore has 1 billion (10^9) possible device_serial_numbers. It is unlikely that an issuer exceeds 1 billion serial numbers. An issuer whishing to group devices in another way can request a second issuer_identifier for another range (of 1 Billion).

Checksum: The final digits of the device ID number are check digits, akin to a checksum. The 3 digits allow 1 out of 10^3 possible errors to remain undetected. Please refer to clause B.15 for an explanation of the algorithm.

B.3.4.3.6.1 Message syntax

The 20 digits of the UDN are encoded in BCD format into the longform_udn(). The message syntax is specified below.

Table B.40: longform_udn

Fields	Length	Type
`longform_udn(){`		
`rot_id`	12	bslbf
`mii`	4	bslbf
`issuer_identifier`	16	bslbf
`device_serial_number`	36	bslbf
`checksum`	12	bslbf
`}`		

B.3.4.3.7 Binary messages

B.3.4.3.7.1 Device data - device_data_inform() message

B.3.4.3.7.1.1 Message description

The Device data SHALL prove that it is unique. In a one way case the device notifies this device data, yet the length of the unique device data SHOULD remain concise.

Because devices can be uniquely identified by the PKI, it is not needed to incorporate unique data like the device certificate into the (device specific) registration data. The OMA DRM 2.0 certificate is global and the link between the manufacturer and the device can be requested from the PKI, based on the device ID.

Table B.41: Notify device data message parameters

Device_Data_Inform()		
parameter	(M)andatory / (O)ptional (see note)	Remark
version	M	
contact_nr	O	
longform_udn()	M	
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device SHALL support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

version: is a <major> representation of the highest ROAP version number supported by the Device. For this version of the protocol, the *version* field SHALL be set to value '**1**'.

contact_nr: is the number to be contacted in order to register the device. It can be a phone number or an SMS number. This number MAY have been entered into the device at production time and if so MAY be shown in the registration display (refer to clause B.3.4.3.2 for an example). This number could also be provided in other ways, like a leaflet in the package of the device, a commercial channel which is viewed after selection of a free to air channel via the ESG or via entirely other means.

longform_udn(): identifies the unique_device_number to the RI. The UDN SHALL be part of the credentials entered at production time into the device, like the private key and the certificate. Refer to clause B.3.4.3.6 for details.

B.3.4.3.7.1.2 Message syntax

Since this is an offline protocol the device data is not really formed into a message that can be transmitted. The device data is decimal and formatted as follows.

Table B.42: Device data

Parameter	**Format and length**	**Description**
version	1 byte	
contact_number	15 bytes	Dependent on target telco network
longform_udn()	20 bytes	UDN protocol

B.3.4.3.7.2 Registration data - device_registration_response() message

B.3.4.3.7.2.1 Message description

Using the 1-pass PDR protocol the RI SHALL send a device_registration_response() message with the registration data to the device as specified below.

Table B.43: message description

Device_Registration_Response()		
Parameter name	(M)andatory / (O)ptional (see note)	remark
message_tag	M	global, not encrypted
protocol_version	M	global, not encrypted
longform_udn()	M	global, not encrypted
status	M	device specific, not encrypted
certificate_version	M	global, not encrypted
ri_certificate_counter	M	global, not encrypted
c_length	M	global, not encrypted
ri_certificate	M	global, not encrypted
ocsp_response_counter	M	global, not encrypted
r_length	M	global, not encrypted
ocsp_response	M	global, not encrypted
local_time_offset_flag	M	device specific, not encrypted
time_stamp_flag	M	device specific, not encrypted
subscriber_group_key_flag	M	device specific, not encrypted
signature_type_flag	M	global, not encrypted
short_udn_flag	M	device specific, not encrypted
surplus_block_flag	M	device specific, not encrypted
keyset_block_length	M	device specific, not encrypted
unique_group_key	O	device specific, encrypted
subscriber_group_key	O	device specific, encrypted
unique_device_key	O	device specific, encrypted
unique_device_filter	M	device specific, encrypted
ri_authentication_key	M	device specific, encrypted
token_delivery_key	O	device specific, encrypted
broadcast_domain_key	O	device specific, encrypted
shortform_domain_id	M	device specific, encrypted
drm_time	M	device specific, not encrypted
local_time_offset	O	device specific, not encrypted
registration_timestamp_start	O	device specific, not encrypted
registration_timestamp_end	O	device specific, not encrypted
shortform_udn	O	device specific, not encrypted
signature_block	M	device specific, not encrypted
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).
If set to 0x0 the format specified in the present document is used. If set to anything else than 0x0, then the format is beyond the scope of the present document

longform_udn(): The long form of the UDN. Refer to clause B.3.4.3.6 for details.

status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.44: Status values

Status value	Meaning
Success	The registration request was executed successfully and the RI completed all data. The device SHALL process the message.
UnknownError	The RI encountered an unknown error after receiving the registration request. The device MAY put forward a subsequent registration request to the RI (context).
NotSupported	The RI does not support the registration request.
AccessDenied	The RI decided that the device will not be granted access to the service and stops the registration. The RI will stop listening to future registration requests of this device. The device is forced to refrain from future registration and SHALL **suppress** broadcast and/or mixed-mode registration requests to the particular RI (context).
NotFound	The RI decided that the device could not be found (offline UDN and/or UaProf). The device MAY put forward a subsequent registration request to the RI (context).
MalformedRequest	The RI decided that the registration request was malformed and will **force** the device to execute a (re)-registration at once. The device SHALL enter (re)registration mode (refer to clause B.3.4.3.5.1)

Please refer to clause B.7 for the value of the error codes.

***certificate_version*:** is a numerical representation of the version of the RI certificate. Using the certificate_version parameter the device can decide if it is needed to update the RI certificate (if it was stored before).

Table B.45: Description of certificate_version parameter

Parameter Fieldname	Field Value ($_h$)	supports
major_version_number	0x0,..,0xF	MSB_4(certificate_version)
minor_version_number	0x0,..,0xF	LSB_4(certificate_version)

The parameter is divided 2 fields of 4 bits, whereas the first 4 bits (MSB left) express the Major number and the last four bits (LSB right) express the Minor version. The major and minor number encode in bslbf format. 16 Major and 16 Minor versions are supported. For example: Major.Minor version <1.2> is expressed as $0001\ 0010_b$.

ri_certificate_counter: This parameter indicates the depth of the RI certificate chain.

Table B.46

Number of certificate in chain	Value ($_h$)	Remark
0	0x0	Will signal absence of ri_certificate e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 RI certificates.		

c_length: This parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): This parameter SHALL be present. When present, the value of a *ri_certificate* parameter SHALL be a certificate chain including the RI's certificate. The chain SHALL NOT include the root certificate. The RI certificate SHALL come first in the list. Each following certificate SHALL directly certify the one preceding it.

The Device MAY store RI certificate verification data indicating that an RI certificate chain has been verified. The purpose of this is to avoid repeated verification of the same certificate chain. The RI certificate verification data stored in this way SHALL uniquely identify the RI certificate and SHALL be integrity protected. The Device SHOULD check if the RI certificate chain received in this parameter corresponds to the stored certificate verification data for this RI. If so, the Device NEED NOT verify the RI certificate chain again, otherwise the Device SHALL verify the RI certificate chain.

If an RI certificate is received that is not in the stored certificate verification data for this RI, and if the Device can determine (in the case of Broadcast Devices that support DRM Time) that the expiry time of the received RI certificate is later than the RI Context for this RI, and the certificate status of the RI certificate as indicated in the OCSP response is good (see [OCSP-MP]), then the Device SHALL verify the complete chain and SHOULD replace the stored RI certificate verification data with the received RI certificate data and set the RI context expiry time to that of the received RI certificate expiry time.

However, if the Device does store RI certificate verification data in this way it SHALL store the expiry period of the RI's certificate (as indicated by the notAfter field within the certificate) and SHALL compare the Device's current DRM Time with the stored RI certificate expiry time whenever verifying the signature on signed messages from the RI. If the Device's current DRM Time is after the stored RI certificate expiry time then the Device SHALL abandon processing the RI message and SHALL initiate the registration protocol.

***ocsp_response_counter*:** This parameter indicates the depth of the OCSP response chain.

Table B.47

Number of responses in chain	Value ($_h$)	Remark
0	0x0	Will signal absence of ocsp_response e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 OCSP responses.		

r_length: This parameter indicates the length in bytes of the ocsp_response.

ocsp_response(): This parameter, when present, SHALL be a complete set of valid OCSP responses for the RI's certificate chain. The Device SHALL NOT fail due to the presence of more than one OCSP response element. A Device SHALL check that an OCSP response is present in the received message. If no OCSP response is present in the device_registration_response() message, then the Device SHALL abort the registration protocol.

local_time_offset_flag: Binary flag to signal presence of the parameter it describes.

Table B.48

local_time_offset_flag	Value ($_h$)	remark
data absent	0x0	
data present	0x1	

time_stamp_flag: Binary flag to signal presence of the parameter it describes.

Table B.49

time_stamp_flag	Value ($_h$)	remark
data absent	0x0	
data present	0x1	

***subscriber_group_key_flag*:** The flag expresses how many subscriber_group_keys (a.k.a. SGK) are delivered with the registration data. When zero message broadcast is used, a set of 8 keys will support a group size of 256. A set of 9 keys will support a group size of 512. Other values or larger group sizes are not supported. A value larger than zero indicates that the registration data message delivers a set of zero message subscriber_group_key (s) to the device and that the device needs to use zero message broadcast style encryption to deduce the decryption key to decrypt the SEK.

Table B.50

subscriber_group_key_flag	Value ($_h$)	Remark
data absent	0x0	will signal absence of keyset_block e.g. on error status to save bandwidth
reserved for future use	0x1 to 0x7	not used in the present document
set of (8) subscriber_group_key	0x8	
set of (9) subscriber_group_key	0x9	
reserved for future use	0xA to 0xF	not used in the present document

***signature_type_flag*:** A flag to signal type of signature algorithm used.

Table B.51

signature_type_flag	Value ($_h$)	Remark
RSA 1 024	0x0	
RSA 2 048	0x1	
RSA 4 096	0x2	
reserved for future use	0x3	not used in the present document

short_udn_flag: Binary flag to signal presence of the parameter it describes.

Table B.52

short_udn_flag	Value ($_h$)	Remark
data absent	0x0	
data present	0x1	

surplus_block_flag: Binary flag to signal the presence of the parameter it describes.

Table B.53

surplus_block_flag	Value ($_h$)	remark
data absent	0x0	
data present	0x1	

keyset_block_length: This parameter indicates the length in bits of the total keyset_block. That is the part in the sessionkey_block() plus the optional second part from the surplus_block().

unique_group_key: A symmetric AES encryption key to address a unique group. This key is also known as UGK. The key length SHALL be 128 bit.

NOTE 1: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

subscriber_group_key: An (set of) AES symmetric encryption key(s) which are used for the zero message subscriber_group_key deduction of the key needed to decrypt the SEK and/or PEK. These subscriber_group_key is also known as SGK. The key length SHALL be 128 bit.

NOTE 2: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

unique_device_key: An AES symmetric key to address a unique device. This key is also known as UDK. The key length SHALL be 128 bit.

NOTE 3: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

unique_device_filter: A [16] style addressing scheme used to filter for messages like BCROs. A device address consists of 5 bytes and is unique within an operation. The shared address is defined as the 4 most significant bytes of the unique address. The least significant byte (byte 5) defines the position (0….255) in the group that shares an address. This means that each group consists of 256 members. An access mask, in an entitlement, is used to identify individual members. So if for a particular group only member 5 and 100 are allowed to have access to a service then their corresponding bits are set in the access mask. Take the device_id_mask equal to 252 ($1111\ 1100_b$) then the least significant byte of the device_id is masked and thereby creating a shared address. This address is also known as UDF.

NOTE 4: This address is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

***ri_authentication_key*:** An AES symmetric key to verify MACs on BCRO and KSM messages. This key is also known as RIAK. The key length SHALL be 128 bit.

NOTE 5: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

***token_delivery_key*:** This is the Token Delivery Key (TDK), which is used in Section B.3.4.5.4.1.

NOTE 6: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

broadcast_domain_key: An AES symmetric key to address a Broadcast Domain. This key is also known as BDK. The key length SHALL be 128 bit.

NOTE 7: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

***longform_domain_id()*:**- This parameter is also known as the Longform Broadcast Domain Filter (LBDF). Please refer to clause B.18.3 for the definition. The longform_domain_id() is used for mixed-mode operation. Note: This address is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

shortform_domain_id: This parameter is also known as the Shortform Broadcast Domain Filter (SBDF). Please refer to clause B.18. An addressing scheme used to filter for messages like BCROs. The shortform_domain_id is used for broadcast mode of operation.

NOTE 8: This address is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

drm_time: This parameter defines the time in Universal Time Coordinated (UTC). This 40-bit field contains the current time and date in UTC and MJD. Refer to clause B.9 for calculation of the UTC and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4-bit BCD.

EXAMPLE: 93/10/13 12:45:00 is coded as '0xC079124500'.

local_time_offset: This parameter indicates the local time offset from the (UTC) drm_time as explained in clause B.9.1.

registration_timestamp_start: Indicates from what time onwards the registration data is valid. This is an extra mechanism above the expiration date of the RI certificate.

NOTE 9: This parameter can also be used against replay attacks.

registration_timestamp_end: Indicates from what time onwards the registration data is expires. This is an extra mechanism above the expiration date of the RI certificate.

NOTE 10: This parameter can also be used against replay attacks.

shortform_udn: This parameter allows the RI to give an own defined short number identifying the device. This number can be used as a shorter alternative to the UDN during offline notifications. The shortform_udn is coded in BCD format.

***signature_block*:** The signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1 024 or RSA-2 048 or RSA-4 096. The signature will apply to the implementation guidelines of PKCS#1, as outlined in clause B.16.

Note Message result:

The stored RI Context SHALL at a minimum contain:

- RI ID, Unique device filter (UDF).
- following keys:
  - UGK, SGK1..n and/or UDK
  - RIAK.
  - SK

  If domain addressing via an OMA DRM 2.0 domain is required the keyset SHALL (additionally to the standard addressing above) include following keys:

  - BDK.
  - Shortform Broadcast Domain Filter (SBDF). A.k.a. 'shortform_domain_id'. Refer to clause B.18.1.
- For mixed-mode devices domain context SHALL additionally contain:
  - Longform Broadcast Domain Filter (LBDF). A.k.a. 'longform_domain_id()'. Refer to clause B.18.3.
- A Device MAY have several Domain Contexts with an RI.
- The RI Context SHALL also contain an RI Context Expiry Time, which is defined to be the timestamp of the registration data if that was send and otherwise the expiration of the RI certificate.
- The RI Context MAY also contain RI certificate validation data.
- If the RI Context has expired, the Device SHALL NOT execute any other protocol than the 1-pass binary device data registration protocol with the associated RI (context), and upon detection of RI Context expiry the Device SHOULD initiate the offline notification of detailed device data protocol using the RI_ID stored in the RI Context. Depending on the implementation a dialogue will be shown to the user and the offline NDD protocol will be executed.
  - Accessing an ESG for purchase is still allowed, as this will require a registration first.
  - The device SHALL be rendered inoperable for any purchase protocol or playback of future content. The device MAY use stored BCROs to play old content for which the device obtained ROs, but SHALL NOT use these BCROs for new content received after the re-registration request until the device is re-registered with the RI.

Requirements:

- The Device SHALL have at most one RI Context with each RI. An existing RI Context SHALL be replaced with a newly established RI Context after successful re-registration with the same RI.
- The device SHALL support at least 6 RI context for broadcast mode of operation.
- For standard addressing the keyset SHALL include a valid set of :
  - UGK, SGK1..n and/or UDK keys.
  - RIAK key. A single RIAK key is bound to a single group, or, if no UGK nor SGKs have been issued to the Device, is bound to a single Device.
  - Unique device filter (UDF).
- If domain addressing via an OMA DRM 2.0 domain is required the keyset SHALL (additionally to the standard addressing above) include a valid set of :
  - BDK key.
  - Shortform Broadcast Domain Filter (SBDF). A.k.a. 'shortform_domain_id'. Refer to clause B.18.1.

  And in case of mixed-mode operation devices the keyset SHALL contain:

  - A Longform Broadcast Domain Filter (LBDF, a.k.a. 'longform_domain_id()') that matches the SBDF. Refer to clause B.18.3.

B.3.4.3.7.2.2 Protection of the keyset

The device_registration_response() message is split in two parts: device specific (time bound) data and global (not time bound) data.

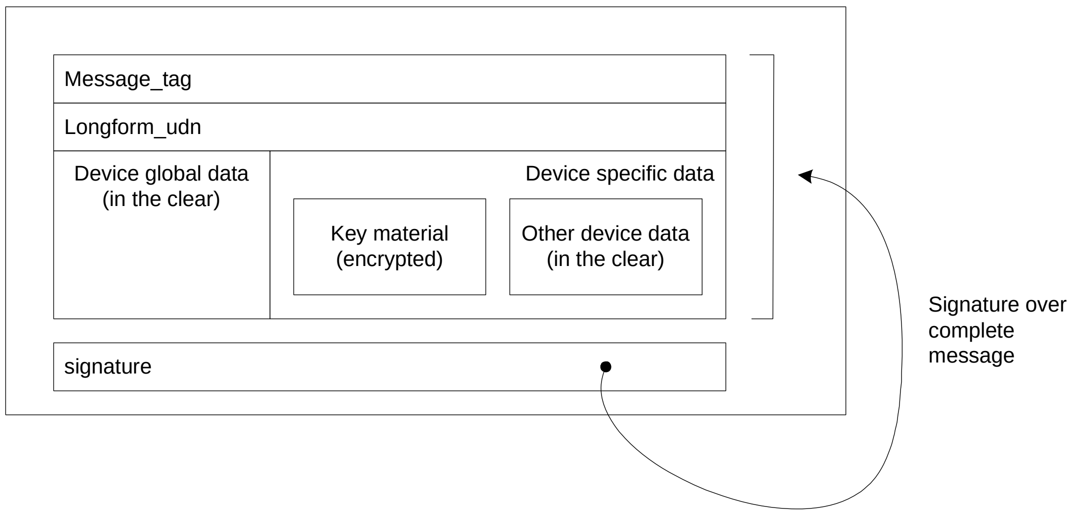

Message_tag
Longform_udn
Device global data (in the clear)
Device specific data
Key material (encrypted)
Other device data (in the clear)
signature
Signature over complete message

Figure B.36: device_registration_response() message

The device global data SHALL be in the clear. The device specific data contains the keyset for the device. This key material SHALL be encrypted, whereas the rest of the device specific data SHALL be in the clear. The key material SHALL be protected by encryption. The RI SHALL use the device's public key to encrypt all key material in the device specific data part of the message.

The RI SHALL use his private key to sign the complete message data. Upon reception the device SHALL verify the RI signature, by using the issuer's public key from the RI certificate. The device SHALL make sure that this message is correct by using a valid and correct RI certificate.

The complete message SHALL be authenticated by a signature from the RI.

Creation of the encrypted message SHALL adhere to the following rules:

1) Generate a (128 or 192 or 256) bit AES key to be used as session key (SK) for the device_registration_response() message.

2) Concatenate the keyset (UGK, SGK1..n, UDK, RIAK, UDF and/or BDK, SBDF plus optional LBDF if applicable) under rules of FIPS PUB 197 [17] and the Tag Length Format described in clause B.18. The concatenated keyset shall be padded with one bit with the value '1' and, after this 1-valued bit, 0 to 63 bits with the value '0', such that the length of the padded keyset is a multiple of 64 bits, see NIST 800-38A [46]. Note that if the non-padded keyset was already a multiple of 64 bits in length, it is padded with 64 bits.

3) Encrypt the keyset using [13] using the generated SK as (AES-WRAP style) KEK. This will produce the *keyset_block*.

4) Calculate the part of the keyblock that would fit into the RSA block (depending on the size of RSA used, be that 1 024, 2 048 or 4 096), including the SK and under implementation rules of the PKCS#1. If the keyset_block fits into one RSA block continue at step 6. Else continue at step 5.

5) If the SK plus keyset_block including PKCS#1 header, aligning, etc did not fit into one RSA block, then keep the remainder part as surplus_block().

6) Encrypt SK plus the (part of the)keyset_block that fits into the RSA block with the public key of the target device using RSA (1 024 or 2 048 or 4 096) under implementation guidelines of [21]. This will produce the *sessionkey_block()*.

7) Concatenate the (non encrypted) parameters that were not used in the key_block and create the message 'header' from this. Refer to clause B.3.4.3.7.2.3 for details. (For reason of completeness: of course the sessionkey_block(), the (optional) surplus_block() and the signature_block are not part of the message header.)

8) Concatenate the message 'header' and the sessionkey_block() . If the SK plus keyset_block including PKCS#1 header, aligning, etc did not fit into one RSA block, then also concatenate surplus_block() part. Hash the result under implementation guidelines of [21]. Please refer to clause B.16. This will produce the signature_input_data.

9) Sign the signature_input_data with RSA (1 024 or 2 048 or 4 096) using the private key of the RI. The signature will apply to the implementation guidelines of PKCS#1, as outlined in clause B.16. This will produce the *signature_block*.

10) The device_registration_response() message comprises of the message 'header' plus sessionkey_block(), optionally the surplus_block() and the signature_block.

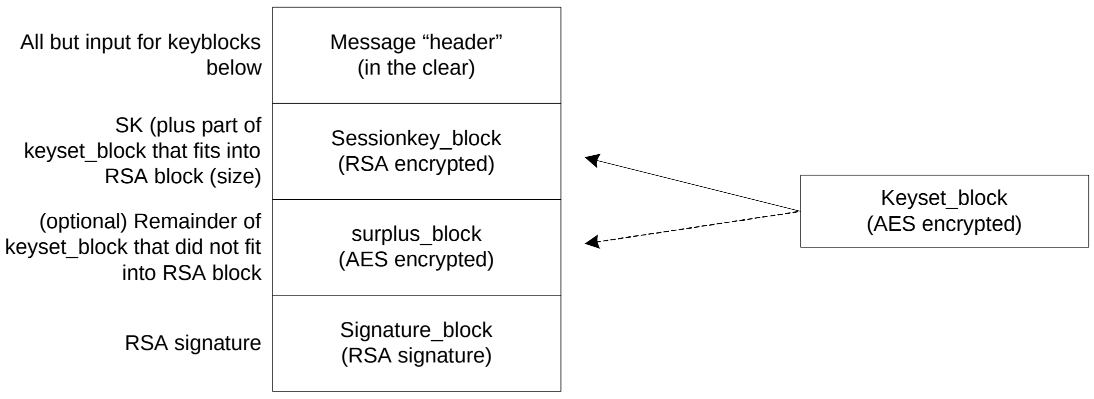

All but input for keyblocks below
Message "header" (in the clear)
SK (plus part of keyset_block that fits into RSA block (size)
Sessionkey_block (RSA encrypted)
(optional) Remainder of keyset_block that did not fit into RSA block
surplus_block (AES encrypted)
RSA signature
Signature_block (RSA signature)
Keyset_block (AES encrypted)

Figure B.37: structure of device_registration_response() message

Concluding: The number of RSA blocks used should be kept to a minimum. The AES surplus_block() is present if and when the keyset does not completely fit into the sessionkey_block() given the RSA block size used. If present the AES surplus_block() contains those keys that did not fit into one RSA block (i.e. the sessionkey_block()). The complete keyset needed for operation after registration is included in the encrypted keyset_block, which is concatenated from the first part in the sessionkey_block() and optionally the surplus_block(). Refer to clause B.12 for calculations on the surplus_block_size.

Decryption of the encrypted message SHALL adhere to the following rules:

1) Locate the message via message_tag.

2) Verify if the message is intended for this device by comparing the long_form_udn with the UDN stored in the device.

3) Verify the signature_block of the message by using the public key from the RI.

4) Locate the sessionkey_block() and decrypt the block with the private key of the local device. Locate the session key (SK) from the header and (eventual) padding (according to PKCS#1). Then locate the keyset_block part from the header and (eventual) padding (according to PKCS#1).

5) (Optionally) If there is a surplus_block() concatenate this part to the keyset_block. This will complete the keyset_block.

6) Use the SK to decrypt the keyset_block.

7) Allocate the individual keyset_items from the keyset_block according to [13] and the Tag Length Format described in clause B.18.

NOTE: The SK SHALL be stored into protected storage. The AES encrypted keyset_block MAY be stored as is into unprotected storage and decrypted by the device upon use. If the keyset_block is not stored but the decrypted keys from that block are stored instead, the device SHALL store all key data safely. The keys SHALL NOT leak outside the device.

B.3.4.3.7.2.3 Message syntax

Table B.54: Message syntax

Fields	Length	Type
device_registration_response(){		
/* signature protected part starts here */		
/* message header starts here */		
message_tag	8	bslbf

Fields	Length	Type
protocol_version	4	bslbf
reserved_for_future_use	4	bslbf
longform_udn()	80	bslbf
status	8	bslbf
flags {		
ri_certificate_counter	3	bslbf
ocsp_response_counter	3	bslbf
local_time_offset_flag	1	bslbf
time_stamp_flag	1	bslbf
subscriber_group_key_flag	4	bslbf
short_udn_flag	1	bslbf
signature_type_flag	2	bslbf
surplus_block_flag	1	bslbf
keyset_block_length	16	uimsbf
}		
certificate_version	8	bslbf
for(cnt1=0; cnt1 < ri_certificate_counter ;cnt1++){		
c_length	16	uimsbf
ri_certificate()	8 × c_length	bslbf
}		
for(cnt2=0; cnt2 < ocsp_response_counter ;cnt2++){		
r_length	16	uimsbf
ocsp_response()	8 × r_length	bslbf
}		
drm_time	40	mjdutc
if (local_time_offset_flag == 0x1) {		
local_time_offset	16	bslbf
}		
if (time_stamp_flag == 0x1) {		
registration_timestamp_start	40	mjdutc
registration_timestamp_end	40	mjdutc
}		
if (short_udn_flag == 0x1) {		
short_udn	32	bslbf
}		
/* message header ends here */		
if (signature_type_flag == 0x0){		
sessionkey_block()	1 024	bslbf
} else if (signature_type_flag == 0x1){		
sessionkey_block()	2 048	bslbf
} else if (signature_type_flag == 0x2){		
sessionkey_block()	4 096	bslbf
}		
if (surplus_block_flag == 0x1){		
surplus_block()	(note)	bslbf
}		
/* signature protected part ends here */		
if (signature_type_flag == 0x0){		
signature_block	1 024	bslbf
} else if (signature_type_flag == 0x1){		
signature_block	2 048	bslbf
} else if (signature_type_flag == 0x2){		
signature_block	4 096	bslbf
}		
}		
NOTE: For details please refer to clause B.12.		

B.3.4.3.7.3 (Force to) Re-register - re_register_msg() message

B.3.4.3.7.3.1 Message description

Using the 1-pass IRD protocol (refer to clause B.3.4.3.4) the RI sends a register_msg message, indirectly triggering a (re)registration . The message is specified as follows.

Table B.55: message description

re_register_msg( )		
Parameter name	(M)andatory / (O)ptional (see note)	Remark
message_tag	M	
protocol_version	M	
longform_udn	M	
status	M	
signature_type_flag	M	
certificate_version	M	
ri_certificate_counter	M	
c_length	M	
ri_certificate	M	
ocsp_response_counter	M	
r_length	M	
ocsp_response	M	
signature_block	M	
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).
If set to 0x0 the format specified in the present document is used. If set to anything else than 0x0, then the format is beyond the scope of the present document.

longform_udn(): The long form of the UDN. Refer to clause B.3.4.3.6 for details.

status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.56: Status values

Status value	**Meaning**
Success	The message contains valid reregistration message and cancels any preceding forced channel usage restrictions.
ForceInteractiveChannel	If the device is a mixed mode device the (re)registration will be possible via OOB and/or the interactive channel. By using this status code the RI can indicate to the device that the device SHALL direct subsequent (re)registrations to the RI over the device's interactive channel only. When the device receives this status code it will also exclusively use the interaction channel for all other messages. When the interactive channel of the device is not able to connect to the RI the mixed mode device MAY revert back to the OOB re-registration dialogue. Please note that a mixed mode device will remain to have full broadcast reception capabilities after receiving this status code.
ForceOobChannel	If the device is a mixed mode device the (re)registration will be possible via OOB and/or the interactive channel. By using this status code the RI can indicate to the device that the device SHALL direct subsequent (re)registrations to the RI over the device's OOB channel. When the device receives this status code it will also exclusively use the OOB channel for all other messages. Please note that a mixed mode device will remain to have full interactive channel capabilities after receiving this status code, but will not use the interactive channel.

Please refer to clause B.7 for the value of the error codes.

signature_type_flag: A flag to signal type of signature algorithm used.

Table B.57

signature_type_flag	Value ($_h$)	remark
RSA 1 024	0x0	
RSA 2 048	0x1	CMLA requirement (2004-2007)
RSA 4 096	0x2	
Reserved for future use	0x3	Not used in the present document

certificate_version: Is a numerical representation of the version of the RI certificate. Using the certificate_version parameter the device can decide if it is needed to update the RI certificate (if it was stored before).

Table B.58: Description of certificate_version parameter

Parameter Fieldname	Field Value ($_h$)	supports
major_version_number	0x0,..,0xF	MSB_4(certificate_version)
minor_version_number	0x0,..,0xF	LSB_4(certificate_version)

The parameter is divided 2 fields of 4 bits, whereas the first 4 bits (MSB left) express the Major number and the last four bits (LSB right) express the Minor version. The major and minor number encode in bslbf format. 16 Major and 16 Minor versions are supported. For example: Major.Minor version <1.2> is expressed as $0001\ 0010_b$.

ri_certificate_counter: This parameter indicates the depth of the RI certificate chain.

Table B.59

number of certificate in chain	Value ($_h$)	remark
0	0x0	Will signal absence of ri_certificate e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 RI certificates.		

c_length: This parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): This parameter SHALL be present. When present, the value of a *ri_certificate* parameter SHALL be a certificate chain including the RI's certificate. The chain SHALL NOT include the root certificate. The RI certificate SHALL come first in the list. Each following certificate SHALL directly certify the one preceding it.

The Device MAY store RI certificate verification data indicating that an RI certificate chain has been verified. The purpose of this is to avoid repeated verification of the same certificate chain. The RI certificate verification data stored in this way SHALL uniquely identify the RI certificate and SHALL be integrity protected. The Device SHOULD check if the RI certificate chain received in this parameter corresponds to the stored certificate verification data for this RI. If so, the Device NEED NOT verify the RI certificate chain again, otherwise the Device SHALL verify the RI certificate chain.

If an RI certificate is received that is not in the stored certificate verification data for this RI, and if the Device can determine (in the case of Broadcast Devices that support DRM Time) that the expiry time of the received RI certificate is later than the RI Context for this RI, and the certificate status of the RI certificate as indicated in the OCSP response is good (see [OCSP-MP]), then the Device SHALL verify the complete chain and SHOULD replace the stored RI certificate verification data with the received RI certificate data and set the RI context expiry time to that of the received RI certificate expiry time.

However, if the Device does store RI certificate verification data in this way it SHALL store the expiry period of the RI's certificate (as indicated by the notAfter field within the certificate) and SHALL compare the Device's current DRM Time with the stored RI certificate expiry time whenever verifying the signature on signed messages from the RI. If the Device's current DRM Time is after the stored RI certificate expiry time then the Device SHALL abandon processing the RI message and SHALL initiate the registration protocol.

ocsp_response_counter: This parameter indicates the depth of the OCSP response chain.

Table B.60

Number of responses in chain	Value (h)	Remark
0	0x0	Will signal absence of ocsp_response e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 OCSP responses.		

r_length: This parameter indicates the length in bytes of the ocsp_response.

ocsp_response(): This parameter, when present, SHALL be a complete set of valid OCSP responses for the RI's certificate chain. The Device SHALL NOT fail due to the presence of more than one OCSP response element. A Device SHALL check that an OCSP response is present in the received message. If no OCSP response is present in the device_registration_response() message, then the Device SHALL abort the registration protocol.

signature_block: The signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1 024 or RSA-2 048 or RSA-4 096. The signature will apply to the implementation guidelines of PKCS#1, as outlined in clause B.16.

B.3.4.3.7.3.2 Message syntax

Table B.61: message syntax

Fields	Length	Type
`re_register_msg() {`		
`   /* signature protected part starts here */`		
`   message_tag`	8	bslbf
`   protocol_version`	4	bslbf
`   reserved for future use`	4	bslbf
`   longform_udn()`	80	bslbf
`   status`	8	bslbf
`   flags {`		
`      signature_type_flag`	2	bslbf
`      ri_certificate_counter`	3	bslbf
`      ocsp_response_counter`	3	bslbf
`      reserved for future use`	8	bslbf
`   }`		
`   certificate_version`	8	bslbf
`   for(cnt1=0; cnt1 < ri_certificate_counter ;cnt1++){`		
`      c_length`	16	uimsbf
`      ri_certificate()`	8 × c_length	bslbf
`   }`		
`   for(cnt2=0; cnt2 < ocsp_response_counter ;cnt2++){`		
`      r_length`	16	uimsbf
`      ocsp_response()`	8 × r_length	bslbf
`   }`		
`   /* signature protected part ends here */`		
`   if (signature_type_flag == 0x0){`		
`      signature_block`	1 024	bslbf
`   } else if (signature_type_flag == 0x1){`		
`      signature_block`	2 048	bslbf
`   } else if (signature_type_flag == 0x2){`		
`      signature_block`	4 096	bslbf
`   }`		
`}`		

B.3.4.3.7.4 Update RI certificate - update_ri_certificate_msg() message

Using the 1-pass IRD protocol (refer to clause B.3.4.3.4) the RI sends an update_ri_certificate_msg() message, forcing the device to update his RI certificate chain.

This update_ri_certificate_msg() trigger is almost identical to the re_register_msg() message described in clause B.3.4.3.7.3, with the only adaptation being that the message_tag is different. Refer to clause B.19 for the value of the message_tag.

B.3.4.3.7.5 Updating the DRM time - update_drmtime_msg() message

B.3.4.3.7.5.1 Message description

Using the 1-pass IRD protocol (refer to clause B.3.4.3.4) the RI sends an update_drmtime trigger message with the drmtime to the device as specified below:

Table B.62: message description

update_drmtime_msg( )		
Parameter name	(M)andatory / (O)ptional (see note)	remark
message_tag	M	
protocol_version	M	
status	M	
signature_type_flag	M	
local_time_offset_flag	M	
drm_time	M	
local_time_offset	O	
signature_block	M	
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).
If set to 0x0 the format specified in the present document is used. If set to anything else than 0x0, then the format is beyond the scope of this version of the specification.

status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.63: Status values

status value	**meaning**
Success	The message contains valid DRM time RI.
NotSupported	The RI does not support the sending of DRM time request. The device will use other means to update DRM time.
DeviceTimeError	The RI concluded that the DeviceTime might be false and forces the device to update its time. As an extra result the device will determine the eventual clock drift and notify this to the RI per ARC (offline notification of short device data; refer to clause B.3.4.4.3. Please note: this capability should be used with great care.)

Please refer to clause B.7 for the value of the error codes.

local_time_offset_flag: Binary flag to signal presence of the parameter it describes.

Table B.64

local_time_offset_flag	Value ($_h$)	Remark
data absent	0x0	
data present	0x1	

signature_type_flag: A flag to signal type of signature algorithm used.

Table B.65

signature_type_flag	Value ($_h$)	Remark
RSA 1 024	0x0	
RSA 2 048	0x1	CMLA requirement (2004-2007)
RSA 4 096	0x2	
reserved for future use	0x3	not used in the present document

drm_time: This parameter defines the time in Universal Time Coordinated (UTC). This 40-bit field contains the current time and date in UTC and MJD. Refer to clause B.9 for calculation of the UTC and Modified Julian Date (MJD). This field is coded as 16 bits giving the 16 LSBs of MJD followed by 24 bits coded as 6 digits in 4-bit BCD.

EXAMPLE: 93/10/13 12:45:00 is coded as '0xC079124500'.

local_time_offset: This parameter indicates the local time offset from the (UTC) drm_time as explained in clause B.9.1.

signature_block: The signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1 024 or RSA-2 048 or RSA-4 096. The signature will apply to the rules of PKCS#1, as outlined in clause B.16.

B.3.4.3.7.5.2 Message syntax

Table B.66: Message syntax

Fields	Length	Type
`update_drmtime_msg(){`		
`   /* signature protected part starts here */`		
`   message_tag`	8	bslbf
`   protocol_version`	4	bslbf
`   reserved_for_future_use`	4	bslbf
`   status`	8	bslbf
`   flags {`		
`      local_time_offset_flag`	1	bslbf
`      signature_type_flag`	2	bslbf
`      reserved for future use`	5	bslbf
`   }`		
`   drm_time`	40	mjdutc
`   if (local_time_offset_flag == 0x1) {`		
`      local_time_offset`	16	bslbf
`   }`		
`   /* signature protected part ends here */`		
`   if (signature_type_flag == 0x0){`		
`      signature_block`	1 024	bslbf
`   } else if (signature_type_flag == 0x1){`		
`      signature_block`	2 048	bslbf
`   } else if (signature_type_flag == 0x2){`		
`      signature_block`	4 096	bslbf
`   }`		
`}`		

B.3.4.3.7.6 Update the contact number - update_contact_number_msg() message

B.3.4.3.7.6.1 Message description

Using the 1-pass IRD protocol (refer to clause B.3.4.3.4) the RI sends an update_contact_number_msg() message with a (set of) contact number(s) to the device as specified below.

Table B.67: message description

update_contact_number_msg( )		
Parameter name	(M)andatory / (O)ptional (see note)	Remark
message_tag	M	
protocol_version	M	
status	M	
signature_type_flag	M	
ri_certificate_counter	M	
c_length	M	
ri_certificate	M	
ocsp_response_counter	M	
r_length	M	
ocsp_response	M	
contact_counter	M	
contact	O	
signature_block	M	
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

***protocol_version*:** This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).
If set to 0x0 the format specified in the present document is used. If set to anything else than 0x0, then the format is beyond the scope of the present document.

status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.68: Status values

Status value	Meaning
Success	The message contains valid contact numbers from the RI.
NotSupported	The RI does not support the sending of contact numbers. The device will use other means to use contact numbers (e.g. via ESG).

Please refer to clause B.7 for the value of the error codes.

signature_type_flag: A flag to signal type of signature algorithm used.

Table B.69

signature_type_flag	Value ($_h$)	Remark
RSA 1 024	0x0	
RSA 2 048	0x1	
RSA 4 096	0x2	
Reserved for future use	0x3	Not used in the present document

certificate_version: is a numerical representation of the version of the RI certificate. Using the certificate_version parameter the device can decide if it is needed to update the RI certificate (if it was stored before).

Table B.70: Description of certificate_version parameter

Parameter Fieldname	Field Value ($_h$)	Supports
major_version_number	0x0,..,0xF	MSB_4(certificate_version)
minor_version_number	0x0,..,0xF	LSB_4(certificate_version)

The parameter is divided 2 fields of 4 bits, whereas the first 4 bits (MSB left) express the Major number and the last four bits (LSB right) express the Minor version. The major and minor number encode in bslbf format. 16 Major and 16 Minor versions are supported. For example: Major.Minor version <1.2> is expressed as 0001 0010_b.

***ri_certificate_counter*:** This parameter indicates the depth of the RI certificate chain.

Table B.71

Number of certificate in chain	Value ($_h$)	Remark
0	0x0	Will signal absence of ri_certificate e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 RI certificates.		

c_length: This parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): This parameter SHALL be present. When present, the value of a *ri_certificate* parameter SHALL be a certificate chain including the RI's certificate. The chain SHALL NOT include the root certificate. The RI certificate SHALL come first in the list. Each following certificate SHALL directly certify the one preceding it.

The Device MAY store RI certificate verification data indicating that an RI certificate chain has been verified. The purpose of this is to avoid repeated verification of the same certificate chain. The RI certificate verification data stored in this way SHALL uniquely identify the RI certificate and SHALL be integrity protected. The Device SHOULD check if the RI certificate chain received in this parameter corresponds to the stored certificate verification data for this RI. If so, the Device NEED NOT verify the RI certificate chain again, otherwise the Device SHALL verify the RI certificate chain.

If an RI certificate is received that is not in the stored certificate verification data for this RI, and if the Device can determine (in the case of Broadcast Devices that support DRM Time) that the expiry time of the received RI certificate is later than the RI Context for this RI, and the certificate status of the RI certificate as indicated in the OCSP response is good (see [OCSP-MP]), then the Device SHALL verify the complete chain and SHOULD replace the stored RI certificate verification data with the received RI certificate data and set the RI context expiry time to that of the received RI certificate expiry time.

However, if the Device does store RI certificate verification data in this way it SHALL store the expiry period of the RI's certificate (as indicated by the notAfter field within the certificate) and SHALL compare the Device's current DRM Time with the stored RI certificate expiry time whenever verifying the signature on signed messages from the RI. If the Device's current DRM Time is after the stored RI certificate expiry time then the Device SHALL abandon processing the RI message and SHALL initiate the registration protocol.

ocsp_response_counter: This parameter indicates the depth of the OCSP response chain.

Table B.72

Number of responses in chain	Value ($_h$)	Remark
0	0x0	Will signal absence of ocsp_response e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 OCSP responses.		

***r_length*:** This parameter indicates the length in bytes of the ocsp_response.

ocsp_response(): This parameter, when present, SHALL be a complete set of valid OCSP responses for the RI's certificate chain. The Device SHALL NOT fail due to the presence of more than one OCSP response element. A Device SHALL check that an OCSP response is present in the received message. If no OCSP response is present in the device_registration_response() message, then the Device SHALL abort the registration protocol.

contacts_counter: This parameter indicates the number of contacts carried in the message.

***contact*:** This object specifies the contact. Please refer to clause B.3.4.3.7.6.2.1.

signature_block: The signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1 024 or RSA-2 048 or RSA-4 096. The signature will apply to the implementation guidelines of PKCS#1, as outlined in clause B.16.

B.3.4.3.7.6.2 Message syntax

Table B.73: message syntax

Fields	Length	Type
`update_contact_number_msg() {`		
`   /* signature protected part starts here */`		
`   message_tag`	8	bslbf
`   protocol_version`	4	bslbf
`   reserved_for_future_use`	4	bslbf
`   status`	8	bslbf
`   flags {`		
`      contacts_counter`	4	bslbf
`      reserved for future use`	4	bslbf
`      signature_type_flag`	2	bslbf
`      ri_certificate_counter`	3	bslbf
`      ocsp_response_counter`	3	bslbf
`   }`		
`   certificate_version`	8	bslbf
`   for(cnt1=0; cnt1 < ri_certificate_counter ;cnt1++){`		
`      c_length`	16	uimsbf
`      ri_certificate()`	8×c_length	bslbf
`   }`		
`   for(cnt2=0; cnt2 < ocsp_response_counter ;cnt2++){`		
`      r_length`	16	uimsbf
`      ocsp_response()`	8×r_length	bslbf
`   }`		
`   for(cnt3=0; cnt3 < contacts_counter ;cnt3++){`		
`      contact()`		
`   }`		
`   /* signature protected part ends here */`		
`   if (signature_type_flag == 0x0){`		
`      signature_block`	1 024	bslbf
`   } else if (signature_type_flag == 0x1){`		
`      signature_block`	2 048	bslbf
`   } else if (signature_type_flag == 0x2){`		
`      signature_block`	4 096	bslbf
`   }`		
`}`		

B.3.4.3.7.6.2.1 Format of the contact object

Table B.74: contact object format

Field	Length	Type
`contact(){`		
`contact_type`	4	uimsbf
`reserved for future use`	4	bslbf
`contact_length`	8	uimsbf
`contactdata`	8×contact_length	bslbf
`}`		

contact_type**:** This field specifies the type of action as listed in table B.75.

Table B.75: contact_type

contact_type	Description	Comments	Max length (chars)
0x00	local_ri_phone_number	The number the user of the device needs to contact to start service provision.	20
0x01	int_ri_phone_number	The number the user of the device needs to contact to start service provision when he would call from abroad.	20
0x02	ri_sms_number	The SMS number the user of the device needs to contact to start service provision.	20
0x03	ri_url	The URL address the user of the device needs to contact to start service provision.	30
0x04	local_home_coc_phone_number	The number the user of the device needs to contact to start service provision.	20
0x05	int_home_coc_phone_number	The number the user of the device needs to contact to start service provision when he would call from abroad.	20
0x06	home_coc_sms_number	The SMS number the user of the device needs to contact to start service provision.	20
0x07	home_coc_url	The URL address the user of the device needs to contact start service provision.	30
0x08	local_reporting_phone_number	The number the user of the device needs to contact to report token consumption.	20
0x09	int_reporting_phone_number	The number the user of the device needs to contact to report token consumption when he would call from abroad.	20
0x0A	reporting_sms_number	The SMS number the user of the device needs to contact to report token consumption.	20
0x0B	reporting_url	The URL address the user of the device needs to contact to report token consumption.	30
0x0C-0x0F	Reserved for future use		

contact_length**:** This parameter indicates the length in bytes of the contact field. Maximum length of the contacts is specified in table B.75.

UTF-8 [30] character encoding for ASCII characters is 'efficient' with 1 byte per character. On the other hand, there are characters that are encoded using 6 bytes (Asian languages).

For example: a URL is limited to 30 characters. The 30 URL UTF-8 characters are translated into bytes as follows:

E.g.: "Western" languages - character is 1 byte - Longest URL encoded as bytes is 1 × 30 characters = 30 bytes.

E.g.: Asian languages - character is 6 bytes - Longest URL encoded as bytes is 6 × 30 characters = 180 bytes.

contactdata: The value in this field specifies any of the contact_type possibilities the user of the device needs to contact (via other means) to start service provision.

Table B.76: Contactdata encoding rules

Contact types	Contactdata encoding rules
phone numbers	The phone number is encoded as alphabetic, supporting telephone numbers like: '0800-123456789' but also for example: '0800-philips'. The string that forms the phone number is encoded using UTF-8.
SMS numbers	The SMS number is encoded as hexadecimal, supporting telephone numbers like: '0800-123456789' but also for example: 'philips+subscribe'. The string that forms the SMS number is encoded using UTF-8.
URLs	The URL is encoded as hexadecimal, according to [3], supporting URLs like: www.philips.com/start. The string that forms the URL is encoded using UTF-8.

B.3.4.4 Domain joining and leaving

Interactive devices will adhere to [49].

- Interactive devices will therefore use OMA DRM 2.0 domain ID.

Broadcast devices will adhere to the mechanisms as described in this clause.

- Broadcast devices will use 'shortform_domain_id' a.k.a. SBDF.

Mixed-mode SHALL have the "interoperability" requirement to support both domains ID formats of interactive and broadcast devices:

- Mixed-mode device will receive:
  - 'longform_domain_id()', a.k.a. LBDF, which is a translation of OMA DRM 2.0 domain ID.
  - 'shortform_domain_id' a.k.a. SBDF.
- mixed-mode devices registered for both interactive and broadcast operations MAY pass either domain ID format to other mixed-mode devices in the domain.
- interactive only devices SHALL pass longform_domain_id() format to other devices in the domain. The mixed-mode device will understand this, while broadcast does not understand.
- broadcast devices SHALL pass shortform_domain_id format to other devices in the domain. The mixed-mode device will understand this, while interactive does not understand.

B.3.4.4.1 Protocol overview

The theory of operation (refer to clause B.2) results in the specification of several protocols:

- offline protocols (from device to RI).

Table B.77

Protocol	Clause	Purpose
offline Domain Join Request protocol	B.3.4.4.2	request to join a domain
offline Domain Leave Request protocol	B.3.4.4.3	request to leave a domain

- 1-pass protocols (from RI to device).

Table B.78

Protocol	Clause	Purpose
1-pass binary Push Device Registration protocol	B.3.4.3.4	transmit registration data to device
1-pass binary Inform Registered Device protocol	B.3.4.3.5	inform device via messages.

The protocols interrelate in following way (roundtrip).

Table B.79

Kicking off action…	…results in
offline domain join request. (request to join a domain).	domain_registration_response() message. (transmit registration data to device).
offline domain leave request. (request to leave a domain).	domain_update_response() message. (inform device via messages)
join_domain_msg(). (inform device via messages).	offline domain join request, which on its turn may result in domain_registration_response() as listed above.
leave_domain_msg(). (inform device via messages).	offline domain leave request, which on its turn may result in domain_update_response() as listed above.

B.3.4.4.2 offline Domain Join Request

When the user of a device might want to join a particular domain, he uses the NSD protocol with the destined action code range. (refer to clause B.3.4.3.3).

B.3.4.4.3 offline Domain Leave Request

When the user of a device might want to leave a particular domain, he uses the NSD protocol with the destined action code range. (refer to clause B.3.4.3.3).

B.3.4.4.4 Binary messages

B.3.4.4.4.1 Domain data - domain_registration_response() message

B.3.4.4.4.1.1 Message description

Using the 1-pass PDR protocol (refer to clause B.3.4.3.4) the RI sends a domain_registration_response() message, informing the device of a new domain keyset. The message is specified below:

Table B.80: Message description

domain_registration_response()		
Parameter name	(M)andatory / (O)ptional (see note)	remark
message_tag	M	global, not encrypted
protocol_version	M	global, not encrypted
longform_udn	M	global, not encrypted
device_nonce	M	device specific, not encrypted
status	M	device specific, not encrypted
time_stamp_flag	M	device specific, not encrypted
certificate_version	M	global, not encrypted
ri_certificate_counter	M	global, not encrypted
c_length	M	global, not encrypted
ri_certificate	M	global, not encrypted
ocsp_response_counter	M	global, not encrypted
r_length	M	global, not encrypted
ocsp_response	M	global, not encrypted
domain_timestamp_start	O	device specific, not encrypted
domain_timestamp_end	O	device specific, not encrypted
signature_type_flag	M	global, not encrypted
keyset_block_length	M	device specific, not encrypted
broadcast_domain_key	M	device specific, encrypted
longform_domain_id()	O	device specific, encrypted
shortform_domain_id	M	device specific, encrypted
signature_block	M	device specific, not encrypted
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).
If set to 0x0 the format specified in the present document. If set to anything else than 0x0, then the format is beyond the scope of the present document.

longform_udn(): The long form of the UDN. Refer to clause B.3.4.3.6 for details.

status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.81: Status values

Status value	Meaning
Success	The message contains valid domain registration data from the RI.
NotSupported	The RI does not support the sending of domain registration data from the RI. The RI SHALL NOT include any valid keyset in the message. The device will use other means to obtain valid domain registration data from the RI.
InvalidDomain	The RI could not recognize the domain identifier that was used in the join domain request or decided that the domain identifier is invalid. The RI SHALL NOT include any valid keyset in the message.
DomainFull	The RI indicates that no more devices are allowed to join the domain. The RI SHALL NOT include any valid keyset in the message.

Please refer to clause B.7 for the value of the error codes.

device_nonce: The device_nonce is the nonce which was present in the request (using the offline NSD protocol) to which this message is a response. This nonce is an encoded in BCD.

time_stamp_flag: Binary flag to signal presence of the parameter it describes.

Table B.82

time_stamp_flag	Value ($_h$)	Remark
data absent	0x0	
data present	0x1	

***certificate_version*:** Is a numerical representation of the version of the RI certificate. Using the certificate_version parameter the device can decide if it is needed to update the RI certificate (if it was stored before).

Table 83: description of certificate_version parameter

Parameter fieldname	Field Value ($_h$)	Supports
major_version_number	0x0,..,0xF	MSB_4(certificate_version)
minor_version_number	0x0,..,0xF	LSB_4(certificate_version)

The parameter is divided 2 fields of 4 bits, whereas the first 4 bits (MSB left) express the Major number and the last four bits (LSB right) express the Minor version. The major and minor number encode in bslbf format. 16 Major and 16 Minor versions are supported. For example: Major.Minor version <1.2> is expressed as $0001\ 0010_b$.

ri_certificate_counter: This parameter indicates the depth of the RI certificate chain.

Table B.84

Number of certificate in chain	Value ($_h$)	Remark
0	0x0	Will signal absence of ri_certificate e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 RI certificates.		

c_length: This parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): This parameter SHALL be present. When present, the value of a *ri_certificate* parameter SHALL be a certificate chain including the RI's certificate. The chain SHALL NOT include the root certificate. The RI certificate SHALL come first in the list. Each following certificate SHALL directly certify the one preceding it.

The Device MAY store RI certificate verification data indicating that an RI certificate chain has been verified. The purpose of this is to avoid repeated verification of the same certificate chain. The RI certificate verification data stored in this way SHALL uniquely identify the RI certificate and SHALL be integrity protected. The Device SHOULD check if the RI certificate chain received in this parameter corresponds to the stored certificate verification data for this RI. If so, the Device NEED NOT verify the RI certificate chain again, otherwise the Device SHALL verify the RI certificate chain.

If an RI certificate is received that is not in the stored certificate verification data for this RI, and if the Device can determine (in the case of Broadcast Devices that support DRM Time) that the expiry time of the received RI certificate is later than the RI Context for this RI, and the certificate status of the RI certificate as indicated in the OCSP response is good (see [OCSP-MP]), then the Device SHALL verify the complete chain and SHOULD replace the stored RI certificate verification data with the received RI certificate data and set the RI context expiry time to that of the received RI certificate expiry time.

However, if the Device does store RI certificate verification data in this way it SHALL store the expiry period of the RI's certificate (as indicated by the notAfter field within the certificate) and SHALL compare the Device's current DRM Time with the stored RI certificate expiry time whenever verifying the signature on signed messages from the RI. If the Device's current DRM Time is after the stored RI certificate expiry time then the Device SHALL abandon processing the RI message and SHALL initiate the registration protocol.

ocsp_response_counter: This parameter indicates the depth of the OCSP response chain.

Table B.85

Number of responses in chain	Value ($_h$)	remark
0	0x0	Will signal absence of ocsp_response e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 OCSP responses.		

***r_length*:** This parameter indicates the length in bytes of the ocsp_response.

ocsp_response(): This parameter, when present, SHALL be a complete set of valid OCSP responses for the RI's certificate chain. The Device SHALL NOT fail due to the presence of more than one OCSP response element. A Device SHALL check that an OCSP response is present in the received message. If no OCSP response is present in the domain_registration_response() message, then the Device SHALL abort the registration protocol.

domain_timestamp_start: Indicates from what time onwards the registration data for the domain is valid. This is an extra mechanism above the expiration date of the RI certificate.

NOTE 1: Please note that this parameter can also be used against replay attacks.

domain_timestamp_end: Indicates from what time onwards the registration data for the domain expires. This is an extra mechanism above the expiration date of the RI certificate.

NOTE 2: Please note that this parameter can also be used against replay attacks.

signature_type_flag: A flag to signal type of signature algorithm used.

Table B.86

signature_type_flag	Value ($_h$)	Remark
RSA 1 024	0x0	
RSA 2 048	0x1	CMLA requirement (2004-2007)
RSA 4 096	0x2	
reserved for future use	0x3	Not used in the present document

keyset_block_length: This parameter indicates the length in bits of the total keyset_block. That is the part in the sessionkey_block().

broadcast_domain_key: An AES symmetric key to address a Broadcast Domain. This key is also known as BDK. The key length SHALL be 128 bit.

NOTE 3: This key is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

longform_domain_id(): This parameter is also known as the Longform Broadcast Domain Filter (LBDF). Please refer to clause B.18.3 for the definition. The longform_domain_id() is used for mixed-mode operation. Note: This address is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

shortform_domain_id: This parameter is also known as the Shortform Broadcast Domain Filter (SBDF). Please refer to clause B.18. An addressing scheme used to filter for messages like BCROs. The shortform_domain_id is used for broadcast mode of operation.

NOTE 4: This address is wrapped into the keyset_block. (Refer to clause B.3.4.3.7.2.2).

signature_block: The signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1 024 or RSA-2 048 or RSA-4 096. The signature will apply to the implementation guidelines of PKCS#1, as outlined in clause B.16.

Note Message result:

The stored domain context SHALL at a minimum contain:

- Following keys:
  - BDK.
  - Shortform Broadcast Domain Filter (SBDF). A.k.a. 'shortform_domain_id'. Refer to clause B.18.1.
- For mixed-mode operation, devices" domain context SHALL additionally contain:
  - Longform Broadcast Domain Filter (LBDF). A.k.a. 'longform_domain_id()'. Refer to clause B.18.3.
- A Device MAY have several Domain Contexts with an RI.
- If the domain context has expired, the Device SHALL NOT execute any other protocol than the 1-pass binary device data registration protocol with the associated RI (context), and upon detection of domain context expiry the Device SHOULD initiate the offline notification of short device data protocol using the correct ARC. Depending on the implementation a dialogue will be shown to the user and the offline NSD protocol will be executed.
  - Accessing an ESG for purchase is still allowed, as this will require a (domain) registration first.

- The device SHALL be rendered inoperable for any purchase protocol or playback of future content. The device MAY use stored BCROs to play old content for which the device obtained ROs, but SHALL NOT use these BCROs for new content received after the re-registration request until the device is re-registered with the RI.

Requirements:

- If domain addressing via an OMA DRM 2.0 domain is required the keyset SHALL (additionally to the standard addressing as explained in clause B.3.4.3.7.2.2) include a valid set of :
  - BDK key.
  - Shortform Broadcast Domain Filter (SBDF). A.k.a. 'shortform_domain_id'. Refer to clause B.18.1.

  And in case of mixed-mode operation devices the keyset SHALL contain:

  - A Longform Broadcast Domain Filter (LBDF, a.k.a. 'longform_domain_id()') that matches the SBDF. Refer to clause B.18.3.

B.3.4.4.4.1.2 Protection of the keyset

The domain_registration_response() message is split in two parts: device specific (time bound) data and global (not time bound) data.

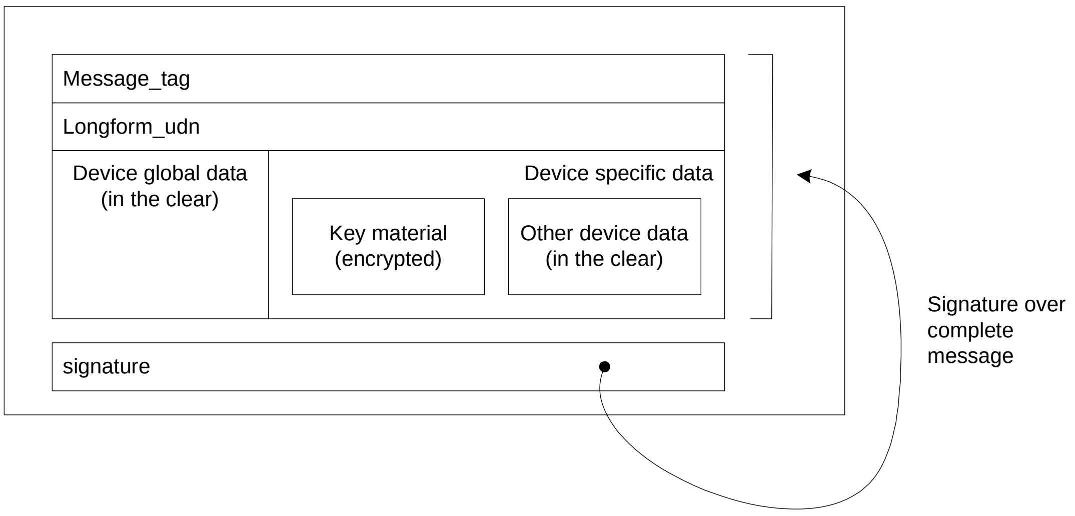

Message_tag
Longform_udn
Device global data (in the clear)
Device specific data
Key material (encrypted)
Other device data (in the clear)
signature
Signature over complete message

Figure B.38: domain_registration_response() message

The device global data SHALL be in the clear. The device specific data contains the keyset for the device. This key material SHALL be encrypted, whereas the rest of the device specific data SHALL be in the clear. The key material SHALL be protected by encryption. The RI SHALL use the device's public key to encrypt all key material in the device specific data part of the message.

The RI SHALL use his private key to sign the complete message data. Upon reception the device SHALL verify the RI signature, by using the issuer's public key from the RI certificate. The device SHALL make sure that this message is correct by using a valid and correct RI certificate.

The complete message SHALL be authenticated by a signature from the RI.

Creation of the encrypted message SHALL adhere to the following rules:

1) Generate a (128 or 192 or 256) bit AES key to be used as session key (SK) for the domain_registration_response() message.

2) Concatenate the keyset (BDK, SBDF plus optional LBDF if applicable) under rules of FIPS PUB 197 [17] and the Tag Length Format described in clause B.18. The concatenated keyset shall be padded with one bit with the value '1' and, after this 1-valued bit, 0 to 63 bits with the value '0', such that the length of the padded keyset is a multiple of 64 bits, see NIST 800-38A [46]. Note that if the non-padded keyset was already a multiple of 64 bits in length, it is padded with 64 bits. Note, for reasons of clarity: The keyset may contain multiple domain contexts (i.e. a matching BDK, SBDF and optionally LBDF) but not more than the maximum indicated size of the single RSA block used for the sessionkey_block.

3) Encrypt the keyset using [13] using the generated SK as (AES-WRAP style) KEK. This will produce the *keyset_block*.

4) Calculate the part of the keyblock that would fit into the RSA block (depending on the size of RSA used, be that 1 024, 2 048 or 4 096), including the SK and under implementation rules of the PKCS#1.

5) Encrypt SK plus the (part of the)keyset_block that fits into the RSA block with the public key of the target device using RSA (1 024 or 2 048 or 4 096) under the implementation guidelines of [21]. This will produce the *sessionkey_block()*. Encrypt the SK plus the keyset_block with the public key of the target device, using RSA (1 024 or 2 048 or 4 096) under the implementation guidelines of [21].

6) Concatenate the (non encrypted) parameters that were not used in the key_block and create the message 'header' from this. Refer to clause B.3.4.4.4.1 for details. (for reason of completeness: of course the sessionkey_block() and the signature_block are not part of the message header).

7) Concatenate the message 'header' and the sessionkey_block() . Hash the result under implementation guidelines of [21]. Please refer to clause B.16. This will produce the signature_input_data.

8) Sign the signature_input_data with RSA (1 024 or 2 048 or 4 096) using the private key of the RI. The signature will apply to the implementation guidelines of PKCS#1 [21], as outlined in B.16. This will produce the *signature_block*.

9) The domain_registration_response() message comprises of the message 'header' plus sessionkey_block() and the signature_block.

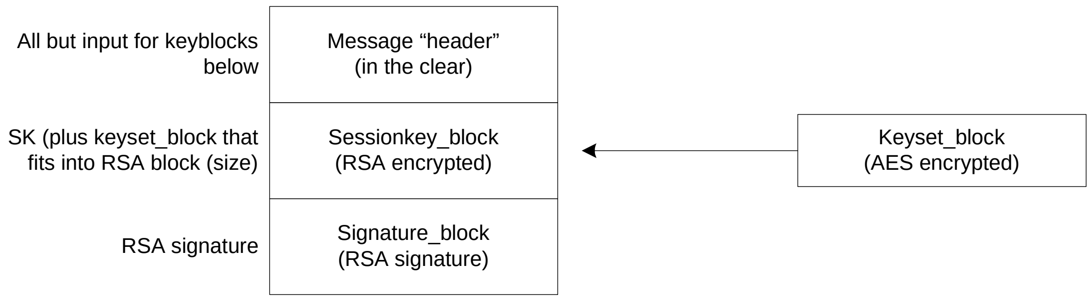

All but input for keyblocks below
Message "header" (in the clear)
SK (plus keyset_block that fits into RSA block (size)
Sessionkey_block (RSA encrypted)
Keyset_block (AES encrypted)
RSA signature
Signature_block (RSA signature)

Figure B.39: Structure of domain_registration_response() message

Decryption of the encrypted message SHALL adhere to the following rules:

1) Locate the message via message_tag.

2) Verify if the message is intended for this device by comparing the long_form_udn with the UDN stored in the device.

3) Verify the signature_block of the message by using the public key from the RI.

4) Locate the sessionkey_block() and decrypt the block with the private key of the local device. Locate the session key (SK) from the header and (eventual) padding (according to PKCS#1 [21]). Then locate the keyset_block part from the header and (eventual) padding (according to PKCS#1 [21]).

5) Use the SK to decrypt the keyset_block.

6) Allocate the individual keyset_items from the keyset_block according to [13] and the Tag Length Format described in clause B.18.

NOTE: The SK SHALL be stored into protected storage. The AES encrypted keyset_block MAY be stored as is into unprotected storage and decrypted by the device upon use. If the keyset_block is not stored but the decrypted keys from that block are stored instead, the device SHALL store all key data safely. The keys SHALL NOT leak outside the device.

B.3.4.4.4.1.3 Message syntax

Table B.87: message syntax

Fields	Length	Type
`domain_registration_response(){`		
`   /* signature protected part starts here */`		
`   /* message header starts here /*`		
`   message_tag`	8	bslbf
`   protocol_version`	4	bslbf
`   reserved_for_future_use`	4	bslbf
`   unique_device_number`	80	bslbf
`   reserved_for_future_use`	4	bslbf
`   device_nonce`	4	bslbf
`   status`	8	bslbf
`   flags {`		
`      ri_certificate_counter`	3	bslbf
`      ocsp_response_counter`	3	bslbf
`      signature_type_flag`	2	bslbf
`      time_stamp_flag`	1	bslbf
`      reserved for future use`	7	bslbf
`      keyset_block_length`	16	uimsbf
`   }`		
`   certificate_version`	8	bslbf
`   for(cnt1=0; cnt1 < ri_certificate_counter ;cnt1++){`		
`      c_length`	16	uimsbf
`      ri_certificate()`	8 × c_length	bslbf
`   }`		
`   for(cnt2=0; cnt2 < ocsp_response_counter ;cnt2++){`		
`      r_length`	16	uimsbf
`      ocsp_response()`	8 × r_length	bslbf
`   }`		
`   if (time_stamp_flag == 0x1) {`		
`      domain_timestamp_start`	40	mjdutc
`      domain_timestamp_end`	40	mjdutc
`   }`		
`   /* message header ends here /*`		
`   if (signature_type_flag == 0x0){`		
`      sessionkey_block()`	1 024	bslbf
`   } else if (signature_type_flag == 0x1){`		
`      sessionkey_block()`	2 048	bslbf
`   } else if (signature_type_flag == 0x2){`		
`      sessionkey_block()`	4 096	bslbf
`   }`		
`   /* signature protected part ends here */`		
`   if (signature_type_flag == 0x0){`		
`      signature_block`	1 024	bslbf
`   } else if (signature_type_flag == 0x1){`		
`      signature_block`	2 048	bslbf
`   } else if (signature_type_flag == 0x2){`		
`      signature_block`	4 096	bslbf
`   }`		
`}`		

B.3.4.4.4.2 Updating a domain - domain_update_response() message

B.3.4.4.4.2.1 Message description

Using the 1-pass IRD protocol (refer to clause B.3.4.3.5) the RI sends a domain_update_response() message, informing the device that it left a particular domain. The message is specified below.

Table B.88: message description

domain_update_response()		
parameter name	(M)andatory / (O)ptional (see note)	remark
message_tag	M	global, not encrypted
protocol_version	M	global, not encrypted
longform_udn	M	global, not encrypted
status	M	device specific, not encrypted
device_nonce	M	device specific, not encrypted
certificate_version	M	global, not encrypted
ri_certificate_counter	M	global, not encrypted
c_length	M	global, not encrypted
ri_certificate	M	global, not encrypted
ocsp_response_counter	M	global, not encrypted
r_length	M	global, not encrypted
ocsp_response	M	global, not encrypted
shortform_domain_id	M	device specific, not encrypted
signature_type_flag	M	global, not encrypted
signature_block	M	device specific, not encrypted
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).

If set to 0x0 the format specified in the present document. If set to anything else than 0x0, then the format is beyond the scope of the present document.

longform_udn(): The long form of the UDN. Refer to clause B.3.4.3.6 for details.

status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.89: Status values

Status value	Meaning
Success	The message informs the device that the RI has removed this device from the domain it was registered in. The device SHALL remove the domain keyset that was associated to the particular domain.
NotSupported	The RI does not support the request to leave a domain. The device will use other means to notify the RI that it wants to leave a particular domain.
InvalidDomain	The RI is unable to support the request to leave a domain, because the domain is invalid.

Please refer to clause B.7 for the value of the error codes.

device_nonce: The device_nonce is the nonce which was present in the request (using the offline NSD protocol) to which this message is a response. This nonce is an encoded in BCD.

certificate_version: Is a numerical representation of the version of the RI certificate. Using the certificate_version parameter the customer device can decide if it is needed to update the RI certificate (if it was stored before).

Table B.90: Description of certificate_version parameter

Parameter Fieldname	Field Value ($_h$)	Supports
major_version_number	0x0,..,0xF	MSB_4(certificate_version)
minor_version_number	0x0,..,0xF	LSB_4(certificate_version)

The parameter is divided 2 fields of 4 bits, whereas the first 4 bits (MSB left) express the Major number and the last four bits (LSB right) express the Minor version. The major and minor number encode in bslbf format. 16 Major and 16 Minor versions are supported. For example: Major.Minor version <1.2> is expressed as 0001 0010_b.

ri_certificate_counter - This parameter indicates the depth of the RI certificate chain.

Table B.91

Number of certificate in chain	Value ($_h$)	Remark
0	0x0	Will signal absence of ri_certificate e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 RI certificates.		

c_length: This parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): This parameter SHALL be present. When present, the value of a *ri_certificate* parameter SHALL be a certificate chain including the RI's certificate. The chain SHALL NOT include the root certificate. The RI certificate SHALL come first in the list. Each following certificate SHALL directly certify the one preceding it.

The Device MAY store RI certificate verification data indicating that an RI certificate chain has been verified. The purpose of this is to avoid repeated verification of the same certificate chain. The RI certificate verification data stored in this way SHALL uniquely identify the RI certificate and SHALL be integrity protected. The Device SHOULD check if the RI certificate chain received in this parameter corresponds to the stored certificate verification data for this RI. If so, the Device NEED NOT verify the RI certificate chain again, otherwise the Device SHALL verify the RI certificate chain.

If an RI certificate is received that is not in the stored certificate verification data for this RI, and if the Device can determine (in the case of Broadcast Devices that support DRM Time) that the expiry time of the received RI certificate is later than the RI Context for this RI, and the certificate status of the RI certificate as indicated in the OCSP response is good (see [OCSP-MP]), then the Device SHALL verify the complete chain and SHOULD replace the stored RI certificate verification data with the received RI certificate data and set the RI context expiry time to that of the received RI certificate expiry time.

However, if the Device does store RI certificate verification data in this way it SHALL store the expiry period of the RI's certificate (as indicated by the notAfter field within the certificate) and SHALL compare the Device's current DRM Time with the stored RI certificate expiry time whenever verifying the signature on signed messages from the RI. If the Device's current DRM Time is after the stored RI certificate expiry time then the Device SHALL abandon processing the RI message and SHALL initiate the registration protocol.

ocsp_response_counter: This parameter indicates the depth of the OCSP response chain.

Table B.92

number of responses in chain	Value ($_h$)	remark
0	0x0	Will signal absence of ocsp_response e.g. on error status to save bandwidth.
1	0x1	
2	0x2	
3	0x3	
4	0x4	
5	0x5	
6	0x6	
7	0x7	
NOTE: The certificate chain can have a depth of up to 7 OCSP responses.		

r_length: This parameter indicates the length in bytes of the ocsp_response.

ocsp_response(): This parameter, when present, SHALL be a complete set of valid OCSP responses for the RI's certificate chain. The Device SHALL NOT fail due to the presence of more than one OCSP response element. A Device SHALL check that an OCSP response is present in the received message. If no OCSP response is present in the domain_registration_response() message, then the Device SHALL abort the registration protocol.

shortform_domain_ID: The shortform_domain_id is the SBDF.

signature_type_flag: A flag to signal type of signature algorithm used.

Table B.93

signature_type_flag	Value ($_h$)	Remark
RSA 1 024	0x0	
RSA 2 048	0x1	CMLA requirement (2004-2007)
RSA 4 096	0x2	
Reserved for future use	0x3	Not used in the present document

signature_block: The signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1 024 or RSA-2 048 or RSA-4 096. The signature will apply to the implementation guidelines of PKCS#1, as outlined in clause B.16.

B.3.4.4.4.2.2 Message syntax

Table B.94: message syntax

Fields	Length	Type
`domain_update_response(){`		
`   /* signature protected part starts here */`		
`   message_tag`	8	bslbf
`   protocol_version`	4	bslbf
`   reserved_for_future_use`	4	bslbf
`   longform_udn()`	80	bslbf
`   reserved_for_future_use`	4	bslbf
`   device_nonce`	4	bslbf
`   status`	8	bslbf
`   flags {`		
`      ri_certificate_counter`	3	bslbf
`      ocsp_response_counter`	3	bslbf
`      signature_type_flag`	2	bslbf
`   }`		
`   certificate_version`	8	bslbf
`   for(cnt1=0; cnt1 < ri_certificate_counter ;cnt1++){`		
`      c_length`	16	uimsbf
`      ri_certificate()`	8 × c_length	bslbf
`   }`		
`   for(cnt2=0; cnt2 < ocsp_response_counter ;cnt2++){`		
`      r_length`	16	uimsbf
`      ocsp_response()`	8 × r_length	bslbf
`   }`		
`   shortform_domain_id`	48	uimsbf
`   /* signature protected part ends here */`		
`   if (signature_type_flag == 0x0){`		
`      signature_block`	1 024	bslbf
`   } else if (signature_type_flag == 0x1){`		
`      signature_block`	2 048	bslbf
`   } else if (signature_type_flag == 0x2){`		
`      signature_block`	4 096	bslbf
`   }`		
`}`		

B.3.4.4.4.3 (Force to) Join a domain - join_domain_msg() message

Using the 1-pass IRD protocol (refer to clause B.3.4.3.5) the RI sends a join_domain_msg() message, forcing the device to join a particular domain.

This join_domain_msg() trigger is almost identical to the re_register_msg() message described in clause B.3.4.3.7.3, with the only adaptation being that the message_tag is different. Refer to clause B.19 for the value of the message_tag.

B.3.4.4.4.4 (Force to) Leave a domain - leave_domain_msg() message

Using the 1-pass IRD protocol (refer to clause B.3.4.3.5) the RI sends a leave_domain_msg() message, forcing the device to leave a particular domain.

This leave_domain_msg() trigger is almost identical to the re_register_msg() message described in clause B.3.4.3.7.3, with the only adaptations being that:

- The message_tag is different. Refer to clause B.19 for the value of the message_tag.
- The shortform_domain_id is incorporated, which is the SBDF.

For the message **description** with an explanation of the parameters refer to the re_register_msg() message. For sake of completion the complete leave_domain_msg() message **syntax** is explained below.

B.3.4.4.4.4.1 message syntax

Table B.95: message syntax

Fields	Length	Type
leave_domain_msg() {		
/* signature protected part starts here */		
message_tag	8	bslbf
protocol_version	4	bslbf
reserved_for_future_use	4	bslbf
longform_udn()	80	bslbf
flags {		
signature_type_flag	2	bslbf
ri_certificate_counter	3	bslbf
ocsp_response_counter	3	bslbf
reserved for future use	8	bslbf
}		
shortform_domain_id	48	uimsbf
certificate_version	8	bslbf
for(cnt1=0; cnt1 < ri_certificate_counter ;cnt1++){		
c_length	16	uimsbf
ri_certificate()	8 x c_length	bslbf
}		
for(cnt2=0; cnt2 < ocsp_response_counter ;cnt2++){		
r_length	16	uimsbf
ocsp_response()	8 x r_length	bslbf
}		
/* signature protected part ends here */		
if (signature_type_flag == 0x0){		
signature_block	1 024	bslbf
} else if (signature_type_flag == 0x1){		
signature_block	2 048	bslbf
} else if (signature_type_flag == 0x2){		
signature_block	4 096	bslbf
}		
}		

B.3.4.5 Token handling

B.3.4.5.1 Protocol overview

The theory of operation (refer to clause B.21) results in the specification of several protocols:

- Offline protocols (from device to RI).

Table B.96

Protocol	Clause	Purpose
token request protocol	B.3.4.5.2	request to purchase tokens
token reporting protocol	B.3.4.5.3	protocol to report the consumption of tokens

- 1-pass protocols (from RI to device).

Table B.97

Protocol	Clause	Purpose
1-pass binary Push Device Registration protocol	B.3.4.3.4	transmit registration data to device
1-pass binary Inform Registered Device protocol	B.3.4.3.5	inform device via messages

The protocols interrelate in following way (roundtrip).

Table B.98

Kicking off action…	…results in
token request protocol (request to purchase tokens)	token delivery response message (transmit tokens to device)
token reporting protocol (report the consumption of tokens)	token delivery response message (transmit tokens to device)

B.3.4.5.2 token request protocol

When the user of a device wants to obtain tokens, he uses the NSD protocol with the token_request action type, (refer to clause B.3.4.3.3).

B.3.4.5.3 token reporting protocol

When the user of a device is instructed by his device to report token consumption, he uses the NSD protocol with the token_consumption_message action type in order to send a token consumption report, (refer to clause B.3.4.3.3.4).

B.3.4.5.4 Binary messages

B.3.4.5.4.1 delivering tokens - token_delivery_response() message

B.3.4.5.4.1.1 Message description

Using the 1-pass PDR protocol (refer to clause B.3.4.3.5) the RI sends a token_delivery_response() message, informing the device of the delivery of new tokens. The message is specified below.

Table B.99: Token delivery response message description

token_delivery_response()		
Parameter name	(M)andatory / (O)ptional (see note)	remark
message_tag	M	not encrypted
protocol_version	M	not encrypted
message_length	M	not encrypted
group_size_flag	M	not encrypted
address_mode	M	not encrypted
one	M	not encrypted
address	M	not encrypted
bit_access_mask	not used in the present document	not encrypted
position_in_group	M	not encrypted
domain_id_extension	not used in the present document	not encrypted
domain_generation	not used in the present document	not encrypted
rights_issuer_id	M	not encrypted
status	M	not encrypted
device_nonce	M	not encrypted
response_flag	M	not encrypted
token_reporting_flag	M	not encrypted
earliest_reporting_time_flag	M	not encrypted
latest_reporting_time_flag	M	not encrypted
token_quantity_flag	M	not encrypted
token_delivery_response_id	M	not encrypted
latest_consumption_time	O	not encrypted
earliest_reporting_time	O	not encrypted
latest_reporting_time_flag	O	not encrypted
encrypted_token_quantity	O	encrypted
encrypted_report_authentication_key	O	encrypted
MAC	M	not encrypted
NOTE: (O)ptional means that the user of the message MAY include the parameter in the message, but the device MUST support the interpretation of the parameter. (M)andatory means that the user of the message SHALL include the parameter in the message.		

message_tag: This parameter identifies the type of the message. Refer to clause B.19 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. The device SHALL ignore messages that have a protocol_version number it does not support. The value of the protocol_version of this message is set to 0x0 (i.e. the original format).

If set to 0x0 the format specified in the present document is used. If set to anything else than 0x0, then the format is beyond the scope of the present document.

message_length: 12-bit field indicating the length in bytes of the message starting immediately after this field.

group_size_flag: 1-bit field indicating the group size used. 0 - a maximum group size 256 is used, 1 - a maximum group size of 512 is used.

address_mode**:** 3-bit field indicating the addressing mode used by this message. Table B.100 lists all possible address modes. Note that not all modes are used in the present document for the token delivery response message.

Table B.100 address_mode for token delivery response message

address_mode	Description
0x0	Addressing whole of unique group This addressing mode is not used in the present document for the token delivery response message
0x1	Addressing of subscriber group using a bit_mask size of 256 bit or 512 bit depending on group_size_flag (subset of unique group). This addressing mode is not used in the present document for the token delivery response message
0x2 to 0x3	Addressing of unique device
0x4	Addressing of OMA DRM 2.0 domain. Address field concatenated with the domain_id_extension will be the domain id in this case. This addressing mode is not used in the present document for the token delivery response message
0x5 to 0x7	Reserved for future use

one: 1-bit flag which SHALL have the value 0x1 in the present document. This field MAY have value 0x0 in future versions of the present document.

address: 4-byte group address. Each rights issuer has its own address space. If the group_size is 512 then the group address is made of the first 31 bit of the address field. If this message is addressed to a unique device in a group then the LSB of the address field is the MSB of the group position.

bit_access_mask: This field is not used in the present document and MAY be used in future versions. It is indicated here, so devices according to the present document know its size. All bits of the field SHALL be set to 0, when the field is not used.

position_in_group: If this message addresses a unique device then this field specifies the position of the unique device in the given subscriber group. If group_size_flag is 0 than the position in the group is directly given by the position_in_group field. If group_size_flag is 1 then 9 bit are used to identify the position in the group. If group_size_flag is 1 then the LSB (bit 0) from the address field is used as the 9th bit, the MSB. The real position in the group is then given by:

```
int real_position_in_group;
if(address_mode&0x6==0x2){
    if(group_size_flag == 0){
    //maximum size of 256 devices in group.
    real_position_in_group = position_in_group;
    }else{
        //maximum size of 512 devices in group;
        real_position_in_group = ((address&0x1)<<8)| position_in_group;
    }
}
```

domain_id_extension: This field is not used in the present document and MAY be used in future versions. It is indicated here, so devices according to the present document know its size. All bits of the field SHALL be set to 0, when the field is not used.

domain_generation: This field is not used in the present document and MAY be used in future versions. It is indicated here, so devices according to the present document know its size. All bits of the field SHALL be set to 0, when the field is not used.

rights_issuer_id(): The ID of the rights issuer. This is the 160-bit SHA-1 hash of the public key of the RI. See X509PKISHash in [49].

Status: The status parameter SHALL indicate one of the values explained in the following table. The device SHALL ignore messages with other error values.

Table B.101: message error codes

Status value	Meaning
Success	The message contains valid token delivery data from the RI.
NotSupported	The RI does not support the sending of tokens from the RI. In this message, the RI SHALL set the value of token_quantity to zero or SHALL set the token_quantity_flag to 0x0.
TokenConsumptionMessageError	The RI did receive a token consumption message, but that it was erroneous and that the device should redo the last token consumption message. In this token delivery response message, the RI SHALL set the value of token_quantity to zero or SHALL set the token_quantity_flag to 0x0. The RI SHALL use a token_reporting_flag of value 0x1. The RI SHALL use the device_nonce of the last token consumption message that the RI successfully processed or set the response_flag to 0x0 in case no token consumption messages have been successfully processed. The device SHALL generate a token consumption message, reporting on the token consumption from the time of the generation of the token consumption message with the same device_nonce as the device_nonce in this token delivery response message, or from first start-up in case the response_flag was set to 0x0.
NoTokenConsumptionMessage	The RI did not receive a token consumption message yet, but was expecting one, because the present date/time is later than the last latest_token_consumption_time sent to the device in a token delivery response message. In this token delivery response message, the RI SHALL set the value of token_quantity to zero or SHALL set the token_quantity_flag to 0x0. The RI SHALL use a token_reporting_flag of value 0x1. The RI SHALL use the device_nonce of the last token consumption message that the RI successfully processed or set the response_flag to 0x0 in case no token consumption messages have been successfully processed. The device SHALL generate a token consumption message, reporting on the token consumption from the time of the generation of the token consumption message with the same device_nonce as the device_nonce in this token delivery response message, or from first start-up in case the response_flag was set to 0x0.

Please refer to clause B.7 for the value of the error codes.

device_nonce: If the response_flag equals 0x1, the device_nonce is the nonce present in the request (using the offline NSD protocol) to which this token delivery response message is a response. If the response_flag field equals 0x0, this token delivery response message does not refer to any request from the device to the RI and the device_nonce MAY be ignored. The nonce is encoded in BCD.

response_flag: If this flag equals 0x1, this token delivery response message is a response to a message from the device to the RI and the device_nonce in this token delivery response message is taken from that message. If this flag equals 0x0, this token delivery response message does not refer to any message from the device to the RI and the device_nonce can be any value.

token_reporting_flag: If this flag equals 0x1, the device has to report to the RI the consumption of the tokens received with this token delivery response message. If this flag equals 0x0, the device can consume all tokens delivered with this token delivery response message, as well as any other previously delivered tokens which are still not consumed, without ever having to report their consumption.

earliest_reporting_time_flag: Binary flag to signal presence of the parameter it describes.

Table B.102

earliest_reporting_time field	Value ($_h$) of earliest_reporting_time_flag	Remark
data absent	0x0	
data present	0x1	

latest_reporting_time_flag: Binary flag to signal presence of the parameter it describes.

Table B.103

latest_reporting_time field	Value ($_h$) of latest_reporting_time_flag	Remark
data absent	0x0	
data present	0x1	

token_quantity_flag: Binary flag to signal presence of the parameter it describes.

Table B.104

token_quantity field	Value ($_h$) of token_quantity_flag	Remark
data absent	0x0	
data present	0x1	

token_delivery_response_id: This is the ID of the token delivery response message. The RI SHALL use the same token_delivery_response_id when retransmitting a token delivery response message. The RI SHALL generate a random number using a sufficiently good pseudo random number generator for every new token delivery response message. Devices SHALL discard token delivery response messages with a token_delivery_response_id identical to the one in an already received token delivery response message.

***latest_token_consumption_time*:** After the date/time indicated in the latest_token_consumption_time field, the device SHALL NOT use any tokens, which have been received after the last token delivery response message that had the token_reporting_flag set to 0x0, for the consumption of protected content controlled by the RI. The device SHALL use the date/time in the latest_token_consumption_time field, if present, of the last received token delivery response message, regardless of the value of the field status.

earliest_reporting_time: If the device reports the consumption of tokens before the date/time indicated in the earliest_reporting_time field, the RI NEED NOT change the latest_token_consumption_time in its subsequent token delivery response message.

latest_reporting_time: The purpose of this field is to make uninterrupted token consumption possible. If the device reports the token consumption before the date/time indicated in the latest_reporting_time field, the RI SHALL send the next token delivery response message before the latest_token_consumption_time, unless the RI wishes to interrupt or disable the token consumption.

encrypted_token_quantity: A 4-byte field, containing the encrypted token_quantity. Token_quantity is a signed, 2's complement 32-bit number. If the value of token_quantity is positive, it specifies the number of tokens the device receives from the RI. If the value of token_quantity is negative, it specifies how many tokens the RI removes from the device. If the field encrypted_token_quantity is not present, no tokens are received from the RI and no tokens are removed from the device by this token delivery response message. The token_quantity is encrypted using AES-128-CBC, with fixed IV 0 and with 0 padding in the last block if needed. The encryption key used depends on the addressing mode of the token delivery response message, see table B.105.

Table B.105: Mapping of address_mode to keys for the token delivery response message

address_mode	Key(s) used to decrypt field
0x0 (unique group)	This addressing mode is not used in the present document for the token delivery response message
0x1 (subscriber group)	This addressing mode is not used in the present document for the token delivery response message
0x2 to 0x3 (unique device)	Token Delivery Key
0x4 (domain)	This addressing mode is not used in the present document for the token delivery response message
0x5 to 0x7	reserved for future use

***encrypted_report_authentication_key*:** This field contains the encrypted Report Authentication Key. The Report Authentication Key a 128 bit key to authenticate the reported number of tokens with in the next token consumption message. The encrypted_report_authentication_key field is only present if the token_reporting_flag has the value 0x1. The RI SHALL generate a random number using a sufficiently good pseudo random number generator for the value of every newly required Report Authentication Key. The Report Authentication Key is encrypted using AES-128-CBC, with fixed IV 0 and with 0 padding in the last block if needed. The encryption key used depends on the addressing mode of the token delivery response message, see table B.106.

Table B.106: Mapping of address_mode to keys for the token delivery response message

address_mode	Key(s) used to decrypt field
0x0 (unique group)	This addressing mode is not used in the present document for the token delivery response message
0x1 (subscriber group)	This addressing mode is not used in the present document for the token delivery response message
0x2 to 0x3 (unique device)	Token Delivery Key
0x4 (domain)	This addressing mode is not used in the present document for the token delivery response message
0x5 to 0x7	reserved for future use

***MAC*:** This is the authentication code calculated over all bytes before this field in this message using HMAC-SHA-1-96 (see [23]). The MAC is used for integrity check of this message. The key used to create the MAC is the token delivery response message authentication key TDRMAK as defined in clause B.14.5. Devices SHALL NOT use token delivery response messages with an invalid MAC.

Note Message result:

- More information on device actions after the reception of this message can be found in clause B.21.2.

B.3.4.5.4.1.2 Message syntax

Table B.107: token delivery response message syntax

Fields	Length	Type
`token_delivery_response(){`		
`   /* MAC protected part starts here */`		
`   message_tag`	8	bslbf
`   protocol_version`	4	bslbf
`   message_length`	12	uimsbf
`   group_size_flag`	1	bslbf
`   reserved for future use`	3	bslbf
`   address_mode`	3	uimsbf
`   one`	1	bslbf
`   address`	32	uimsbf
`   if(address_mode == 0x1 && group_size_flag == 0){ (see note)`		
`        bit_access_mask`	256	bslbf
`   }else if(address_mode == 0x1 && group_size_flag == 1){`		
`        bit_access_mask (see note)`	512	bslbf
`   }else if (address_mode&0x6 == 0x2){`		
`        position_in_group`	8	uimsbf
`   }else if (address_mode == 0x4){ (see note)`		
`        domain_id_extension`	6	bslbf
`        domain_generation`	10	uimsbf
`   }`		
`   rights_issuer_id()`	160	bslbf
`   status`	8	bslbf
`   device_nonce`	4	bslbf
`   flags {`		
`      response_flag`	1	bslbf
`      token_reporting_flag`	1	bslbf
`      earliest_reporting_time_flag`	1	bslbf
`      latest_reporting_time_flag`	1	bslbf
`      token_quantity_flag`	1	bslbf
`      reserved for future use`	7	bslbf
`   }`		
`   token_delivery_response_id`	96	bslbf
`   if (token_reporting_flag == 0x1) {`		
`      latest_token_consumption_time`	40	mjdutc
`      if (earliest_reporting_time_flag == 0x1) {`		
`        earliest_reporting_time`	40	mjdutc
`      }`		
`      if (latest_reporting_time_flag == 0x1) {`		
`        latest_reporting_time`	40	mjdutc
`      }`		
`   }`		
`   /* encrypted part starts here`		
`   if(token_quantity_flag == 1){`		

Fields	Length	Type
`encrypted_token_quantity`	32	bslbf
`}`		
`encrypted_report_authentication_key`	128	bslbf
`/* encrypted part ends here`		
`/* MAC protected part ends here */`		
`MAC`	96	bslbf
`}`		
NOTE: Although this addressing mode is indicated here to facilitate future upgrades, this addressing mode is NOT used in the present document. for the token delivery response message.		

B.4 Rights Issuer Services

Rights Issuer Streams are used to carry Registration Layer and Rights Management Layer objects and messages. These include all the messages that are allocated a message tag in clause B.19.

Within this clause, the objects and messages to be carried are referred to as 'objects'.

The data carried by IP Datacast systems is logically divided into services. Each Rights Issuer Service consists of one or more IP streams.

A Rights Issuer Stream SHALL be a distinct IP stream within a service.

Rights Issuer Services SHALL carry only Rights Issuer Streams. It is also allowed for other types of service, including media services, to carry RI Streams. The following types of RI Stream are described:

- Ad-hoc RI Stream.
- Scheduled RI Stream.
- In-Band RI Stream.

Additionally, Rights Issuers MAY use Rights Issuer Streams to deliver messages in any way they require. A Rights Issuer Service MAY contain any number of Rights Issuer Streams.

RI Services SHALL be identified as services in the ESG. RI Services SHALL contain only RI Streams.

All RI Streams forming part of any service SHALL be identified as such in the ESG.

An informative schedule MAY be broadcast for RI Services. Where available, this SHALL be provided as part of the ESG. It is used to indicate times at which data for particular sets of devices or Subscriber Groups will be broadcast. This allows devices to listen to RI Services only when necessary, and will also allow Service Operation Centres to make use of spare network capacity when available; for example, at night.

Where a Rights Issuer broadcasts a complete schedule covering all its registered devices, it MAY have any number of Rights Issuer Services. This schedule SHALL indicate, for each device or group of devices, a single Rights Issuer Service which will be used to deliver objects to that set of devices. Otherwise, Rights Issuers SHALL have exactly one Rights Issuer Service. This requirement allows a device to determine exactly one Rights Issuer Service to which it listens.

The present document aims to allow enough flexibility for operators to fulfil their own requirements for message and Rights Object delivery, and to trade off latency against bandwidth, while also allowing devices to minimize power consumption. To support this, there are no restrictions on which messages can be carried in which type of stream, although the expected mode of operation is described in clause B.4.1.

B.4.1 Expected Mode of Operation

It is foreseen that the system will be used in the following way. However, it is noted that considerable variation in actual operation is possible within the scope of the present document, in order to support the needs of Service Operation Centres and Rights Issuers. Any message can be carried in any RI Service, at the discretion of the Service Operation Centre and Rights Issuer.

- Using the ESG, a device can determine which Rights Issuer Service will be used to deliver messages to it. This service will be used to deliver the messages mentioned above.
- An RI Service can contain a number of streams, some of which carry scheduled data while others carry ad-hoc data. When a device is receiving an RI Service, it will receive all the streams within that service.
- A Scheduled RI Stream carries all the messages that an RI wishes to make available.
  - These messages are grouped in time by device or Subscriber Group. A schedule giving the times at which information for particular sets of devices or Subscriber Groups will be carried is made available in the ESG. Devices need only listen to the service at the times relevant to it.
  - It is expected that all BCROs required by any authorized device to receive a protected service will be carried in a carousel format within a Scheduled RI Stream.
  - Future BCROs will also be carried, to prevent breaks in service when services keys are changed.
  - It is also expected that re-registration messages, domain update messages, etc will be carried.
  - RI Certificate Chain updates can be separately scheduled within the ESG. The mechanism specified in this clause makes it possible for RIs to make these updates available in a scheduled stream alongside other messages, or for a stream to be available which continuously repeats the RI Certificate Chain message.
  - The bandwidth used for individual RI Services can be varied, for example to use any spare capacity that is available at certain times of the day.
- An Ad-hoc RI Stream is used to deliver messages with low latency. In order to receive an Ad-hoc RI Stream, a device will select the relevant RI Service - to do this it could be put into a special mode or have a particular service selected by the user. Examples of messages expected to be carried in an ad-hoc service include:
  - Registration messages, sent directly after a user has registered.
  - BCROs for services that a user has just purchased.
  - Domain control messages.
  - Token delivery messages.
- Additionally, there can also be In-Band RI Streams. These are broadcast as a separate IP stream within, most likely, a media service. These services can be used in whatever way an RI requires, but it is expected that they will carry:
  - BCROs which require immediate delivery, probably to many devices. Examples include BCROs for Free To View services or for free previews.
  - BCROs for content items being delivered within the service.
  - Any message that an RI wishes to make available immediately.

B.4.2 Scheduled RI Stream

In a Scheduled RI Stream, the timing of message broadcast may be scheduled in some way, according to device or Subscriber Groups.

The schedule describes, for each RI Service, blocks of times at which messages are expected to be available for particular ranges of devices or Subscriber Groups. Where provided, it SHALL be available in the ESG (see clause B.22).

Note that although the schedule applies to the whole RI Service, it may be that there are streams within the service that do not follow the schedule - for example, Ad-hoc RI Streams.

It is recommended that a Rights Issuer fulfil the advertised schedule. However, when circumstances require, a Rights Issuer MAY deviate from the schedule that has been broadcast. This MAY cause some devices to miss schedule slots.

A Scheduled RI Service does not have to be available continuously. It could, for example, only be broadcast at night. It is also possible for an RI Service's bandwidth to vary.

B.4.3 Ad-hoc RI Stream

An Ad-hoc Stream is used to carry messages that a Rights Issuer wishes to be sent spontaneously, i.e. with low latency. It is expected that a device will receive this stream when it is in some special registration mode or when the Rights Issuer Service is specifically selected.

B.4.4 In-Band RI Streams within a Media Service

Each protected service MAY contain In-Band RI Streams. When receiving a protected service which has associated In-Band RI Streams, a device SHALL listen to the RI Streams for Rights Issuers with which it is registered when receiving the protected service.

It is expected that In-Band RI Streams will contain:

- Messages that need to be delivered immediately to large numbers of devices. Examples include Rights Objects for free previews or free-to-view services; or
- Rights Objects for content being carried by the service.

Devices SHALL be able to identify In-Band RI Streams within protected services from the ESG.

B.4.5 Broadcast Format of RI Streams

All the objects defined in the present document are carried in Rights Issuer Streams. The format of these streams is defined in this clause.

These streams SHOULD have the following characteristics:

- The bandwidth overhead of the stream format SHOULD be minimized.
- Objects of varying sizes (smaller than, similar to and larger than the size of an IP packet) SHOULD be efficiently carried.
- Devices SHOULD be able to start interpreting the stream at any packet.
- Where packet reception is unreliable or where packets have been reordered, devices SHOULD be able determine which objects have been correctly and completely received.

Note that it is assumed that the underlying IP stack, and the layers below it, will provide all the necessary error detection, and that IP packets received by the service protection system can be assumed to be as transmitted.

B.4.5.1 IP Characteristics

Rights Issuer streams are IP streams advertised in the ESG. The ESG SHALL also carry an identifier for the version of the present document used to generate each RI Stream. The format of the IP packets is UDP [22]. The present document does not specify any limits to the length of these IP packets - this will instead be determined by the underlying network.

Clause B.4.5.2 defines the format of the packets of the RI Stream.

B.4.5.2 RI Stream Packet Format

The Rights Issuer Stream is made up of RI Stream Packets. There SHALL be at most one RI Stream Packet per UDP packet, and each UDP packet SHALL contain only an RI Stream Packet. The length of the RI Stream Packet is determined by the broadcaster.

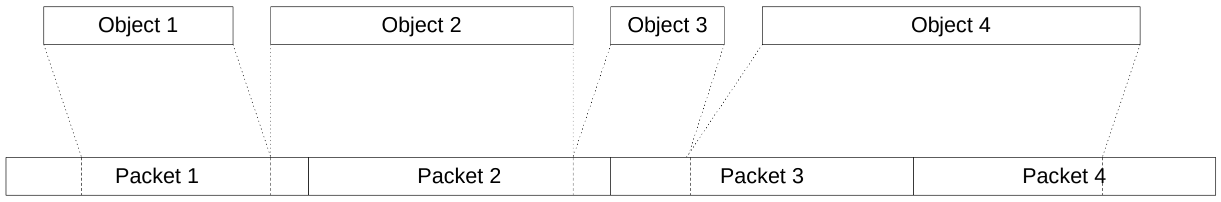


Figure B.40: Example mapping of objects to RI Stream Packets

Objects are placed into packets according to the following rules:

- If the length of an object and its RI Stream header is less than or equal to the remaining empty length of a packet, the object is placed in the packet in its entirety and the split_flag is set to zero.
- If the length of an object is greater than the remaining empty length of a packet:
    - The object is allocated an object_id.
    - The number of packets required to carry the object is calculated, including the remaining space in the current packet. The part of an object to be placed in each packet is hereafter referred to as a fragment.
    - The object is split into the appropriate fragments. Note that fragments will be of varying length, for example, if the first fragment of the object begins part way through a packet.
    - While fragments remain to be carried:
        - A header for each fragment is generated, containing the object ID, the number of this fragment within the object and the total number of fragments in the object. The split_flag is set to one.
- Packets SHALL NOT contain any empty space. The end of the last bytes within a packet carrying information SHALL be the end of the packet and the length field of the UDP and IP packet headers will be filled in appropriately. No padding bytes are allowed as part of this protocol.
- The process is repeated with the next object. The size of each packet can be decided by the Rights Issuer, up to the maximum MTU supported by the network.

The format of the packet is as follows.

Table B.108: Format of the Rights Issuer Stream

Fields	Length	Format
`while(bytes left in packet){`		
`   split_flag`	1	bslbf
`   if(split_flag == 1) {`		
`      object_id`	7	bslbf
`      fragment_number_within_object`	4	bslbf
`      total_number_of_fragments_for_object`	4	bslbf
`      if(fragment_number_within_object == total_number_of_fragments_for_object) {`		
`         reserved_for_future_use`	4	
`         remaining_length_in_packet`	12	bslbf
`      }`		
`      bytes_of_object()`		
`   }`		
`   else{`		
`      reserved_for_future_use`	3	
`      length_of_object`	12	bslbf
`      bytes_of_object()`		
`   }`		
`}`		

split_flag: If 1, this object is split over multiple packets. If 0, this object is completely contained in this packet.

object_id: An identifier for this object. All fragments of an object (that are carried in separate packets) have the same object ID. This is only required for objects that are split over multiple packets. For each split object generated, this object ID SHALL be incremented by 1mod(2^7).

fragment_number_within_object: The number of this fragment within the object.

total_number_of_fragments_for_object: The total number of fragments that make up the object.

remaining_length_in_packet: The length of the remaining bytes of the current object in this packet.

length_of_object: The length of this object (which is completely contained in this packet).

bytes_of_object(): The bytes of the object to be carried in this packet.

B.4.5.3 Implementation notes

B.4.5.3.1 Unreliable delivery

IP networks do not usually offer reliable delivery of packets - this is particularly true of broadcast systems. Devices might not receive all the packets of the RI Stream. Where missing packets cause the device to receive only part of an object, the device SHALL discard this object, although see clause B.4.5.3.2 as apparently missing packets could later be received due to packet reordering.

B.4.5.3.2 Changes in packet order

IP packet order can change between the source and destination hosts on some types of IP network. Of course, this cannot happen on a broadcast link, but it could happen within head-end systems or where this service protection scheme is used over other types of link.

At reception time, it is not possible for a device to tell whether an apparently missing packet has been missed due to a reception problem, or whether it will be later received due to some upstream packet reordering. Consider the situation where three packets 1, 2, 3 are reordered and a device receives them in the order 1, 3, 2. When the packet processing module receives packet number 3, it will appear as if packet 2 has been missed. However, if the device stores packet 3, and then receives and processes packet 2, it can reconstruct all the objects completely contained in all three packets. In order to implement this reconstruction scheme, the device buffers partly received objects for some time, and then reconstruct the whole object if the remainder is later received. Incomplete objects are discarded after some period of time. The limit on the use of this technique and the extent of reordering it can cope with is the amount of buffering provided within a device for partly received objects.

The implementation of any scheme of this kind is not required by the present document.

It is recommended that Service Operations Centres and Rights Issuers minimize changes in packet order within their systems.

B.4.5.3.3 Addressing of objects

The RI Stream Packet format does not contain addressing information for objects. The format of each object includes addressing information relevant to that object. Devices can determine when an object is addressed to the local device or a group of which the device is a member in the following way:

- The device examines the message tag and the version number of the message to determine what type of message is being broadcast.
- The format of the message contains fields addressing the message to devices in some way. These fields are used to determine whether the local device is being addressed.

B.4.6 Mapping of messages to RI services and streams

Within an IP Datacast network, devices discover streams using the ESG and various SI/PSI tables.

- The ESG maps services to IP addresses, allowing a device to discover what services are available, on which IP stream or streams these services are carried and on which IP addresses these streams can be found.
- SI/PSI data describes how a device can receive IP Datacast broadcasts to particular IP addresses, including such information as the PID of the stream carrying the data.

Information about RI Services is carried in the same way as for any other service. RI Services MAY contain any number of IP streams. When receiving a service, a device will receive all the streams that make up that service.

IP Datacast systems typically use a number of multicast streams to transmit data to receiving devices. It is not anticipated that devices will be allocated individual IP addresses that will then be used to address streams to single devices.

The following clauses describe how messages are mapped to services and streams.

B.4.6.1 Rights issuer services with complete schedule information

As mentioned above, Rights Issuers MAY provide a schedule for the broadcast of messages to sets of devices. If a Rights Issuer broadcasts a complete schedule of messages to be sent to all devices (excluding ad-hoc streams), that Rights Issuer MAY have any number of RI Services, containing any number of RI Streams.

For each service, the Rights Issuer SHALL broadcast, within the ESG, one or more schedule items containing a list of devices for which messages may be broadcast on that service, and the times at which those messages will be broadcast. Any device registered with the Rights Issuer SHALL be able to locate a single RI Service to listen to at any one time.

B.4.6.2 Rights issuer services without complete schedule information

If a Right Issuer broadcasts either no schedule information or incomplete schedule information, that Rights Issuer SHALL broadcast only one Rights Issuer Service.

Devices for which schedule information is broadcast SHOULD listen at the appropriate times. Devices for which schedule information is not broadcast SHOULD listen as often to practical, but no requirements are placed on their behaviour.

B.4.7 Discovery of RI services, streams and schedule information

The ESG carries information about RI Services, Streams and their associated schedule information.

The ESG requirements are:

- The capability to describe a service as an RI Service, and have an associated Rights Issuer ID.
- The capability to describe a stream within any service as an RI Stream, and, when the stream is not part of an RI Service, have an associated Rights Issuer ID.
- The capability to describe the version of the specification used to construct an RI Stream.
- The capability to include multiple RI schedule blocks, which may be placed within ESG context or also associated with a purchase channel.
- The capability to indicate that a Certificate Chain update will be included in a particular schedule slot.

For Scheduled RI Services, a number of particular 'content items' are announced, corresponding to device, group or domain ranges (or no range, corresponding to all devices). One of the existing attributes of the content item is for this purpose defined to have a particular structure and semantics.

The broadcast times of these particular content items will be announced within schedule items, and thereby a device will be able to determine when it has to receive the RI service.

B.4.8 Certificate Chain Updates

It is important that devices can acquire Certificate Chain updates, which may include an OCSP response, as quickly as possible. A device will not be able to decode services until it has a current certificate chain (although a grace mechanism is defined in clause B.6.2 to make this more user-friendly). The following requirements are made on the broadcast of Certificate Chain updates.

- It is strongly recommended that schedule information for certificate chain updates is made available in the ESG. When such schedule information is carried, devices SHOULD listen to the relevant RI Services when they need to acquire updates. No firm requirement on device behaviour can be made, as a device may not be able to receive a service at a particular time, for example because it has a low battery or is out of range of the broadcast network.
  - Signalling of certificate chain updates is described in clause B.22.
- Furthermore, it is strongly recommended that a reference to at least the next certificate chain update is always carried in the ESG.
- Where such schedule information is not carried, certificate chain updates SHALL be carried, at least, in the RI Service belonging to the relevant Rights Issuer.

Using the mechanisms described in this clause, two possible schemes for the broadcast of Certificate Chain updates are informatively described below.

- Certificate Chain updates can be broadcast continuously in an RI Stream. A schedule block indicating a certificate chain update, with no device range limit and a time limit of, say, midnight to midnight, is broadcast for this stream, indicating that Certificate Chain updates can always be found on this stream.
- Certificate Chain updates are broadcast periodically on an RI Stream. Schedule blocks indicating a certificate chain update, with no device range limit and the time limit for when the updates will be broadcast, is broadcast for this stream.

B.4.9 Resending of BCROs

There is no guarantee that a device will receive the BCROs sent to it via the broadcast channel. A device may request that the BCROs be sent once again by the Rights Issuer.

B.4.9.1 Resending of BCROs to interactive devices

For an interactive device, requests to resend BCROs can be made via the interactivity channel. If the BCROs are to be delivered via the broadcast channel, the device will listen to the relevant Rights Issuer Service after sending the request. It is recommended that devices listen to this channel for at least one hour, or until the BCROs are received. It is expected that the BCROs will be delivered in an Ad-hoc RI Stream.

When a Rights Issuer receives a request from an interactive device to resend BCROs over the broadcast channel, it MAY resend the BCROs for that device.

B.4.9.2 Resending of BCROs to broadcast devices

Rights Issuers may allow users of broadcast devices to request that BCROs for that device are resent. If the Rights Issuer does allow this, the device may prompt the user to make such a request, as specified in clause B.3.4.3.3. The device SHOULD then listen to the relevant Rights Issuer Service, possibly after the user has acknowledged that the request has been made. It is recommended that devices listen to this channel for at least one hour, or until the BCROs are received. It is expected that the BCROs will be delivered in an Ad-hoc RI Stream.

No firm requirement on device behaviour can be made, as a device may not be able to receive a service at a particular time, for example because it has a low battery or is out of range of the broadcast network.

When the Rights Issuer receives such a request, it MAY resend the BCROs for that user.

B.4.10 Summary of requirements for Rights Issuers

If a Rights Issuer delivers messages to devices via the broadcast channel, it SHALL use Rights Issuer Services and Streams to do so and SHALL meet the requirements below. If a Rights Issuer does not deliver messages via the broadcast channel, it will not have Rights Issuer Services and Streams, and the remainder of this clause does not apply.

- Each Rights Issuer SHALL either:
  - provide a complete schedule for their Rights Issues services, covering all registered devices and allowed any registered device to identify one RI Service to listen to; or
  - have exactly one Rights Issuer Service.
- Each Rights Issuer Service:
  - SHALL NOT contain more than three RI Streams;
  - SHALL contain only Rights Issuer Streams.
- Rights Issuers SHOULD provide an informative schedule for the broadcast of messages in their RI Service, unless the system is being used in an environment where power consumption of devices is not an issue (as the scheduling of RI Services is primarily intended as a power-saving feature for devices).
- Any other type of service MAY carry, at most, one In-Band Rights Issuer Stream per Rights Issuer.
- Rights Issuers SHOULD broadcast both the current and next Rights Objects required to receive services, to reduce the likelihood of a device not having the Rights Object required to receive a service which it is entitled to receive.

B.4.11 Summary of Requirements for Devices

The following is a summary of the requirements relating to RI Services for devices which support the Broadcast mode of operation. Note that none of these requirements apply to devices which only use the interactivity channel to communicate with Rights Issuers.

- For each Rights Issuer with which the device is registered, a device SHALL listen to the associated Rights Issuer Service, subject to the following:
  - Where a schedule for the RI Service is available, devices MAY receive that schedule and MAY listen to the RI Service only at the relevant times:
    - Devices that make use of the schedule SHOULD check for new schedule data at least once per day.
  - Otherwise, when a schedule for the RI Service is not available or a device does not listen to it:
    - Mains powered devices or devices under charge SHOULD listen to that service continuously.
    - Battery powered devices SHOULD listen to that service at least when the device is powered on for some purpose.
- When receiving a Rights Issuer Service, devices SHALL listen to all streams within that service.
- It SHALL be possible to put a device into a mode in which it receives the RI Service of a particular Rights Issuer, for some period, in order to receive, for example, registration data, domain messages and recently purchased Rights Objects. These are expected to be delivered in Ad-hoc RI Streams. This does not apply in the case that a device continuously receives Rights Issuer Services.
- When a device is receiving a service containing an In-Band RI stream for a Rights Issuer with which it is registered, the device SHALL listen to that stream.

B.5 Service Subscription and Purchase

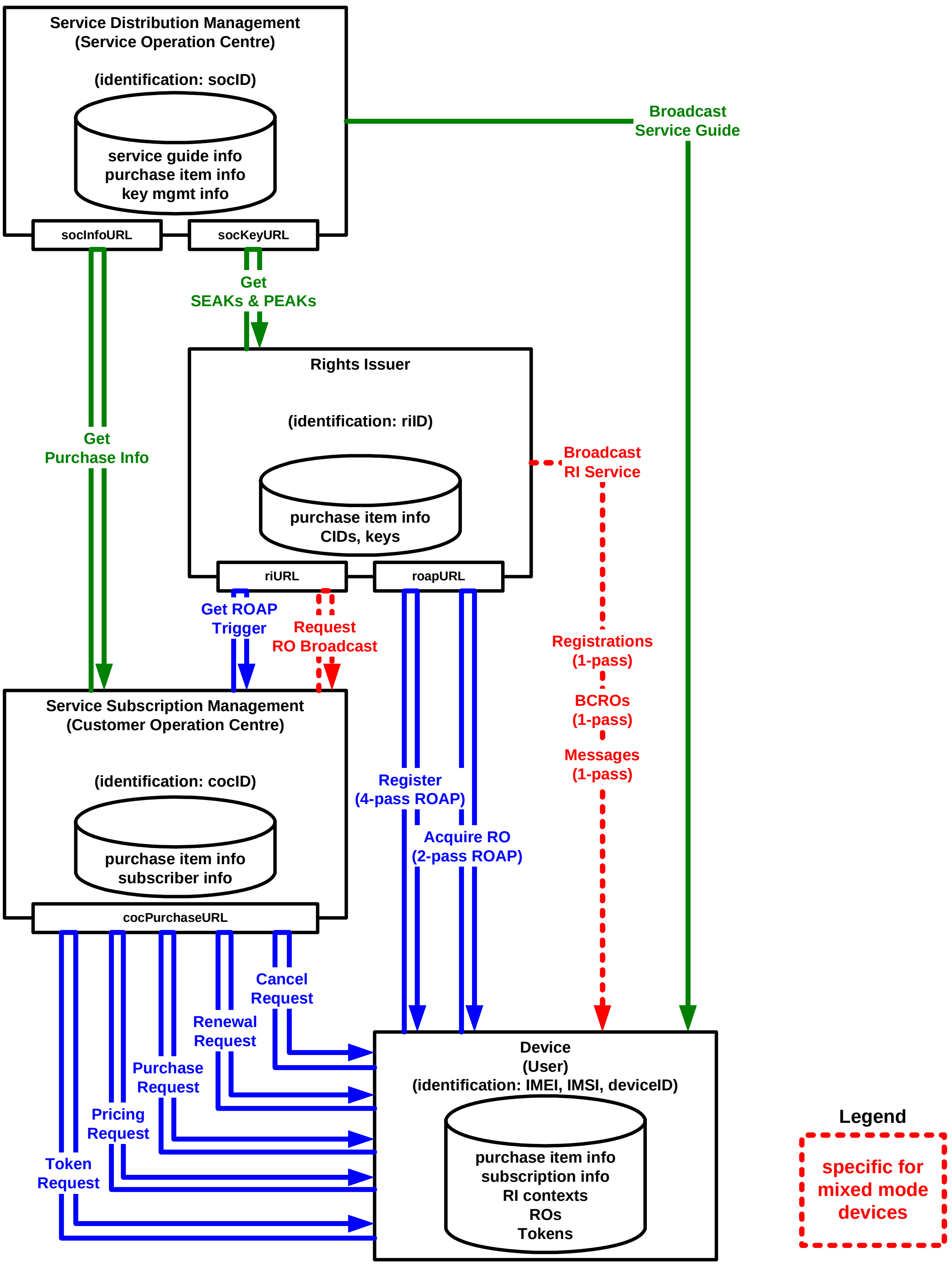


Figure B.41: Message Flows for Service Subscription and Purchase for the connected mode of operation

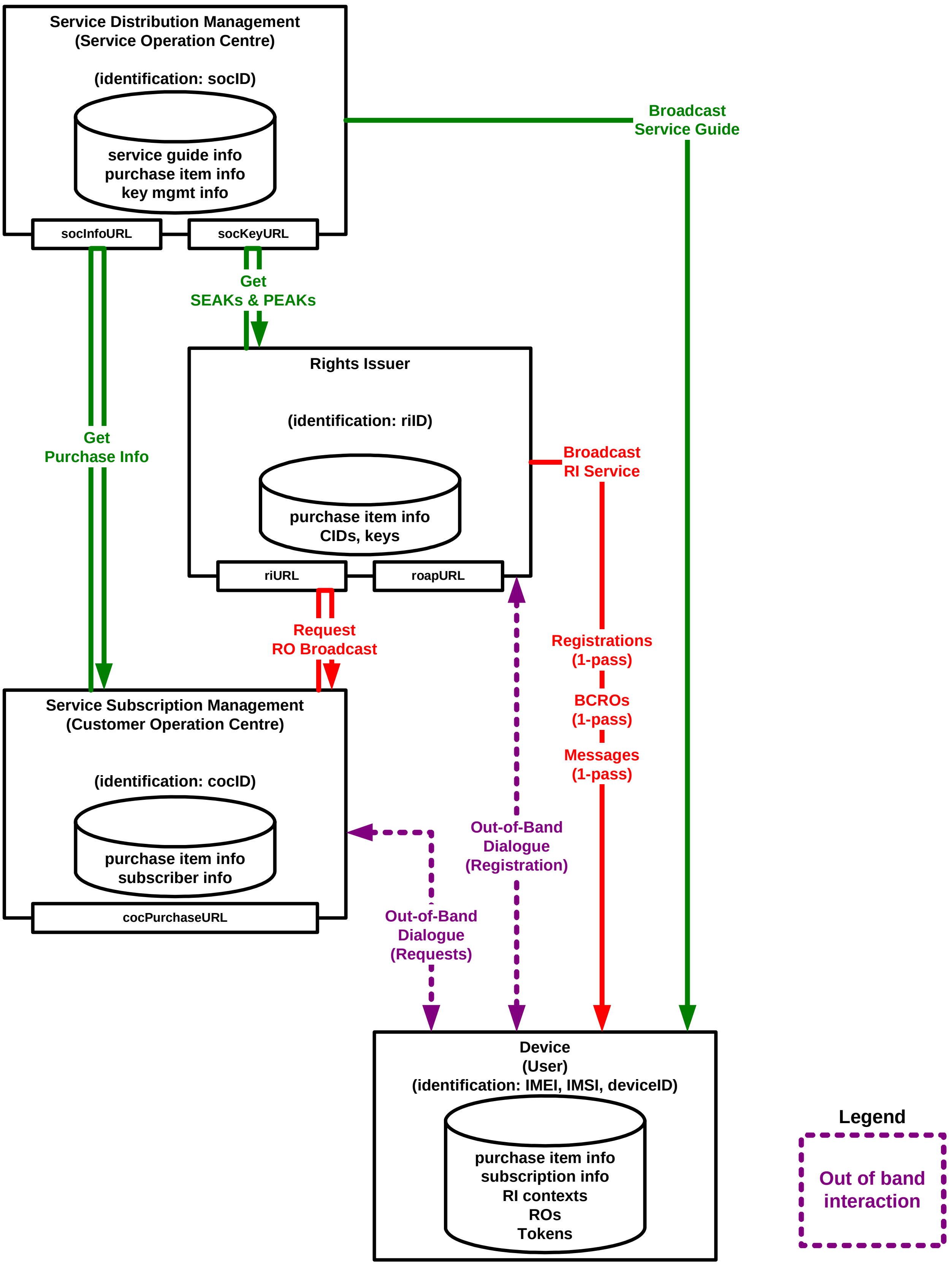


Figure B.42: Message Flows for Service Subscription and Purchase for the unconnected mode of operation

B.5.1 Purchase over the interactivity channel

Interactive devices MAY use the interactivity channel for purchase transactions. The protocol and messages of these purchase transactions are specified later in this clause.

A device supporting purchase over the interactivity channel SHALL implement the protocol for communication with the network elements (SOC, COC, RI) defined in this clause. However, the messages exchanged between network elements are listed here only as an informative reference.

Messages specified in this clause apply also to mixed-mode devices. However for mixed-mode devices the RI MAY choose to deliver a BCRO instead of an ICRO (see clause B.5.2).

B.5.1.1 Typical purchase sequences

Table B.109

Entity Definitions for Interaction Diagrams	
DIST Mgmt	**Service Distribution Management, part of Service Operation Centre (SOC)** The distribution management system is the system from which the device is receiving broadcast services, and is therefore always local to the device's current position, whether the device is located at home or roaming
SUB Mgmt	**Service Subscription Management, part of Customer Operation Centre (COC)** The user is assumed to have a contractual relationship with a home COC (there are no restrictions regarding how many COCs the user may have a contractual relationship with; in case there is more than one, the user needs to choose from which COC to purchase access to a particular service). The operators of the home COC and the SOC (home or visited) need to have a contract ('roaming agreement') in place.
RI	**Rights Issuer (can be part of SOC or COC)** The rights issuer is the server-side DRM implementation, and can be assumed to be part of either the SOC or part of the COC (from architecture point of view, no assumption shall be made, and also a stand-alone rights issuer implementation should be enabled).
OCSP	**OCSP Responder** The OCSP responder is the external certification authority that is able to certify DRM-related device credentials.
Device	**Device** The device is assumed to be a mobile broadcast device with interaction capabilities ('interactive device'). The device is further assumed to have an OMA DRM 2.0 compliant DRM agent, which supports the broadcast extensions as specified in the present document. If it is a device supporting mixed-mode registration, it is also assumed to be able to receive BCROs via the broadcast channel.

B.5.1.1.1 Bulk download of service and program keys

In cases where the RI is part of the SOC, the download of service and program keys from the key generator within Service Distribution Management to the RI may be implemented as a periodical 'bulk download'.

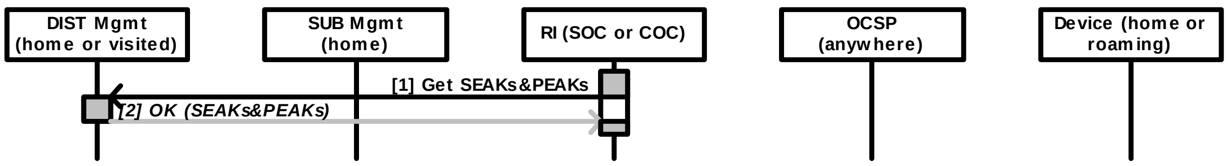


Figure B.43: Interactions for Bulk Download of Service and Program Keys

Table B.110

1	**Get SEAKs and PEAKs** The SEAKs and PEAKs are downloaded from the Service Distribution Management to the RI, together with the purchase item identification. SEAKs and PEAKs have a time span of validity (indicating during which time interval they are effectively used to protect broadcast traffic). Each SEAK or PEAK can be associated with one or more purchase items and each purchase item can be associated with multiple SEAKs or PEAKs. With each purchase item, there may be some information regarding usage rules specified, which influence the generation of the rights expression by the RI. Recommended content of the message: • time interval for which associations of purchase items and their respective SEAKs and PEAKs should be returned *This message is network-internal, and is not further specified.*
2	**OK (SEAKs and PEAKs)** The answer to a "Get SEAKs and PEAKs" request contains a set of associations between purchase item identification and the corresponding SEAKs or PEAKs. Recommended content of the message: • list of purchase item associations, containing: - purchase item ID - time span of validity of contained SEAKs and PEAKs - list of key records, containing ▪ CID ▪ SEAK or PEAK - usage rule info *This message is network-internal, and is not further specified.*

B.5.1.1.2 Bulk download of purchase information

In cases where the COC is 'local' to the SOC, the download of purchase and bundling information from the SOC to the COC(s) providing the broadcast services and content to the users may be implemented as a periodical 'bulk download'.

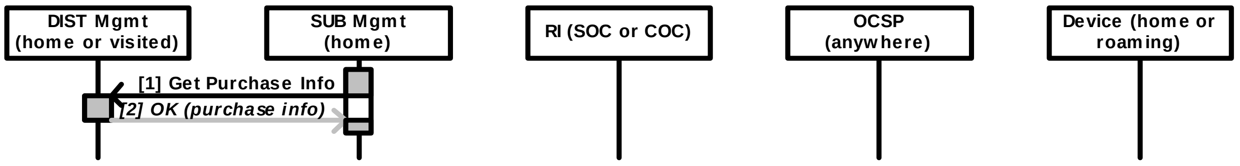


Figure B.44: Interactions for Bulk Download of Purchase Information

Table B.111

1	**Get Purchase Info** Information about purchase items (e.g. service bundles) and the items contained in the purchase item (services, schedule items, content items) is fetched from the SOC (its Service Distribution Management) to the COC (its Service Subscription Management); the information how to bundle multiple items into a single purchase item may originate from the Service Subscription Management, but this message still makes sense, because the identifiers of purchase items are assumed to be managed by the Service Distribution Management and synchronized with the identifiers in the service guide. In the case that items that can be subscribed to (service bundles), the subscription options may be indicated to the Service Subscription Management. Recommended content of the message: • time interval for which associations of purchase items and their respective composition and purchase options should be returned *This message is network-internal, and is not further specified.*
2	**OK (purchase info)** The answer to the "Get Purchase Info" request contains the purchase info of all purchase items that can currently be sold by the Service Subscription Management. Recommended content of the message: • list of purchase item associations, containing: - purchase item ID - flag whether or not re-keying is used (indicating a subscription) - list of purchase item records, containing (for a service or schedule item or content item) ▪ ID ▪ name (multi-language) ▪ description (multi-language) - list of purchase option records, containing ▪ purchase option ID ▪ purchase option description (multi-language) ▪ purchase option price (incl. currency) The list of purchase item records is not strictly necessary for Service Subscription Management to obtain. However, it can be assumed that for purposes of customer care it will be important for the Service Subscription Management to know what items are included in a purchase item. *This message is network-internal, and is not further specified.*

B.5.1.1.3 Announcement of Purchase Items in Service Guide

The purchase items that relate to the services that are broadcast in the network controlled by the Service Distribution Management are listed in the service guide. In the case that multiple services, schedule items or content items are bundled into a single purchase item, this bundling information SHALL be part of the service guide. This allows a user to decide which bundle to purchase in order to get access to that particular item.

Should not all purchase items be obtainable from a particular COC (through its Service Subscription Management), the information about which purchase items can be obtained from which COC may be included in the service guide.

In the case that a particular purchase item may be obtained via multiple purchase options (e.g. 1 month subscription, 12 months subscription, renewal option), the service guide should list these purchase options.

The service guide should include COC-specific price indications for all purchase options.

It is assumed that the service guide will contain availability and pricing information only for 'local' COCs. In the case of roaming, the device will have the following options:

a) to inquire this information from its home Service Subscription Management (and execute the purchase via its home COC);

b) to establish a contractual relationship with one of the local COCs who thereby becomes a home COC.

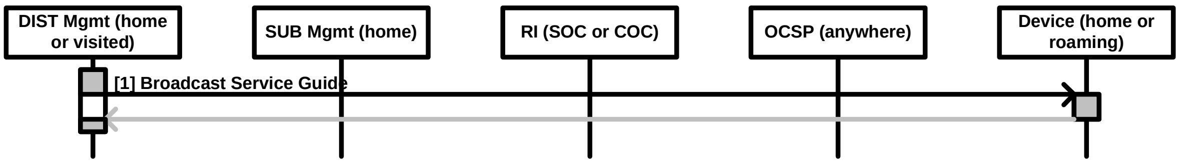


Figure B.45: Interactions for Announcement of Purchase Items in Service Guide

Table B.112

1	**Broadcast Service Guide** The service guide is assumed to be broadcast, and to contain the purchase-relevant data as specified further down (whereas distribution over the broadcast channel is assumed to be the normal case, the service guide may also be retrievable over the interactivity channel). *The service guide is not in scope of the present document, however.*

B.5.1.1.4 Pricing inquiry

In the case that the device intends to buy a purchase item that is announced in the service guide, but the service guide:

a) contains no information about the home COC of the device; or

b) the service guide lacks availability and/or pricing information about the desired purchase item in relation to the home COC;

then the device needs to inquire the availability and pricing information from its home COC prior to making a purchase request.

This pricing inquiry is adding an additional interaction between the device and the COC. In order to improve usability and reduce interaction traffic in the 'normal' case of home service consumption, the service guide should therefore contain all pricing-relevant information concerning the local COCs.

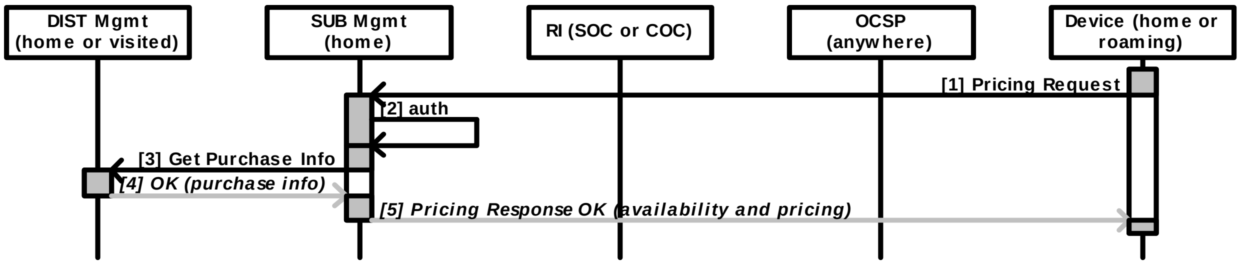


Figure B.46: Interactions for Pricing Inquiry

Table B.113

1	**Pricing Request** The device may send a pricing request to its home COC. The pricing request refers to a purchase item. In order for the COC to further get the pricing information from the SOC (through its Service Distribution Management), the SOC needs to be identified. The pricing request may include an indication of the user's preferred language. *This message is further specified in this clause.*
2	authenticate Before doing any further processing, the COC authenticates the device and/or user, using some or all of the credentials that are part of the purchase request, or even some credentials that can be supplied by the cellular network, such as MSISDN. *This operation is network-internal, and is not further specified.*
3	**Get Purchase Info** In case the COC does not do 'bulk download' of purchase and pricing information from the SOC, and very typically in the case that the COC has a different geographical location (e.g. different country) than the SOC, the COC inquires availability and pricing information from the SOC specifically for the requested purchase item. Recommended content of the message: • purchase item ID • preferred language *This message is network-internal, and is not further specified.*
4	**OK (purchase info)** In the case that the purchase item requested by the device is valid, the SOC returns its purchase and pricing information (e.g. there may be multiple purchasing options with different pricing) back to the COC. Recommended content of the message: • purchase item ID • flag whether or not re-keying is used (indicating a subscription) • list of purchase item records, containing (for a service or schedule item or content item) - ID (of service or schedule item or content item) - name (user-specified or default language) - description (user-specified or default language) • list of purchase option records, containing - purchase option ID - purchase option description (user-specified or default language) - purchase option price (incl. currency) The list of purchase item records is not strictly necessary for COC to obtain. However, it can be assumed that for purposes of customer care it will be important for the COC to know what items are included in a purchase item. *This message is network-internal, and is not further specified.*
5	**Pricing Response (availability and pricing)** Upon reception of the availability and pricing information from the SOC, the COC may, based on its own logic and pricing rules, modify the pricing and decide whether or not to return a successful pricing response to the device. The successful pricing response lists all pricing options and their prices. The text within the pricing response may be in the user's preferred language, if supported by SOC, otherwise in a default language. *This message is further specified in this clause.*

B.5.1.1.5 Unsuccessful purchase

Establishing a contractual relationship (commonly called 'subscription', and not to be confused with the subscription of a particular service or service bundle) between a user and a Service Subscription Management is not in scope of the present document.

Even after a contractual relationship is established, the first purchase might fail, because no device registration has yet been taken place. The following interactions illustrate this scenario which might be so common that it cannot be treated as an exception.

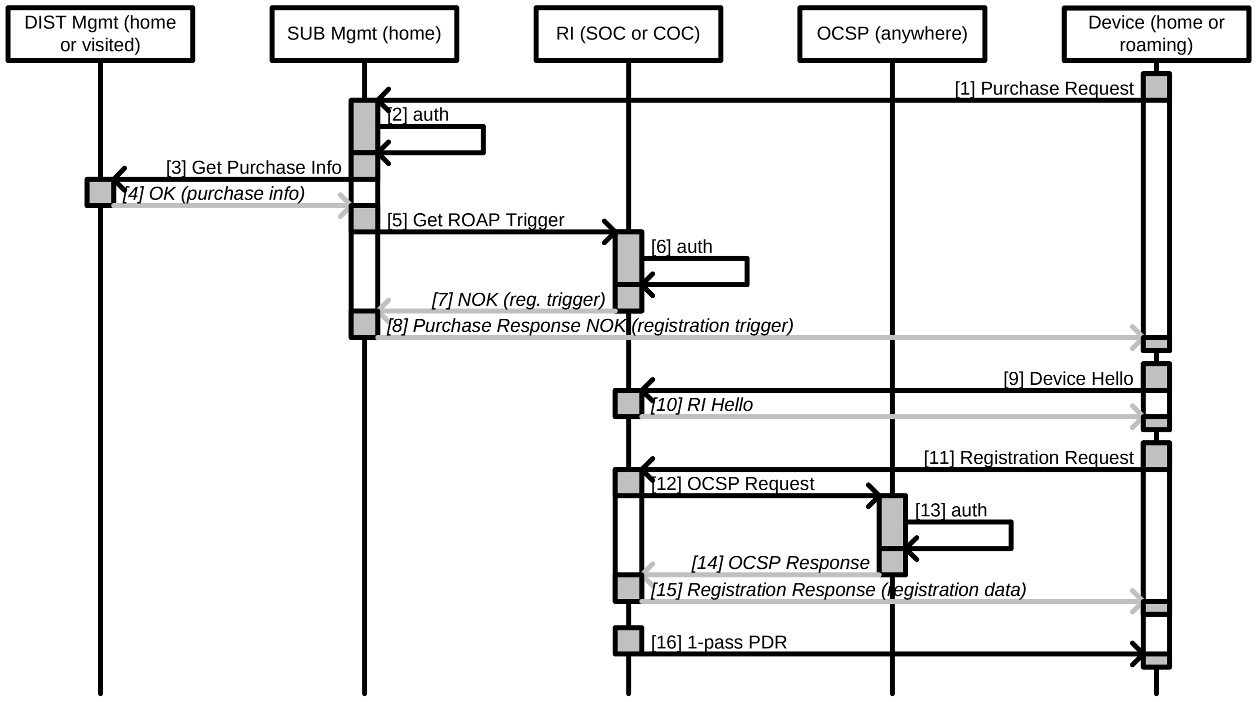


Figure B.47: Interactions for Unsuccessful Purchase

Table B.114

1	**Purchase Request** In order to initiate a purchase (this can be a one-off purchase of a content item or of a schedule item, or a subscription of a service or service bundle), the device sends a purchase request to its home Service Subscription Management (or one of its home Service Subscription Management, if there are multiple of them). Prior to sending the purchase request, the device should have established the availability of the purchase item from the selected Service Subscription Management, and its pricing for all purchase options. By sending the purchase request, the device accepts the pricing. *This message is further specified in this clause.*
2	**authenticate** Before doing any further processing, the Service Subscription Management authenticates the device and/or user, using some or all of the credentials that are part of the purchase request, or even some credentials that can be supplied by the cellular network, such as MSISDN. The device certificate may be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
3	**Get Purchase Info (optional step)** In the case that the Service Subscription Management does not do 'bulk download' of purchase and pricing information from the Service Distribution Management, and very typically in the case that the Service Subscription Management has a different geographical location (e.g. different country) than the Service Distribution Management, the Service Subscription Management inquires availability and pricing information from the Service Distribution Management specifically for the requested purchase item. Recommended content of the message: • purchase item ID • preferred language *This operation is network-internal, and is not further specified.*
4	**OK (purchase info)** In the case that the purchase item requested by the device is valid, the Service Distribution Management returns its purchase and pricing information (e.g. there may be multiple purchasing options with different pricing) back to the Service Subscription Management. Recommended content of the message: • purchase item ID • flag whether or not re-keying is used (in case of subscription, 'yes', otherwise, 'no') • list of purchase item records, containing (for a service or schedule item or content item) - ID (of service or schedule item or content item) - name (user-specified or default language) - description (in user-specified or default language) • list of purchase option records, containing - purchase option ID - purchase option description (in user-specified or default language) - purchase option price (incl. currency) The list of purchase item records is not strictly necessary for Service Subscription Management to obtain. However, it can be assumed that for purposes of customer care it will be important for the Service Subscription Management to know what items are included in a purchase item. *This operation is network-internal, and is not further specified.*

5	**Get ROAP Trigger** The Service Subscription Management requests a ROAP trigger from the RI. The request may also include (part of) the rights expression that will be included in the RO. It is up to the Service Subscription Management to decide which RI to use (e.g. this can be the own RI of the Service Subscription Management, or the RI of the Service Distribution Management). Recommended content of the message: • RI device ID • RI domain ID (optionally used in case the device requests the RO to be valid for a broadcast domain) • purchase item ID (which the RI can use to identify all the keys to pack into the RO) • socID • socKeyURL • user-specific rights expression (which the RI puts into the RO, in accordance with rules defined by Service Distribution Management; if defined, these will have to be observed by the device in addition to any post-acquisition usage rules) • current/next flag set to 'current' (indicating to the RI that the 'current' keys SHALL be put into the RO) *This operation is network-internal, and is not further specified.*
6	**authenticate** Before doing any further processing, the RI authenticates the device. *This operation is network-internal, and is not further specified.*
7	**NOK (reg. trigger)** In case the device is found to have no valid registration (e.g. never registered, or the registration expired or is not trusted anymore), a negative response is sent back to the Service Subscription Management, containing a registration trigger that instructs the device to register. Recommended content of the message: • trigger for device registration (the trigger includes the rights issuer URL to which the registration can be initiated). *This operation is network-internal, and is not further specified.*
8	**Purchase Response NOK (registration trigger)** The negative response from the RI, including the registration trigger, is forwarded to the device. The ROAP trigger includes the URL of the RI that the device is supposed to register with. *This message is further specified in this clause.*
9	**Device Hello** Using the information contained in the trigger, the device initiates the 4-pass device registration. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
10	**RI Hello** *This is a standard OMA DRM 2.0 operation, and is not further specified.*
11	**Registration Request** *This is a standard OMA DRM 2.0 operation, and is not further specified.*
12	**OCSP Request** *This is a standard OMA DRM 2.0 operation, and is not further specified.*
13	**authenticate** Before doing any further processing, the OCSP authenticates the device. *This operation is network-internal, and is not further specified.*
14	**OCSP Response** *This is a standard OMA DRM 2.0 operation, and is not further specified.*
15	**Registration Response** As a result of a successful registration, the registration data is securely stored in the device and ready for subsequent use. The device can now re-initiate the purchase transaction. The RI MAY then include a join domain trigger with the ROAP-RegistrationResponse as an additional MIME part of the payload. This will result in the 2-pass ROAP registration for the OMA DRM 2.0 domain. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
16	**1-Pass PDR (mixed-mode devices only)** For a mixed-mode device, the RI MAY decide to send the device_registration_response message, which contains the registration data for the broadcast mode of operation (see clause B.3.4.3.4).

B.5.1.1.6 Successful purchase

This sequence is executed if a device is already registered with the RI at the moment when it initiates the purchase transaction. It is also assumed that all authentication steps are carried out successfully, and the purchase can be successfully charged.

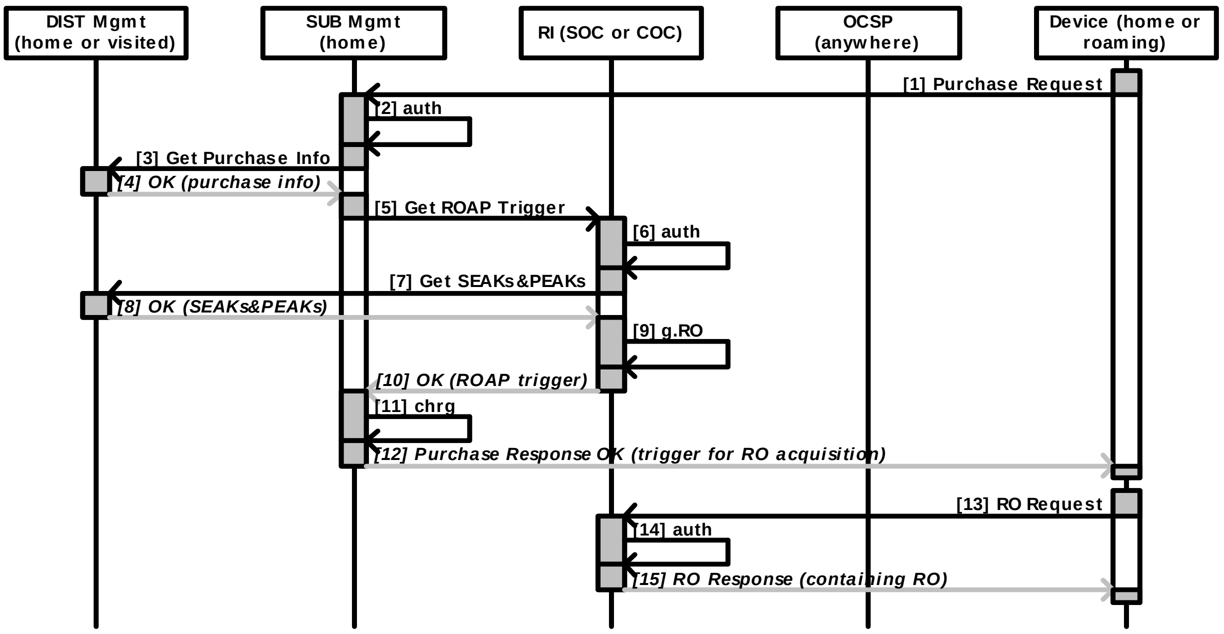


Figure B.48: Interactions for successful purchase

Table B.115

1	**Purchase Request** In order to initiate a purchase (this can be a one-off purchase of a content item or of a schedule item, or a subscription of a service or service bundle), the device sends a purchase request to its home Service Subscription Management (or one of its home Service Subscription Management, if there are multiple of them). Prior to sending the purchase request, the device should have established the availability of the purchase item from the selected Service Subscription Management, and its pricing for all purchase options. By sending the purchase request, the device accepts the pricing. *This message is further specified in this clause.*
2	**authenticate** Before doing any further processing, the Service Subscription Management authenticates the device and/or user, using some or all of the credentials that are part of the purchase request, or even some credentials that can be supplied by the cellular network, such as MSISDN. The device certificate may be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
3	**Get Purchase Info (optional step)** In the case that the Service Subscription Management does not do 'bulk download' of purchase and pricing information from the Service Distribution Management, and very typically in the case that the Service Subscription Management has a different geographical location (e.g. different country) than the Service Distribution Management, the Service Subscription Management inquires availability and pricing information from the Service Distribution Management specifically for the requested purchase item. Recommended content of the message: • purchase item ID • preferred language *This operation is network-internal, and is not further specified.*
4	**OK (purchase info)** In case the purchase item requested by the device is valid, the Service Distribution Management returns its purchase and pricing information (e.g. there may be multiple purchasing options with different pricing) back to the Service Subscription Management. Recommended content of the message: • purchase item ID • flag whether or not re-keying is used (in case of subscription, 'yes', otherwise, 'no') • list of purchase item records, containing (for a service or schedule item or content item) - ID (of service or schedule item or content item) - name (user-specified or default language) - description (user-specified or default language) • list of purchase option records, containing - purchase option ID - purchase option description (in user-specified or default language) - purchase option price (incl. currency) The list of purchase item records is not strictly necessary for Service Subscription Management to obtain. However, it can be assumed that for purposes of customer care it will be important for the Service Subscription Management to know what items are included in a purchase item. *This operation is network-internal, and is not further specified.*

5	**Get ROAP Trigger** The Service Subscription Management requests a ROAP trigger from the RI. If the request includes any rights expression, they will be included in the RO. It is up to the Service Subscription Management to decide which RI to use (e.g. this can be the own RI of the Service Subscription Management, or the RI of the Service Distribution Management). Recommended content of the message: • RI device ID • RI domain ID (optionally used in case the device requests the RO to be valid for a broadcast domain) • purchase item ID (which the RI can use to identify all the keys to pack into the RO) • socID • socKeyURL • user-specific rights expression (which the RI puts into the RO, in accordance with rules defined by Service Distribution Management; if defined, these will have to be observed by the device in addition to any post-acquisition usage rules) • current/next flag set to 'current' (indicating to the RI that the 'current' keys shall be put into the RO) *This operation is network-internal, and is not further specified.*
6	**Authenticate** Before doing any further processing, the RI authenticates the device. In this example of a purchase sequence, the device is found to have a valid registration (RI context). *This operation is network-internal, and is not further specified.*
7	**Get SEAKs and PEAKs (optional step)** In the case that there is no 'bulk download' of associations between purchase items and the corresponding keys (service keys, program keys) from the Service Distribution Management to the RI, the RI requests the necessary keys by sending the purchase item identification to the Service Distribution Management. Recommended content of the message: • purchase item ID • current/next flag set to 'current' (indicated that the 'current' keys are requested) *This operation is network-internal, and is not further specified.*
8	**OK (SEAKs and PEAKs)** The service and program keys corresponding to the purchase item as specified in the request are returned by the Service Distribution Management to the RI. There may be some information regarding usage rules specified, which influence the generation of the rights expression by the RI. Recommended content of the message: • purchase item ID • time span of validity of contained SEAKs or PEAK • list of key records, containing - CID - SEAK or PEAK • usage rule info *This operation is network-internal, and is not further specified.*
9	**generate RO** The RI generates the RO containing the desired keys (e.g. multiple SEAKs in case of a subscription to a service bundle, or a PEAK in case of pay-per-view of a service). In the case that the Service Subscription Management has specified some particular usage rights, these are included in the rights expression of the RO, under consideration of the usage rules that may have been specified by the Service Distribution Management. *This operation is network-internal, and is not further specified.*

10	**OK (ROAP trigger)** In the case of success, the RI sends an OK message back to the Service Subscription Management, containing also the ROAP trigger that the device may use to request the prepared RO. Recommended content of the message: • trigger for the rights object acquisition (the trigger includes the rights issuer URL from which the RO can be acquired by the device). The trigger MAY be omitted for mixed-mode devices (see clause B.5.2). The device certificate may also be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
11	**Charge** Before sending the RO acquisition trigger back to the device, the Service Subscription Management may charge the user (if consumption-based charging based on metering is used, the user is not charged when the 'purchase request' is made, but when tokens are requested). How charging is done (e.g. credit card, direct bank debit, generation of charging record to be included in a periodical invoice) is completely up to the implementation and to the contract between the user and Service Subscription Management. *This operation is network-internal, and is not further specified.*
12	**Purchase Response OK (trigger for RO acquisition)** As part of the positive response to the purchase request, the trigger for RO acquisition is sent to the device. The ROAP trigger includes the URL of the RI that the device can use to retrieve the prepared RO. Alternatively, if the device is a mixed-mode device the RI MAY decide to deliver a BCRO over the broadcast channel. In this case the ROAP trigger will not be delivered to the device and the flow continues with the delivery of a BCRO (see clause B.5.2). If the request was for a domain RO, and the device has not yet joined the domain, the RI MAY include a join domain trigger as an additional MIME part of the response payload. The join domain operation is not described in this sequence. *This message is further specified in this clause.*
13	**RO Request** Using the information contained in the ROAP trigger, the device initiates the 2-pass ROAP. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
14	**authenticate** Before doing any further processing, the RI authenticates the device and/or user. *This operation is network-internal, and is not further specified.*
15	**RO Response** As a result of a successful RO acquisition, the RO is available in the device. *This is a standard OMA DRM 2.0 operation, and is not further specified.*

B.5.1.1.7 Subscription RO Renewal and Asynchronous Charging

The ROs for subscriptions will have to be periodically renewed. It can be expected that the lifetime of a Subscription RO will be on the order of 1 day to 1 month, and no longer than the subscription period. A shorter lifetime means higher security at the expense of more processing and bandwidth usage.

Renewing a subscription RO can be understood as 're-keying of the service key'. The purpose is to provide the device with the 'next service key' for all services that a device or user is already subscribed to, and may already have paid for. The need for authentication by the Service Subscription Management makes the RO renewal request very similar in nature to the normal purchase request, and the data flows are largely identical.

The device MAY initiate RO renewal any time during the lifetime of the 'current' RO. The lifetime of the RO is signalled in form of a 'datetime' restriction to the 'access' permission of the RO. In order to avoid all devices to renew their ROs at the same time, the following random delay mechanism SHALL be used to spread renewal over the whole renewal period.

T1 = the point in time when all of the SEAKs in the current RO first became active

T2 = the point in time when all of the SEAKs in the current RO are due to expire

DT = a device SHALL request the next RO within DT of the current RO's expiry (implementation-dependent, and big enough to ensure that the device gets the next RO timely). It is expected that DT is signalled in the service guide.

T1 and T2 define the time interval in the 'datetime' restriction in the 'access' permission in the RO. The device SHALL request the next RO at a random point in time T, where T1 <= T <= T2-DT.

In case of open-ended subscriptions, it is expected that the Service Subscription Management will periodically charge the user. If this is the case, the charging operation SHOULD be completely separated from the RO renewal, and MAY continue until cancellation of the subscription, whether the device renews the related RO or not.

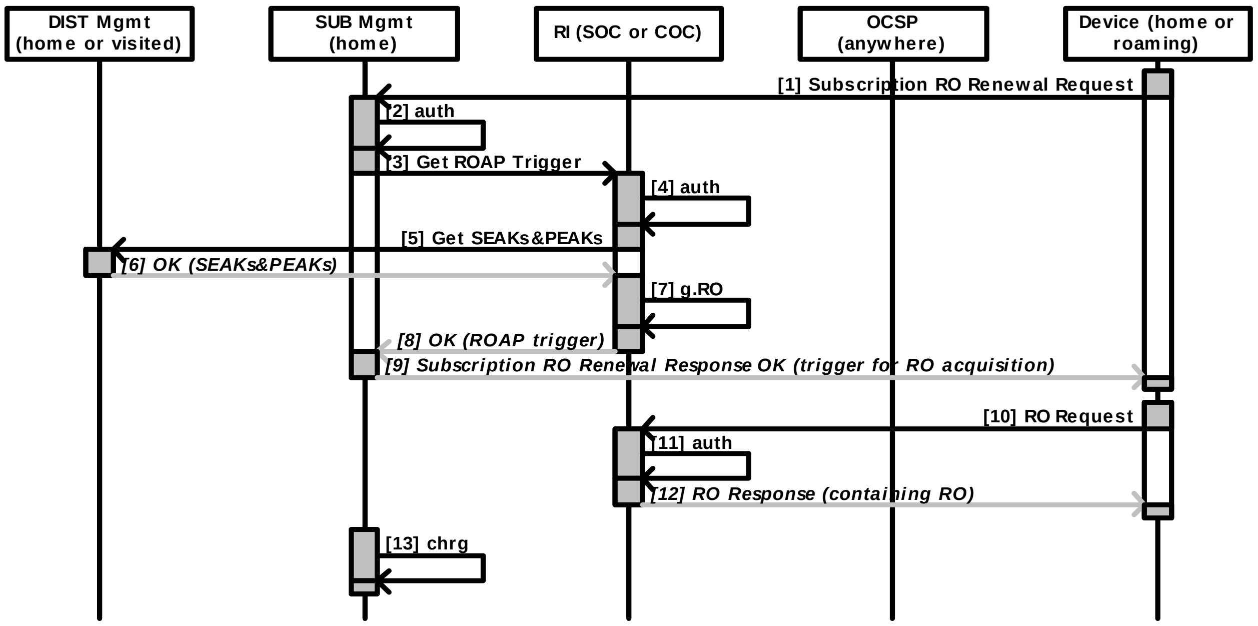


Figure B.49: Interactions for Subscription RO Renewal and Asynchronous Charging

Table B.116

1	**Subscription RO Renewal Request** The device sends the renewal request to the same Service Subscription Management from where it originally purchased the subscription. This allows the Service Subscription Management to check that the renewal request indeed corresponds to a valid subscription. In order to signal to the Service Subscription Management which RO to return, the renewal request contains the expiry time of the current RO. If no expiry time is given, then the Service Subscription Management will assume that the current RO has been lost, and treat the request as a replacement request for the current RO. *This message is further specified in this clause.*
2	**authenticate** Before doing any further processing, the Service Subscription Management authenticates the device and/or user, using some or all of the credentials that are part of the purchase request, or even some credentials that can be supplied by the cellular network, such as MSISDN. The device certificate may be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
3	**Get ROAP Trigger** The Service Subscription Management requests a ROAP trigger from the RI. The request may also include (part of) the rights expression that will be included in the RO. The Service Subscription Management will use the same RI as for the original purchase request. Recommended content of the message: • RI device ID • RI domain ID (optionally used in case the device requests the RO to be valid for a broadcast domain) • purchase item ID (which the RI can use to identify all the keys to pack into the RO) • socID • socKeyURL • user-specific rights expression (which the RI puts into the RO, in accordance with rules defined by Service Distribution Management; if defined, these will have to be observed by the device in addition to any post-acquisition usage rules) • current/next flag set to 'next' (indicating to the RI that the 'next' keys SHALL be put into the RO) *This operation is network-internal, and is not further specified.*
4	**authenticate** Before doing any further processing, the RI authenticates the device. *This operation is network-internal, and is not further specified.*
5	**Get SEAKs and PEAKs (optional step)** In the case that there is no 'bulk download' of associations between purchase items and the corresponding keys (service encryption and authentication keys, program encryption and authentication keys) from the Service Distribution Management to the RI, the RI requests the necessary service encryption and authentication keys by sending the purchase item identification to the Service Distribution Management. Importantly, the Service Subscription Management will specify whether the 'current' or 'next' keys are desired. Recommended content of the message: • purchase item ID • current/next flag set to 'current' (indicated that the 'current' keys are requested) *This operation is network-internal, and is not further specified.*

6	**OK (SEAKs and PEAKs)** The SEAKs or PEAKs corresponding to the purchase item as specified in the request are returned by the Service Distribution Management to the RI. There may be some information regarding usage rules specified, which influence the generation of the rights expression by the RI. Recommended content of the message: • purchase item ID • time span of validity of contained SEAKs or PEAK • list of key records, containing - CID - SEAK or PEAK • usage rule info *This operation is network-internal, and is not further specified.*
7	**generate RO** The RI generates the RO containing the desired keys. In the case that the Service Subscription Management has specified some particular usage rights, these are included in the rights expression of the RO, under consideration of the usage rules that may have been specified by the Service Distribution Management. *This operation is network-internal, and is not further specified.*
8	**OK (ROAP trigger)** In the case of success, the RI sends an OK message back to the Service Subscription Management, containing also the ROAP trigger that the device may use to request the prepared RO. The device certificate may also be returned as a result of this operation. Recommended content of the message: • trigger for the rights object acquisition (the trigger includes the rights issuer URL from which the RO can be acquired by the device) *This operation is network-internal, and is not further specified.*
9	**Subscription RO Renewal Response OK (trigger for RO acquisition)** As part of the positive response to the purchase request, the trigger for the RO acquisition is sent to the device. The ROAP trigger includes the URL of the RI that the device can use to retrieve the prepared RO. Alternatively, if the device is a mixed-mode device the RI MAY decide to deliver a BCRO over the broadcast channel. In this case the ROAP trigger will not be delivered to the device and the flow continues with the delivery of a BCRO (see clause B.5.2). *This message is further specified in this clause.*
10	**RO Request** Using the information contained in the ROAP trigger, the device initiates the 2-pass ROAP. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
11	**Authenticate** Before doing any further processing, the RI authenticates the device and/or user. *This operation is network-internal, and is not further specified.*
12	**RO Response** As a result of a successful RO acquisition, the RO is available in the device. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
13	**Charge** How charging is done (e.g. credit card, direct bank debit, generation of charging record to be included in a periodical invoice) is completely up to the implementation and to the contract between the user and Service Subscription Management. *This operation is network-internal, and is not further specified.*

B.5.1.1.8 Asynchronous Charging and Cancellation of Open-Ended Subscriptions

In the case of open-ended subscriptions, the user is charged asynchronously from time to time, irrespective of any RO renewals. Typically, such asynchronous charging could happen on a monthly basis.

Open-ended subscriptions are valid until they are cancelled by the user. Depending on the contract, they may also have to be cancelled (and renewed by issuing a new purchase request) when the price per subscription period changes.

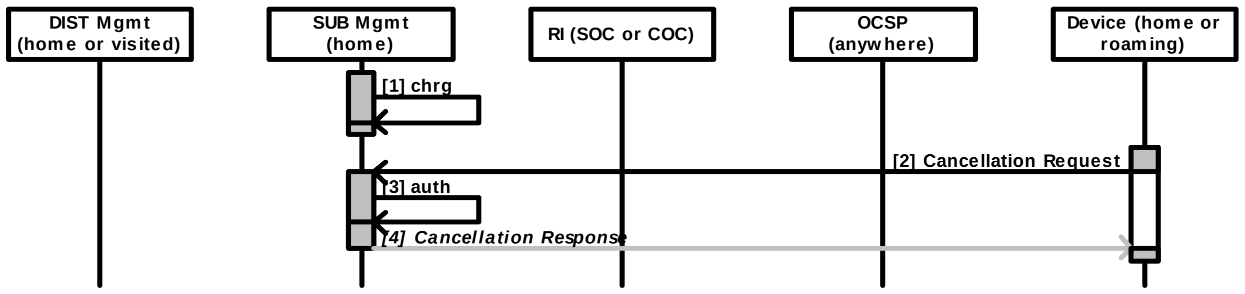


Figure B.50: Interactions for Asynchronous Charging and Cancellation of Open-Ended Subscriptions

Table B.117

1	**charge** How charging is done (e.g. credit card, direct bank debit, generation of charging record to be included in a periodical invoice) is completely up to the implementation and to the contract between the user and Service Subscription Management. *This operation is network-internal, and is not further specified.*
2	**Cancellation Request** The device sends the cancellation request to the same Service Subscription Management from where it originally purchased the subscription. If the cancellation is received only after the device has already retrieved the ROs pertaining to the next subscription period, cancellation may become effective at the end of the current or at the end of the next subscription period. The cancellation may or may not have an immediate effect. If the user has already paid for the current subscription period, the subscription RO renewal is expected to succeed until the current subscription period is over, and fail thereafter. *This message is further specified in this clause.*
3	**authenticate** Before doing any further processing, the Service Subscription Management authenticates the device and/or user, using some or all of the credentials that are part of the purchase request, or even some credentials that can be supplied by the cellular network, such as MSISDN. The device certificate may be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
4	**Cancellation Response** The cancellation response includes a text which tells the user until when the cancelled service can still be received. This means that the device should continue renewing ROs until the renewal fails. *This message is further specified in this clause.*

B.5.1.1.9 Purchase of Tokens for Consumption-based Charging

The following sequence shows the interactions needed for the purchase of tokens that can be used for the metering-based pre- and post-paid models.

The purchase of tokens is fully separated from the consumption metering, and independent of the existence of particular purchase items. However, it may depend on the contract a user has with the SUB Mgmt whether or not that particular user can purchase tokens, and whether to use pre- or post-paid mode.

Concerning the delivery of tokens, the interactions for pre- and post-paid mode are identical, but the charging step bears different semantics:

- in the pre-paid case, the tokens that are delivered as a result of the purchase transaction are charged immediately;
- in the post-paid case, the charging is done after the token usage has been reported by the device;
- the mechanisms for acquiring the token amount and reporting the token usage are specified in [49].

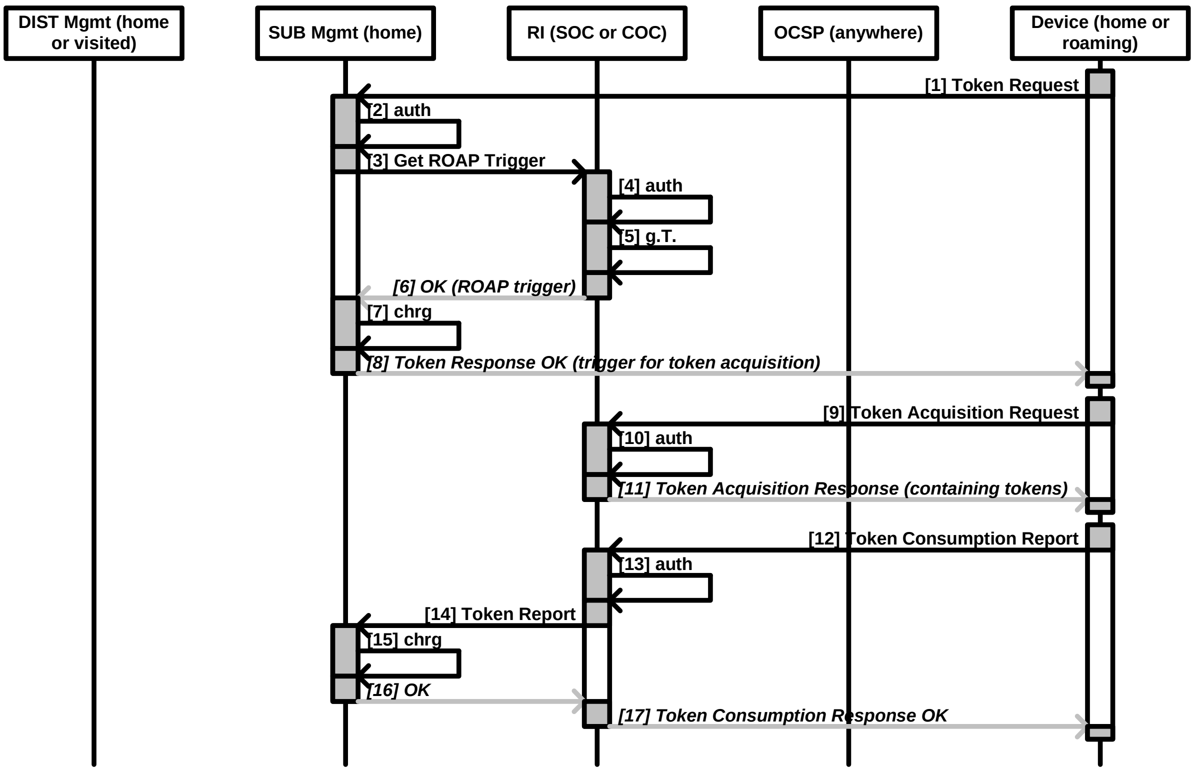

DIST Mgmt (home or visited)
SUB Mgmt (home)
RI (SOC or COC)
OCSP (anywhere)
Device (home or roaming)
[1] Token Request
[2] auth
[3] Get ROAP Trigger
[4] auth
[5] g.T.
[6] OK (ROAP trigger)
[7] chrg
[8] Token Response OK (trigger for token acquisition)
[9] Token Acquisition Request
[10] auth
[11] Token Acquisition Response (containing tokens)
[12] Token Consumption Report
[13] auth
[14] Token Report
[15] chrg
[16] OK
[17] Token Consumption Response OK

Figure B.51: Interactions for Acquisition and Charging of Tokens

Table B.118

1	**Token Request** Based on an internal logic, which is not in the scope of the present document, the terminal or user decides to purchase additional tokens from the SUB Mgmt, and issues a token request. The attributes of the token request are: • the type of charging (pre-paid or post-paid) • the requested number of tokens. In pre-paid mode these will be charged. In post-paid mode they indicate the requested credit limit (the actual number of tokens that will be sent to the device is up to the Subscription Management). *This message is further specified in this clause.*
2	**authenticate** Before doing any further processing, the SUB Mgmt authenticates the terminal and/or user, using some or all of the credentials that are part of the purchase request, or even some credentials that can be supplied by the cellular network, such as MSISDN. The device certificate may be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
3	**Get ROAP Trigger** The SUB Mgmt requests a ROAP trigger from the RI. The request may also include (part of) the rights expression that will be included in the RO. The SUB Mgmt will use the same RI as for the original purchase request. Recommended content of the message: • RI device ID • number of tokens • in the case of post-paid, an indication that metering is requested for this token delivery and the URL to which the RI should send the metering report. *This operation is network-internal, and is not further specified*
4	**Authenticate** Before doing any further processing, the RI authenticates the terminal and/or user. *This operation is network-internal, and is not further specified.*
5	**generate Token** The RI generates the desired amount of tokens. *This operation is network-internal, and is not further specified.*
6	**OK (ROAP trigger)** In the case of success, the RI sends an OK message back to the SUB Mgmt, containing also the ROAP trigger that the terminal may use to request the prepared tokens. Recommended content of the message: • trigger for the rights object acquisition (the trigger includes the rights issuer URL from which the token RO can be acquired by the terminal) The device certificate may also be returned as a result of this operation. *This operation is network-internal, and is not further specified.*
7	**Charge (pre-paid only)** How charging is done (e.g. credit card, direct bank debit, generation of charging record to be included in a periodical invoice) is completely up to the implementation and to the contract between the user and SUB Mgmt. This step is executed only if pre-paid charging is selected. *This operation is network-internal, and is not further specified.*
8	**Token Response OK** As part of the positive response to the token request, the trigger for token acquisition is sent to the terminal. The ROAP trigger includes the URL of the RI that the terminal can use to retrieve the prepared token. *This message is further specified in this clause.*
9	**Token Acquisition Request** Using the information contained in the ROAP trigger, the terminal initiates the 2-pass ROAP. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
10	**Authenticate** Before doing any further processing, the RI authenticates the terminal and/or user. *This operation is network-internal, and is not further specified.*

11	**Token Acquisition Response** As a result of a successful token acquisition, the token is available in the terminal. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
12	**Token Consumption Report (post-paid only)** If post-paid charging was selected, the device reports the token consumption to the RI. The rest of this sequence applies only to the post-paid charging case. *This is a standard OMA DRM 2.0 operation, and is not further specified.*
13	**Authenticate** Before doing any further processing, the RI authenticates the terminal and/or user. *This operation is network-internal, and is not further specified.*
14	**Token Report** After having validated it, the RI forwards the token report to the SUB Mgmt. *This operation is network-internal, and is not further specified.*
15	**Charge** The SUB Mgmt charges the user based on the amount of tokens consumed. *This operation is network-internal, and is not further specified.*
16	**OK** *This operation is network-internal, and is not further specified.*
17	**Token Consumption Report OK** *This is a standard OMA DRM 2.0 operation, and is not further specified.*

B.5.1.2 Protocol

The Service Provider SHALL support HTTP POST, which MAY be used for purchase requests over the Interactivity channel.

The Service Provider MAY support HTTPS POST, which MAY be used for purchase requests over the Interactivity channel.

The Service Provider SHALL use either HTTP POST or HTTPS POST for purchase requests over the Interactivity channel.

The Device SHALL support both HTTP POST and HTTPS POST for purchase requests over the Interactivity channel.

The device needs to know the URL for HTTP or HTTPS sessions. It is expected that this is supported by information contained in the Service Guide.

B.5.1.2.1 HTTP headers

Request messages are sent as HTTP content of type 'application/xml'. Responses are always sent as part of the '200 OK' response to the original request. The content type is 'application/xml' if the response has only one payload, or 'multipart/mixed' in the case of multiple payloads (e.g. if the response includes one or more ROAP triggers). In the latter case, the content type of a single payload of the multipart content will be 'application/xml' for the messages defined here, and 'application/vnd.oma.drm.roap-trigger+xml' for the ROAP triggers.

B.5.1.2.2 Signatures

All request messages SHALL be signed with the Private Device Key, using the RSASSA-PSS algorithm [21] (see also clause B.16). The input to the signature operation SHALL be the XML payload of the request without the 'signature' element, canonicalized according to the Exclusive XML Canonicalization algorithm [33].

B.5.1.3 XML Schemas for Request and Response Messages

B.5.1.3.1 Basic Types

This clause contains the XML schema fragment definitions for the elementary data types used in the subsequent message definitions.

B.5.1.3.1.1 User Data Type

The user data includes the user's identity known to the Customer Operator Centre that will be used for billing and, optionally, the user's preferred language (specified as a two-letter code as defined in [18]). Below is the XML schema fragment for the UserDataType:

```
  <xs:complexType name="userDataType">
   <xs:sequence>
     <xs:element name="userID" type="xs:string"/>
     <xs:element name="lang" type="xs:language" minOccurs="0"/>
   </xs:sequence>
 </xs:complexType>
```

B.5.1.3.1.2 Device Data Type

The device data includes the device identification and the device capabilities. Device capabilities are optional, and if they are omitted the Subscription Management MAY assume the capabilities announced in a previous request. However, if the device has never announced its capabilities to the Subscription Management, the capabilities SHALL be included in the request.

The detailed device data are:

- The deviceID known to the Subscriber Management. For mixed-mode devices, the content of this field SHALL be the UDN of the device.
- The device ID known to the Right Issuer, specified in [49].
- The broadcast bearers supported by the device (currently defined identifiers: 'dvbh', 'mbms', 'bcmcs').
- The service protection protocols supported by the device (currently defined identifiers: 'ipsec', 'srtp', 'ismacryp').
- The device type ('interactive' for Interactive-Only devices, 'broadcast' for Broadcast-Only devices, 'mixed' for Mixed-Mode devices).

```
  <xs:complexType name="deviceDataType">
   <xs:sequence>
     <xs:element name="deviceID" type="xs:token"/>
     <xs:element name="riDeviceID" type="xs:token"/>
     <xs:element name="bcastBearer" type="xs:token" minOccurs="0" maxOccurs="unbounded"/>
     <xs:element name="serviceProtection" type="xs:token" minOccurs="0"
                                         maxOccurs="unbounded"/>
     <xs:element name="deviceType" type="xs:token" minOccurs="0"/>
   </xs:sequence>
 </xs:complexType>
```

B.5.1.3.1.3 Domain Type

If the Service Guide indicates that the purchase is available for a domain, the user is allowed to select such an option by including the domain ID in the purchase request. For mixed-mode devices, the domain data also specifies whether the domain is an OMA DRM 2.0 domain ('omadrm2') or a Broadcast Domain ('broadcast').

```
  <xs:complexType name="domainType">
   <xs:sequence>
     <xs:element name="domainID" type="xs:token"/>
     <xs:element name="domainType" type="xs:token" minOccurs="0"/>
   </xs:sequence>
 </xs:complexType>
```

B.5.1.3.1.4 ServiceOperatorCentreType

The Service Operation Centre data contains the following information. Please note that there are two XML elements describing the Service Operation Centre. One XML element, the ServiceOperationCentreType is used in ESG signalling while this XML element of the type ServiceOperatorCentreType is used in request and response messages.

SocID: Globally coordinated ID of the Service Operation Centre.

SocInfoURL: The URL through which the SubMgmt can retrieve purchase information from the SOC.

SocKeyURL: The URL from which an RI can receive service and programme keys.

RiID: Globally coordinated ID of the Rights Issuer.

RiURL: The Rights Issuer URL, from which the SubMgmt can retrieve the ROAP triggers that will be delivered to the device.

RiProxyURL: The RI Proxy URL, from which the broadcast of BCROs in a foreign network can be requested (see clause C.2.2). If this parameter is received via the service guide, it is mandatory to include it in the SOC data.

If SOC information is included in one of the requests, the socID is always MANDATORY. RiProxyUrl is always OPTIONAL. The other elements may or may not be needed, depending on the particular message, as illustrated in table below:

Table B.119: Definition of Mandatory SOC Attributes in Request/Response Messages

Message \ Element	SocID	socInfoURL	socKeyURL	riID	RiURL	riProxyURL
Pricing Request	M	M				
Purchase Request	M	M	M	M	M	
Renewal Request	M		M	M	M	
Cancel Request	M					
Token Request	M			M	M	
<Any> Response	M					
Legend: 'M' means that the element is MANDATORY for the particular request.						

Below is the XML schema fragment for the ServiceOperatorCentreType:

```
  <xs:complexType name="serviceOperatorCentreType">
   <xs:sequence>
     <xs:element name="socID" type="xs:token"/>
     <xs:element name="socInfoURL" type="xs:anyURI" minOccurs="0"/>
     <xs:element name="socKeyURL" type="xs:anyURI" minOccurs="0"/>
     <xs:element name="riID" type="xs:token" minOccurs="0"/>
     <xs:element name="riURL" type="xs:anyURI" minOccurs="0"/>
     <xs:element name="riProxyURL" type="xs:anyURI" minOccurs="0"/>
   </xs:sequence>
 </xs:complexType>
```

B.5.1.3.1.5 PriceType

The price information contains the price as a decimal value and an optional currency qualifier (specified as defined in [20]), as illustrated by the following XML schema fragment.

```
 <xs:complexType name="priceType">
   <xs:simpleContent>
     <xs:extension base="xs:decimal">
       <xs:attribute name="currency" type="xs:token" use="optional"/>
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
```

B.5.1.3.1.6 Purchase Item Type

The purchase item data contains the information needed to identify the user's purchase:

ItemID: the purchase_item_id received from the service guide.

PurchaseOption: an identifier of the particular type of purchase, e.g. open-ended subscription or one-time subscription, a received from the service guide.

Price: the price of the item known to the user (optional)

Version: the version number of this purchase item, as indicated in the service guide.

Below is the XML schema fragment for the PurchaseItemType.

```
  <xs:complexType name="purchaseItemType">
   <xs:sequence>
     <xs:element name="itemID" type="xs:token"/>
     <xs:element name="version" type="xs:nonNegativeInteger" minOccurs="0"/>
     <xs:element name="purchaseOption" type="xs:token" minOccurs="0"/>
     <xs:element name="price" type="c18:priceType" minOccurs="0"/>
   </xs:sequence>
 </xs:complexType>
```

B.5.1.3.1.7 Request Type

Each request message extends the base request type. The request type has one 'version' attribute that indicates the interface version used for this communication.

```
 <xs:complexType name="requestType">
   <xs:attribute name="interfaceVersion" type="xs:nonNegativeInteger" use="required"/>
 </xs:complexType>
```

B.5.1.3.1.8 Response Type

Each response message extends the base response type. The response type has a 'status' attribute, which is either success or error, a 'reason code' attribute that indicates the cause of error among the ones listed in table B.3 and an optional free-form error text.

```
  <xs:complexType name="responseType">
   <xs:sequence>
     <xs:element name="message" type="xs:string" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="status" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:token">
         <xs:enumeration value="success"/>
         <xs:enumeration value="error"/>
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="reasonCode" type="xs:nonNegativeInteger" use="optional"/>
 </xs:complexType>
```

B.5.1.3.2 Error Codes

Table B.120 lists all the possible reasonCode values for error case, and their applicability to each transaction.

Table B.120: Occurrence of Error Codes in Response Messages

Code	Error Situation	Pricing	Purchase	Renewal	Cancel	Token
0	**Authentication Failed** This code indicates that the Service Subscription Management was unable to authenticate the user or the device, which may be due to the fact that the user or the device is not registered with the Service Subscription Management. In this case, the user may contact the Service Subscription Management, and establish a contract, or get the credentials in place that are used for authentication.	X	X	X	X	X
1	**Purchase Item Unknown** This code indicates that the requested purchase item is unknown. This can happen e.g. if the device has a cached service guide with old information. In this case, the user may re-acquire the service guide.	X	X			
2	**Device Not Authorized** This code indicates that the device is not authorized to get ROs from the RI, e.g. because the device certificate was revoked. In this case, the user may contact the Service Subscription Management operator.		X	X		X
3	**Device Not Registered** This code indicates that the device is not registered with the RI that is used for the transaction. When this code is sent, the response message includes a registration trigger that allows the device to register. In this case, the device MAY automatically perform the registration, and, if the registration is successful, re-initiate the original transaction.		X	X		X
4	**Server Error** This code indicates that there was a server error, such as a problem connecting to a remote back-end system. In such a case, the transaction may succeed if it is re-initiated later.	X	X	X	X	X
5	**Device Error** This code indicates that there has been a device malfunction, such as a mal-formed XML request. In such a case, the transaction may or may not (e.g. if there is an interoperability problem) succeed if it is re-initiated later.	X	X	X	X	X
6	**Charging Error** This code indicates that the charging step failed (e.g. agreed credit limit reached, account blocked) and therefore the requested RO cannot be provided. The user may in such a case contact the Service Subscription Management operator.		X			X
7	N**o Subscription** This code indicates that there has never been a subscription for this purchase item, or that the subscription for this purchase item has terminated. The user may in such a case issue a purchase request for a new subscription.			X	X	

Code	Error Situation	Pricing	Purchase	Renewal	Cancel	Token
8	Operation not Permitted This code indicates that the operation that the device attempted to perform is not permitted under the contract between Service Subscription Management and user. The user may in this case contact Service Subscription Management operator and change the contract.	X	X	X	X	X
9	**Unsupported Version** This code indicates that the version number specified in the request message is not supported by the network. In this case, the user may contact the Service Subscription Management operator.	X	X	X	X	X
10	**Signature Error** This code indicates that the validation of the message signature failed. In this case, the user may contact the Service Subscription Management operator.	X	X	X	X	X
11	**Domain Error** This code indicates that the device has requested a purchase for a domain it does not belong to. When this code is sent, the response message includes a 'join domain' trigger that allows the device to join the domain if it has the necessary permissions. In this case, the device may automatically perform the join domain procedure, and, if the operation is successful, re-initiate the original transaction.		X	X		
12	**Purchase Item Version Mismatch** This code indicates that the purchase item requested by the device has an older version number compared to the one stored in the COC. In this case, the device should update its ESG and retry the purchase.		X			
Legend: 'X' means that only the particular request may result in the corresponding error code.						

B.5.1.3.3 Pricing Request

The pricing request includes the user, device and SOC data described above, and a list of the identifiers of the purchase items for which the user is requesting the price details.

B.5.1.3.3.1 XML Schema

```
<xs:element name="pricingRequest" type="pricingRequestType"/>
<xs:complexType name="pricingRequestType">
  <xs:complexContent>
    <xs:extension base="requestType">
      <xs:sequence>
        <xs:element name="user" type="userDataType"/>
        <xs:element name="device" type="deviceDataType"/>
        <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
        <xs:element name="purchaseItemList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="purchaseItem" type="purchaseItemType" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="signature" type="xs:base64Binary"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

B.5.1.3.3.2 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pricingRequest interfaceVersion="1">
   <user>
      <userID>24403123456</userID>
      <lang>en</lang>
   </user>
   <device>
      <deviceID>0044005817853</deviceID>
      <riDeviceID>1234567890</riDeviceID>
      <bcastBearer>dvbh</bcastBearer>
      <serviceProtection>ipsec</serviceProtection>
      <deviceType>interactive</deviceType>
   </device>
   <serviceOperationCentre>
      <socID>12345</socID>
      <socInfoURL>http://www.soc.com/info</socInfoURL>
   </serviceOperationCentre>
   <purchaseItemList>
      <purchaseItem>
         <itemID>5432</itemID>
      </purchaseItem>
   </purchaseItemList>
   <signature>f7jwx3rvEPO0vKtMup4NbeVu9kn=</signature>
</pricingRequest>
```

B.5.1.3.4 Pricing Response

The pricing response contains a global status code ('success' or 'error') and an optional global failure code in case the transaction failed completely. The response includes a list of purchase items and for each of them:

- in the case of success, the purchase options and their price;
- otherwise the item-specific error code.

B.5.1.3.4.1 XML Schema

```
<xs:element name="pricingResponse" type="pricingResponseType"/>
<xs:complexType name="pricingResponseType">
  <xs:complexContent>
    <xs:extension base="responseType">
      <xs:sequence minOccurs="0">
        <xs:element name="datacastOperator" type="serviceOperatorCentreType"/>
        <xs:element name="purchaseItemList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="purchaseItem" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:choice>
                    <xs:element name="purchaseOptionList">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="purchaseOption" maxOccurs="unbounded">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="optionID" type="xs:token"/>
                                <xs:element name="desc" type="xs:string"/>
                                <xs:element name="price" type="priceType"/>
                              </xs:sequence>
                              <xs:attribute name="type" use="required">
                                <xs:simpleType>
                                  <xs:restriction base="xs:token">
                                    <xs:enumeration value="one-time"/>
                                    <xs:enumeration value="continuous"/>
                                  </xs:restriction>
                                </xs:simpleType>
                              </xs:attribute>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                      </xs:element>
                      <xs:element name="reasonCode" type="xs:negativeInteger"/>
                    </xs:choice>
                    <xs:attribute name="itemID" type="xs:token"/>
                    <xs:attribute name="version" type="xs:nonNegativeInteger"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

B.5.1.3.4.2 Example: Successful Pricing Response

The successful pricing response contains for each individual purchase item the same purchase information that is available through the service guide, though limited to the Service Subscription Management to which the pricing request has been addressed, and containing only the availability and pricing-related information.

```
<?xml version="1.0" encoding="UTF-8"?>
<pricingResponse status="success">
   <serviceOperationCentre>
      <socID>12345</socID>
   </serviceOperationCentre>
   <purchaseItemList>
      <purchaseItem itemID="5432" version="1234345">
         <purchaseOptionList>
            <purchaseOption type="one-time">
               <optionID>A</optionID>
               <desc>one-month subscription</desc>
               <price currency="EUR">5.00</price>
            </purchaseOption>
            <purchaseOption type="continuous">
               <optionID>B</optionID>
               <desc>monthly subscription, renewable until cancellation</desc>
               <price currency="EUR">5.00</price>
            </purchaseOption>
         <purchaseOptionList>
      </purchaseItem>
   </purchaseItemList>
</pricingResponse>
```

B.5.1.3.5 Purchase Request

The purchase request includes the user, device and SOC data described above, and a list of the identifiers of items the user wants to purchase, including a purchase option and the price known to the user. The purchase can be requested for a domain, in which case optional domain data is specified.

B.5.1.3.5.1 Schema

```
  <xs:element name="purchaseRequest" type="purchaseRequestType"/>
  <xs:complexType name="purchaseRequestType">
    <xs:complexContent>
      <xs:extension base="requestType">
        <xs:sequence>
          <xs:element name="user" type="userDataType"/>
          <xs:element name="device" type="deviceDataType"/>
          <xs:element name="domain" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="domainID"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
              <xs:element name="purchaseItem" type="purchaseItemType" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="signature" type="xs:base64Binary"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

B.5.1.3.5.2 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseRequest interfaceVersion="1">
   <user>
      <userID>24403123456</userID>
      <lang>en</lang>
   </user>
   <device>
      <deviceID>0044005817853</deviceID>
      <riDeviceID>1234567890</riDeviceID>
      <bcastBearer>dvbh</bcastBearer>
      <serviceProtection>ipsec</serviceProtection>
      <deviceType>interactive</deviceType>
   </device>
   <domain>
      <domainID>dhf5434jddAdfD</domainID>
   </domain>
   <serviceOperationCentre>
      <socID>12345</socID>
      <socInfoURL>http://www.soc.com/info</socInfoURL>
      <socKeyURL>http://www.soc.com/keys</socKeyURL>
      <riID>4567</riID>
      <riURL>http://www.soc.com/ri</riURL>
   </serviceOperationCentre>
   <purchaseItemList>
      <purchaseItem>
         <itemID>5432</itemID>
         <version>123435</version>
         <purchaseOption>B</purchaseOption>
         <price currency="EUR">5.00</price>
      </purchaseItem>
   </purchaseItemList>
   <signature>h8twx3rvEPO0vKtMup4NbeVu0flO</signature>
</purchaseRequest>
```

B.5.1.3.6 Purchase Response

The purchase response contains a global status code ('success' or 'error') and an optional global failure code in the case that the transaction failed completely. The response includes a list of item-specific status codes and in the case of a subscription to a service that requires RO renewal, the last day and time of validity of the RO per subscription is given.

B.5.1.3.6.1 XML Schema

```
<xs:element name="purchaseResponse" type="purchaseResponseType"/>
<xs:complexType name="purchaseResponseType">
  <xs:complexContent>
    <xs:extension base="responseType">
      <xs:sequence>
        <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
        <xs:element name="purchaseItemList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="purchaseItem" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="reasonCode" type="xs:nonNegativeInteger" minOccurs="0"/>
                    <xs:element name="roValidityEndTime" type="xs:dateTime" minOccurs="0"/>
                  </xs:sequence>
                  <xs:attribute name="itemID" type="xs:token" use="required"/>
                </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

B.5.1.3.6.2 Example: Successful Purchase Response with RO Acquisition Trigger

The successful purchase response is a multi-part message, including a RO acquisition trigger (not shown) for each of the requested purchase items.

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseResponse status="success"/>
```

B.5.1.3.6.3 Example: Unsuccessful Purchase Response with Registration Trigger

If the device is not registered, the unsuccessful purchase response is a multi-part message, containing a registration trigger (not shown).

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseResponse status="error" reasonCode="3"/>
```

B.5.1.3.6.4 Example: Unsuccessful Purchase Response with Purchase-Item-specific Error

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseResponse status="error">
 <serviceOperationCentre>
   <socID>12345</docID>
 </serviceOperationCentre>
 <purchaseItemList>
   <purchaseItem itemID="5432">
   <reasonCode>1</reasonCode>
   </purchaseItem>
 </purchaseItemList>
</purchaseResponse>
```

B.5.1.3.7 Subscription RO Renewal Request

The subscription renewal request includes the user, device and SOC data described above, and a list of the identifiers of purchase items corresponding to subscriptions the user wants to renew.

B.5.1.3.7.1 XML Schema

```
 <xs:element name="renewalRequest" type="renewalRequestType"/>
 <xs:complexType name="renewalRequestType">
   <xs:complexContent>
     <xs:extension base="requestType">
       <xs:sequence>
         <xs:element name="user" type="userDataType"/>
         <xs:element name="device" type="deviceDataType"/>
         <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
         <xs:element name="purchaseItemList">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="purchaseItem" type="purchaseItemType" maxOccurs="unbounded"/>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="signature" type="xs:base64Binary"/>
```

```
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

B.5.1.3.7.2 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<renewalRequest interfaceVersion="1">
 <user>
   <userID>24403123456</userID>
   <lang>en</lang>
 </user>
 <device>
   <deviceID>0044005817853</deviceID>
   <riDeviceID>1234567890</riDeviceID>
   <bcastBearer>dvbh</bcastBearer>
   <serviceProtection>ipsec</serviceProtection>
   <deviceType>interactive</deviceType>
 </device>
 <serviceOperationCentre>
   <socID>12345</socID>
   <socKeyURL>http://www.soc.com/keys</socKeyURL>
    <riID>4567</riID>
    <riURL>http://www.soc.com/ri</riURL>
 </serviceOperationCentre>
 <purchaseItemList>
   <purchaseItem>
     <itemID>5432</itemID>
   </purchaseItem>
 </purchaseItemList>
 <signature>q2ewx3rvEPO0vKtMup4NbeVu1wd9</signature>
</renewalRequest>
```

B.5.1.3.8 Subscription RO Renewal Response

The subscription renewal response contains a global status code ('success' or 'error') and an optional global failure code in case the transaction failed completely. The response includes a list of item-specific status codes the updated last day and time of validity of the RO after the renewal.

B.5.1.3.8.1 Schema

```
<xs:element name="renewalResponse" type="renewalResponseType"/>
<xs:complexType name="renewalResponseType">
  <xs:complexContent>
    <xs:extension base="responseType">
      <xs:sequence>
        <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
        <xs:element name="purchaseItemList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="purchaseItem" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="reasonCode" type="xs:nonNegativeInteger" minOccurs="0"/>
                    <xs:element name="roValidityEndTime" type="xs:dateTime" minOccurs="0"/>
                  </xs:sequence>
                  <xs:attribute name="itemID" type="xs:token"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

B.5.1.3.8.2 Example: Successful Renewal Response with RO Acquisition Trigger

The successful renewal response is a multi-part message, including a RO acquisition trigger (not shown) for each of the purchase items for which the RO is renewed.

```
<?xml version="1.0" encoding="UTF-8"?>
<renewalResponse status="success">
 <serviceOperatorCentre>
   <socID>12345</socID>
 </serviceOperatorCentre>
 <purchaseItemList>
   <purchaseItem itemID="5432">
     <roValidityEndTime>2005-02-19T00:59:59Z</roValidityEndTime>
   </purchaseItem>
 </purchaseItemList>
</renewalResponse>
```

B.5.1.3.8.3 Example: Unsuccessful Renewal Response with Registration Trigger

If the device is not registered, the unsuccessful renewal response is a multi-part message, containing a registration trigger (not shown).

```
<?xml version="1.0" encoding="UTF-8"?>
<renewalResponse status="error" reasonCode="3"/>
```

B.5.1.3.8.4 Example: Unsuccessful Renewal Response with Purchase-Item-specific Error

If individual items cannot be found, the unsuccessful renewal response contains purchase-item-specific reason codes.

```
<?xml version="1.0" encoding="UTF-8"?>
<renewalResponse status="error">
 <serviceOperationCentre>
   <socID>12345</socID>
 </serviceOperationCentre>
 <purchaseItemList>
   <purchaseItem itemID="5432">
       <reasonCode>7</reasonCode>
   </purchaseItem>
 </purchaseItemList>
</renewalResponse>
```

B.5.1.3.9 Subscription Cancellation Request

The subscription cancellation request includes the user, device and SOC data described above, and a list of the identifiers of purchase items corresponding to subscriptions the user wants to cancel.

B.5.1.3.9.1 XML Schema

```
 <xs:element name="cancellationRequest" type="cancellationRequestType"/>
 <xs:complexType name="cancellationRequestType">
   <xs:complexContent>
     <xs:extension base="requestType">
       <xs:sequence>
         <xs:element name="user" type="userDataType"/>
         <xs:element name="device" type="deviceDataType"/>
         <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
         <xs:element name="purchaseItemList">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="purchaseItem" type="purchaseItemType" maxOccurs="unbounded"/>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="signature" type="xs:base64Binary"/>
```

```
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

B.5.1.3.9.2 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<cancellationRequest interfaceVersion="1">
 <user>
   <userID>24403123456</userID>
   <lang>en</lang>
 </user>
 <device>
   <deviceID>0044005817853</deviceID>
   <riDeviceID>1234567890</riDeviceID>
   <bcastBearer>dvbh</bcastBearer>
   <serviceProtection>srtp</serviceProtection>
   <deviceType>interactive</deviceType>
 </device>
 <serviceOperationCentre>
   <socID>12345</socID>
 </serviceOperationCentre>
 <purchaseItemList>
   <purchaseItem>
     <itemID>5432</itemID>
   </purchaseItem>
 </purchaseItemList>
 <signature>w2jwx3rvEPO0vKtMup4NbeVu9en1</signature>
</cancellationRequest>
```

B.5.1.3.10 Subscription Cancellation Response

The subscription cancellation response contains a global status code ('success' or 'error') and an optional global failure code in case the transaction failed completely. In case of success, the response includes a text per purchase item that tells the user when the cancellation takes effect, i.e. from when on the ROs cannot be renewed any more.

The device is expected to continue renewing ROs until the renewal fails with a 'no subscription' error message.

B.5.1.3.10.1 Schema

```
  <xs:element name="cancellationResponse" type="cancellationResponseType"/>
  <xs:complexType name="cancellationResponseType">
    <xs:complexContent>
      <xs:extension base="responseType">
        <xs:sequence>
          <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="purchaseItem" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="reasonCode" type="xs:nonNegativeInteger" minOccurs="0"/>
                      <xs:element name="cancellationInfoMessage" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="itemID" type="xs:token"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

B.5.1.3.10.2 Example: Successful Cancellation Response

```
<?xml version="1.0" encoding="UTF-8"?>
<cancellationResponse status="success">
 <serviceOperationCentre>
   <socID>1234




5</socID>
 </serviceOperationCentre>
 <purchaseItemList>
   <purchaseItem itemID="5432">
      <cancellationInfoMessage>Service valid until 2005-03-27</cancellationInfoMessage>
   </purchaseItem>
 </purchaseItemList>
</cancellationResponse>
```

B.5.1.3.10.3 Example: Unsuccessful Cancellation Response With Purchase-Item-specific Error

```
<?xml version="1.0" encoding="UTF-8"?>
<cancellationResponse status="error">
 <serviceOperationCentre>
   <socID>12345</socID>
 </serviceOperationCentre>
 <purchaseItemList>
   <purchaseItem itemID="5432">
       <reasonCode>7</reasonCode>
   </purchaseItem>
 </purchaseItemList>
</cancellationResponse>
```

B.5.1.3.11 Token Request

In addition to the user, device and SOC data described above, the token request includes:

ChargingType: the type of charging (pre-paid or post-paid) the user wishes to use. The SubMgmt will verify that the requested charging type is available for this user.

RequestedTokens: the amount of new tokens requested by the device. In case of pre-paid, the amount of tokens requested is subtracted from the user's credit. In case of post-paid, it is verified that the amount of tokens requested does not exceed the user's credit limit.

B.5.1.3.11.1 XML Schema

```
 <xs:element name="tokenRequest" type="tokenRequestType"/>
 <xs:complexType name="tokenRequestType">
   <xs:complexContent>
     <xs:extension base="requestType">
       <xs:sequence>
         <xs:element name="user" type="userDataType"/>
         <xs:element name="device" type="deviceDataType"/>
         <xs:element name="serviceOperatorCentre" type="serviceOperatorCentreType"/>
         <xs:element name="paymentType">
           <xs:simpleType>
             <xs:restriction base="xs:token">
               <xs:enumeration value="prePaid"/>
               <xs:enumeration value="postPaid"/>
             </xs:restriction>
           </xs:simpleType>
         </xs:element>
         <xs:element name="requestedTokens" type="xs:negativeInteger"/>
         <xs:element name="signature" type="xs:base64Binary"/>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>
```

B.5.1.3.11.2 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<tokenRequest interfaceVersion="1">
<user>
   <userID>24403123456</userID>
   <lang>en</lang>
 </user>
 <device>
   <UDN>0044005817853</UDN>
   <riDeviceID>1234567890</riDeviceID>
   <bcastBearer>DVBH</bcastBearer>
   <serviceProtectionProtocol>IPsec</serviceProtectionProtocol>
   <broadcastMode>no</broadcastMode>
 </device>
 <serviceOperationCentre>
   <socID>12345</socID>
 </serviceOperatorCentre>
 <paymentType>postPaid</paymentType>
 <reportedTokens>10</reportedTokens>
 <requestedTokens>20</requestedTokens>
 <signature>d2jwx3rvEPO0vKtMup4NbeVu9ke7</signature>
</tokenRequest>
```

B.5.1.3.12 Token Response

The token response reports the success or failure of the operation, and includes a ROAP trigger for token acquisition as additional HTTP payload (not shown here).

B.5.1.3.12.1 Schema

```
 <xs:element name="tokenResponse" type="tokenResponseType"/>
 <xs:complexType name="tokenResponseType">
   <xs:complexContent>
     <xs:extension base="responseType"/>
   </xs:complexContent>
 </xs:complexType>
```

B.5.1.3.12.2 Example: Successful Token Response

```
<?xml version="1.0" encoding="UTF-8"?>
<tokenResponse  status="success"/>
```

B.5.1.3.12.3 Example: Unsuccessful Token Response

```
<?xml version="1.0" encoding="UTF-8"?>
<tokenResponse  status="error" failureCode="1"/>
```

B.5.1.3.13 XML schema definition for request and response related XML elements

The schema definition below defines the namespace http://www.18crypt.com/2005/XMLSchema and all XML elements used for requests and response.

The schema SHALL take precedence over XML elements declared previously in this chapter.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:c18="http://www.18crypt.com/2005/XMLSchema"
targetNamespace="http://www.18crypt.com/2005/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:complexType name="userDataType">
    <xs:sequence>
      <xs:element name="userID" type="xs:string"/>
      <xs:element name="lang" type="xs:language" minOccurs="0"/>
    </xs:sequence>
    </xs:complexType>
    <xs:complexType name="deviceDataType">
      <xs:sequence>
        <xs:element name="deviceID" type="xs:token"/>
        <xs:element name="riDeviceID" type="xs:token"/>
        <xs:element name="bcastBearer" type="xs:token" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="serviceProtection" type="xs:token" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="deviceType" type="xs:token" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="domainType">
      <xs:sequence>
        <xs:element name="domainID" type="xs:token"/>
        <xs:element name="domainType" type="xs:token" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="serviceOperatorCentreType">
      <xs:sequence>
        <xs:element name="socID" type="xs:token"/>
        <xs:element name="socInfoURL" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="socKeyURL" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="riID" type="xs:token" minOccurs="0"/>
        <xs:element name="riURL" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="riProxyURL" type="xs:anyURI" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="priceType">
      <xs:simpleContent>
        <xs:extension base="xs:decimal">
          <xs:attribute name="currency" type="xs:token" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="purchaseItemType">
      <xs:sequence>
        <xs:element name="itemID" type="xs:token"/>
        <xs:element name="version" type="xs:nonNegativeInteger" minOccurs="0"/>
        <xs:element name="purchaseOption" type="xs:token" minOccurs="0"/>
        <xs:element name="price" type="c18:priceType" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="requestType">
      <xs:attribute name="interfaceVersion" type="xs:nonNegativeInteger" use="required"/>
    </xs:complexType>
    <xs:complexType name="responseType">
      <xs:sequence>
        <xs:element name="message" type="xs:string" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="status" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="success"/>
            <xs:enumeration value="error"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="reasonCode" type="xs:nonNegativeInteger" use="optional"/>
    </xs:complexType>
    <xs:element name="pricingRequest" type="c18:pricingRequestType"/>
    <xs:complexType name="pricingRequestType">
      <xs:complexContent>
        <xs:extension base="c18:requestType">
          <xs:sequence>
            <xs:element name="user" type="c18:userDataType"/>
            <xs:element name="device" type="c18:deviceDataType"/>
            <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
            <xs:element name="purchaseItemList">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="purchaseItem" type="c18:purchaseItemType" maxOccurs="unbounded"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="signature" type="xs:base64Binary"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  <xs:element name="pricingResponse" type="c18:pricingResponseType"/>
  <xs:complexType name="pricingResponseType">
    <xs:complexContent>
      <xs:extension base="c18:responseType">
        <xs:sequence minOccurs="0">
          <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="purchaseItem" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:choice>
                      <xs:element name="purchaseOptionList">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="purchaseOption" maxOccurs="unbounded">
                              <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="optionID" type="xs:token"/>
                                  <xs:element name="desc" type="xs:string"/>
                                  <xs:element name="price" type="c18:priceType"/>
                                </xs:sequence>
                                <xs:attribute name="type" use="required">
                                  <xs:simpleType>
                                    <xs:restriction base="xs:token">
                                      <xs:enumeration value="one-time"/>
                                      <xs:enumeration value="continuous"/>
                                    </xs:restriction>
                                  </xs:simpleType>
                                </xs:attribute>
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="reasonCode" type="xs:negativeInteger"/>
                    </xs:choice>
                    <xs:attribute name="itemID" type="xs:token"/>
                    <xs:attribute name="version" type="xs:nonNegativeInteger"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="purchaseRequest" type="c18:purchaseRequestType"/>
  <xs:complexType name="purchaseRequestType">
    <xs:complexContent>
      <xs:extension base="c18:requestType">
        <xs:sequence>
          <xs:element name="user" type="c18:userDataType"/>
          <xs:element name="device" type="c18:deviceDataType"/>
          <xs:element name="domain" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="domainID"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="purchaseItem" type="c18:purchaseItemType" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="signature" type="xs:base64Binary"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="purchaseResponse" type="c18:purchaseResponseType"/>
  <xs:complexType name="purchaseResponseType">
    <xs:complexContent>
```

```
      <xs:extension base="c18:responseType">
        <xs:sequence>
          <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="purchaseItem" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="reasonCode" type="xs:nonNegativeInteger" minOccurs="0"/>
                      <xs:element name="roValidityEndTime" type="xs:dateTime" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="itemID" type="xs:token" use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="renewalRequest" type="c18:renewalRequestType"/>
  <xs:complexType name="renewalRequestType">
    <xs:complexContent>
      <xs:extension base="c18:requestType">
        <xs:sequence>
          <xs:element name="user" type="c18:userDataType"/>
          <xs:element name="device" type="c18:deviceDataType"/>
          <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="purchaseItem" type="c18:purchaseItemType" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="signature" type="xs:base64Binary"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="renewalResponse" type="c18:renewalResponseType"/>
  <xs:complexType name="renewalResponseType">
    <xs:complexContent>
      <xs:extension base="c18:responseType">
        <xs:sequence>
          <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="purchaseItem" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="reasonCode" type="xs:nonNegativeInteger" minOccurs="0"/>
                      <xs:element name="roValidityEndTime" type="xs:dateTime" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="itemID" type="xs:token"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="cancellationRequest" type="c18:cancellationRequestType"/>
  <xs:complexType name="cancellationRequestType">
    <xs:complexContent>
      <xs:extension base="c18:requestType">
        <xs:sequence>
          <xs:element name="user" type="c18:userDataType"/>
          <xs:element name="device" type="c18:deviceDataType"/>
          <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
          <xs:element name="purchaseItemList">
            <xs:complexType>
              <xs:sequence>
              <xs:element name="purchaseItem" type="c18:purchaseItemType" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="signature" type="xs:base64Binary"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="cancellationResponse" type="c18:cancellationResponseType"/>
<xs:complexType name="cancellationResponseType">
  <xs:complexContent>
    <xs:extension base="c18:responseType">
      <xs:sequence>
        <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
        <xs:element name="purchaseItemList">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="purchaseItem" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="reasonCode" type="xs:nonNegativeInteger" minOccurs="0"/>
                    <xs:element name="cancellationInfoMessage" type="xs:string" minOccurs="0"/>
                  </xs:sequence>
                  <xs:attribute name="itemID" type="xs:token"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="tokenRequest" type="c18:tokenRequestType"/>
<xs:complexType name="tokenRequestType">
  <xs:complexContent>
    <xs:extension base="c18:requestType">
      <xs:sequence>
        <xs:element name="user" type="c18:userDataType"/>
        <xs:element name="device" type="c18:deviceDataType"/>
        <xs:element name="serviceOperatorCentre" type="c18:serviceOperatorCentreType"/>
        <xs:element name="paymentType">
          <xs:simpleType>
            <xs:restriction base="xs:token">
              <xs:enumeration value="prePaid"/>
              <xs:enumeration value="postPaid"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="requestedTokens" type="xs:nonNegativeInteger"/>
        <xs:element name="signature" type="xs:base64Binary"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="tokenResponse" type="c18:tokenResponseType"/>
<xs:complexType name="tokenResponseType">
  <xs:complexContent>
    <xs:extension base="c18:responseType"/>
  </xs:complexContent>
</xs:complexType>
</xs:schema>
```

B.5.2 Purchase for Mixed-Mode Devices

The sequences described in clause B.5.1.1 apply also to mixed-mode devices. However, for mixed-mode devices the RI MAY decide to deliver a BCRO over the broadcast channel as a result of a successful purchase. In this case the device will not receive a ROAP trigger, but instead it SHALL prepare to receive BCROs as described in clause B.3.3.4. The flows in clause B.5.1.1 are thus modified as follows:

Unsuccessful purchase (figure B.47, step 16): after the ROAP registration, the RI MAY send registration data via the PDR protocol to the device, in the case the device has not registered before.

Successful purchase (figure B.48, step 12): the device receives the 'success' response but no RO acquisition trigger. The device SHALL prepare to receive a BCRO.

RO renewal (figure B.49, step 9): the device receives the 'success' response but no RO acquisition trigger. The device SHALL prepare to receive a BCRO.

Token Request (figure B.51, step 8): the device receives the 'success' response but no RO acquisition trigger. The device SHALL prepare to receive a token BCRO.

A mixed-mode device MAY join a broadcast domain, and domain ROs for that domain MAY be delivered via the interactivity channel. In this case, the ICRO will be addressed to the longform domain ID (identical to OMA domain ID) and the device SHALL obtain the shortform domain ID from the association between longform and shortform domain IDs it has established when it received the registration data (see clause B.3.4.4).

B.5.3 Out-of-Band Purchase

B.5.3.1 Means of purchase

Out-of-band purchase is needed in order to make the purchasing functionality available to broadcast devices. Interactive devices MAY also use out-of-band purchase, for the broadcast mode of operation.

Figure B.42 shows the dataflow that governs the broadcast mode of operation. Device registration as well as the transactions related to purchasing require that information is conveyed from the device or its user to the network.

There are numerous ways how out-of-band communication can be implemented (e.g. web shop, call centre, automated SMS service, etc.). A viable operation would be where the user of the device calls the COC and tells that he wants to see the football match of tonight. Such options are not mandated.

The following clause describes the normative behaviour of the device in case it is equipped with an appropriate contact() either via the ESG or via the update_contact_number_msg() message.

B.5.3.2 Out-of-Band purchase from service guide data

Generally, the request/response pairs for purchase over the interactivity channel need to be performed out-of-band. If the out-of-band communication requires that the user speaks or types some information, it is desirable that instead of long numeric identifiers, shorter (properly scoped) numbers, codes, or names can be used. These 'human-friendly' numbers and names are expected to be part of the service guide, or, if they are used to identify the device, be put in place during device registration. In case the short version of an identifier is not available in the service guide, the long version will be used instead. The service guide will also include a textual representation of the Customer Operation Centre contact information and the out-of-band channels available, to be displayed to the user. These numbers, codes and names are established in clause B.5.4.

In order to request a purchase, the user is required to provide the following data to the Customer Operation Centre, via the out-of-band channel indicated by the service guide.

Table B.121: Data to be provided to the Customer Operation Centre

	Short concise version	Human friendly version
contact() 'number'	local phone number or international phone number or SMS number or URL	local phone number or international phone number or SMS number or URL
Device ID	ARC code (refer to clause B.15.2) including purchase code	ARC code (refer to clause B.15.2) including purchase code
Service ID	Service ID code (i.e. #serviceID)	Service name (i.e. serviceName)
Service Operator Centre ID	Soc ID (i.e. socID)	Soc Name (i.e. socName)
Purchase Item ID	purchase item code (i.e. #pItemName)	purchase item name (i.e. PItemName)
NOTE 1: The data in table B.121 is provided by the service guide information, please refer to clause B.5.4 for details. NOTE 2: A device which has not been registered before will show the UDN instead of the ARC.		

During the out-of-band purchase, the device MAY display a dialogue with instructions. Notifying the device data can be done in various ways, for example by showing the user of the device a dialogue on the screen of the device, displaying the data necessary for purchase and a contact() 'number' that needs to be contacted. A phone number may be used for vocal notification of the data to the RI. Another example is to display instructions to send an SMS message via a mobile phone to the RI. Yet another example is to display a URL, which may be used by the user to contact a web shop via a(n) offline browser.

An example of a displayed OOB purchase message for a registered device follows, where the following information is reported back to the RI.

NOTE: It is the sequence of the defined values that is specified. The use of dashes as the delimiter is shown with an example placement to be consistent with the examples used elsewhere in this specification. The text portion of this screen is shown as an example only; there is no implied requirement to duplicate the exact wording or formatting shown. The numeric fields MUST be included as defined above (please note: the ARC code will only be displayed after the first registration, when that data MAY available for display).

To purchase the selected service,
please contact customer service at:
XXXX-XXX-XXXXXXX

action request code:
XXXX XXXX XXXX

Service: <#serviceID>
Provider: <Soc ID>
Purchase ID: <#pItemID>

An example dialogue concise instructions for vocal notification of ARC and purchase data to call-centre.

To purchase the selected service,
please contact customer service at:
XXXX-XXX-XXXXXXX

action request code:
XXXX XXXX XXXX

Service: <serviceName>
Provider: <SocName>
Purchase ID: <pItemName>

An example dialogue showing human friendly instructions for vocal notification of ARC and purchase data to call-centre.

Figure B.52: Samples of out-of-band purchase information displays for a registered device

Key:

- The contact can be any number as specified in contact() (refer to clause B.3.4.3.7.6.2.1) or delivered per service guide.
- The action request code SHALL incorporate the purchase action request code (refer to clause B.3.4.3.3).

The following dialogue is an example of what an unregistered device could display.

To purchase the selected service,
please contact customer service at:
XXXX-XXX-XXXXXXX

unique device number:
XXXX XXXX XXXX XXXX XXXX

Service: <serviceName>
Provider: <SocName>
Purchase ID: <pItemName>

An example dialogue showing human friendly instructions for vocal notification of UDN and purchase data to call-centre.

NOTE: An unregistered device performing a purchase request will need to register first. Please refer to clause B.2.3.3 for details.

Figure B.53: Sample of out-of-band purchase information displays for an unregistered device

The result of a successful purchase is usually a BCRO. For its acquisition, the standard 2-pass ROAP cannot be used. Instead an RI service is defined (c.f. clause B.4), that comprises mechanisms to broadcast BCROs spontaneously (as a result of the purchase transaction) or in a scheduled manner (e.g. for re-keying the SEAKs).

B.5.4 Service Guide Information

This clause introduces the terms used in the ESG that relate to purchasing.

Details of how the relevant data is carried in the DVB-IPDC ESG can be found in clause B.22.

B.5.4.1 Service Operation Centre

(including Service Distribution Management):

socID: is the globally co-ordinated ID of the Service Operation Centre (aka. 'socID'); the DVB platform ID MAY (and is recommended to) be used for this purpose.

socName: is the name of the Service Operation Centre, reasonably globally unique (e.g. this can be achieved by concatenating of the ISO country code and an acronym that identifies the operator within the country). This name may be used for identifying the SOC operator within out-of-band purchase transactions. The name SHALL be encoded using UTF-8 [30].

socKeyURL: URL through which an RI can retrieve keys from Service Distribution Management.

socInfoURL: URL through which Service Subscription Management can retrieve purchase info from Service Distribution Management.

socRiProxyURL: URL of the RI Proxy associated with Service Operation Centre (used by a Service Subscription Management in a different network to insert BCROs, as described in clause C.2.2). This parameter is present only if the RI Proxy functionality is supported by the SOC.

B.5.4.2 Customer Operation Centre

(including Service Subscription Management):

cocID: is the globally co-ordinated ID of the Customer Operation Centre.

cocLocalFlag: indicates, if 'true', that a COC is local to the SOC, and therefore advertises the availability and purchase information completely in the service guide. This knowledge helps avoiding unnecessary requests by the device to obtain information whether or not a particular purchase item is available from a certain COC (not all items will in general be purchasable through all COCs).

cocName: is the name of the Customer Operation Centre, reasonably globally unique (e.g. this can be achieved by concatenating of the ISO country code and an acronym that identifies the operator within the country). This name may be used for identifying the COC operator within out-of-band purchase transactions. The name SHALL be encoded using UTF-8 [30].

cocRekeyingSafetyWindow: is the time interval DT, as specified in clause B.5.1.1.7, during which it cannot be guaranteed that RO renewal will succeed timely for uninterrupted access to a particular service.

cocPurchaseURL: specifies the URL for a pricing or purchase or subscription RO renewal request can be sent by the device over HTTP POST or HTTPS POST. If present, all the purchase transactions that are specified in the present document SHALL be supported.

cocPortalURL: specifies the URL on which Customer Operation Centre may offer out-of-band self-provisioning via HTTP(S). If present, the portal SHALL support all the out-of-band purchase transactions that are specified in the present document.

cocContactInfo: is a text string that indicates to a user how to contact a COC to initiate an out-of-band purchase transaction (e.g. toll-free phone number, international phone number, letter address, e-mail address, SMS number, etc.).

cocRiID: ID of the RI associated with Customer Operation Centre (needed to allow broadcast devices to identity the RI service that may be operated by their Home COC).

cocRiURL (mandatory, 1): URL of the RI associated with Service Operation Centre (used by Service Subscription Management for requesting ROAP triggers or requesting registration data or BCROs to be broadcast)

The service guide SHALL include an entry for every COC with which the SOC has an agreement ('locality agreement', 'roaming agreement', c.f. clause C.2). For 'local COCs', the service guide SHALL include the complete availability and purchase information, and SHOULD include also the pricing information (purchase options with their respective prices).

B.5.4.3 Service

serviceID: is the ID of the service, unique within the scope of a Service Distribution Management (socID).

serviceTypeEnum: is the type of the service. The allowed values are:

'media': the service is a regular media service.

'sg': the service is a service guide, and its streams carry service guide objects.

'ri': the service is an RI Service, and its streams carry RI objects (registration data, BCROs, RI triggers, RI messages) for the broadcast mode of operation.

serviceName: is the name of the service, unique within the scope of a Service Distribution Management (socID). The service name may be used for identifying the service within out-of-band purchase transactions. The name SHALL be encoded using UTF-8 [30].

serviceDescription: is the description of the service.

serviceBaseCID: is used as part of the CID of all service and program keys related to the service.

B.5.4.4 ScheduleItem

sItemID: is the ID of the schedule item (programme), unique within the scope of a Service Distribution Management (socID).

sItemName: is the user-friendly name that is used to identify the schedule item, unique within the scope of a Service Distribution Management (socID). The schedule item name may be used for identifying the schedule item within out-of-band purchase transactions. The name SHALL be encoded using UTF-8 [30].

sItemDescription: is the description of the schedule item.

sItemStartTime: is the start time of the schedule item, specified as yyyy-mm-ddThh:mm:ss.

sItemEndTime: is the end time of the schedule item, specified as yyyy-mm-ddThh:mm:ss

B.5.4.5 ContentItem

cItemID: is the ID of the content item, unique within the scope of a Service Distribution Management (socID).

cItemTypeEnum: is the type of the content item. The allowed values are:

'media' - the content item is a regular media item.

'riSet' - the content item is defining a set of devices using the broadcast mode of operation.

cItemName: is the user-friendly name that is used to identify the content item, unique within the scope of a Service Distribution Management (socID). The schedule item name may be used for identifying the content within out-of-band purchase transactions. The name SHALL be encoded using UTF-8 [30].

cItemDescription: is the description of the content item. If, and only if, the type of the content item is 'riSet', then the description is a structured mono-language attribute with the following sub-structure:

cItemGroupRange: is a range of subscriber groups, specified as a tuple (low, high).

cItemDeviceRange: is a range of individual device IDs, specified as a tuple (low, high).

cItemDomainRange: is a range of domain IDs, specified as a tuple (low, high).

B.5.4.6 Purchase Item

pItemID: is the ID of the purchase item (aka. 'purchase_item_id'), unique within the scope of a Service Distribution Management (socID).

pItemVersion: is the version number of the purchase item, used to verify that the purchase item data received via the ESG is synchronized with the data in the Subscription Management.

pItemName: is the user-friendly name that is used to identify the service, unique within the scope of a Service Distribution Management (socID). The purchase item name may be used for identifying the purchase item within out-of-band purchase transactions. The name SHALL be encoded using UTF-8 [30].

pItemDescription: is the description of the service, used for the purposes of the user.

pItemServiceID: is the serviceID of a service that is part of the purchase item (which is in this case a subscription item to a service bundle, and is not intended to have any schedule or content items associated).

pItemScheduleItemID: is the ID of a schedule item (sItemID) that is part of the purchase item (which is in this case a pay-per-view purchase for a program, and is not intended to have any services or content items associated).

pItemContentItemID: is the ID of a content item (cItemID) that is part of the purchase item (which SHOULD in this case not have any services or schedule items associated).

B.5.4.7 Purchase Data

cocID: is the ID of the Customer Operation Centre.

pItemID: is the ID of the purchase item.

purchaseOption: is a structure including all the information (purchase option code [unique within purchase data record], description [multi-language], price [with currency], type [continuous or one-time], availability for domain subscription and for which domain type [OMA DRM 2.0, broadcast or both]) that is needed for the user to decide upon a purchase, and for the device to execute a purchase (i.e. to make a purchase request); this should essentially be the same information as the information returned to the device in a successful pricing response.

For a 'local COC' the availability information SHALL be complete, i.e. for all purchase items that are available through the COC there SHALL be a purchase data fragment in the service guide, which SHALL include also all purchase options and their respective pricing.

B.6 Security Considerations

B.6.1 Handling Weak Keys

(The responsible components in) the head-end architecture SHALL NOT use weak keys that will be used for messages in the IP Datacast network. At the time of this writing there are no specified weak keys for use in AES. This does not mean to imply that weak keys do not exist. If, at some point, a set of weak keys for AES are identified, the use of these weak keys SHALL be rejected in the head-end architecture followed by a request for replacement key.

B.6.2 Handling OCSP Grace Period

If a device without a return channel inspects a certificate, because the user wants to consume certain content for which he has acquired the RO, and the device finds out that the OCSP response of the certificate chain has expired, then the device is still allowed to use it for a short period of time during which the user has time to set the process in motion through which the device will receive a new OCSP response. This means that the user can enjoy the content he was entitled to consume straight away, at the expense of a slightly increased security risk of being able to use possibly compromised certificates for a somewhat longer time.

A device in broadcast-only mode SHALL implement the grace period mechanism:

1) The device checks periodically a particular or all RI context for expiration.

2) If a RI context is expired, the device displays an OCSP response expiry reminder for the associated RI context. The reminder notifies the user that the user needs to get a new OCSP response (of course in terms that a user can understand like 'Call this number with this message please').

3) Until this OCSP response expiry reminder is invoked the device will be rendered inoperable, but only in relation with the associated RI (context) as described below:

    a) Accessing an ESG for purchase is still allowed.

    b) The device SHALL be rendered inoperable for any purchase protocol or playback of future content. The device MAY use stored BCROs to play old content for which the device obtained ROs, but SHALL NOT use these BCROs for new content received after the re-registration request until the device received a fresh OCSP response or is re-registered with the RI.

4) A device SHALL be allowed to use an expired OCSP response for a pre-defined grace period. The grace period SHALL NOT be more than the OCSP response's lifetime (the difference between the nextUpdate and thisUpdate fields in the OCSP response), and MUST NOT exceed 48 hours.. During the grace period, the device can use the expired OCSP response:

    a) The grace period is for a one-time use only.

    b) The terminal SHALL support secure DRM time.

c) Rules in ROs SHALL have precedence over the OCSP response grace period usage.

5) If the secure timer (i.e. grace period) expires and a fresh OCSP response has not been received, the device will be rendered inoperable, but only in relation with the associated RI (context) as described below:

a) Accessing an ESG for purchase is still allowed.

b) The device SHALL be rendered inoperable for any purchase protocol or playback of future content. The device MAY use stored BCROs to play old content for which the device obtained ROs, but SHALL NOT use these BCROs for new content received after the re-registration request until the device received a fresh certificate chain or is re-registered with the RI.

A device in broadcast mode MAY implement a mechanism to automatically schedule the certificate chain updates.

1) An update (powerup/powerdown) timeslot is programmed in which the RI will transmit the certificate chain. The timeslot may be obtained from the ESG. The device SHOULD parse the received ESG data to find a time at which it can receive a certificate chain update. Note that it may be the case that certificate chain updates are broadcast continuously. See clause B.4.8 for more details.

2) Upon power down before update the device may display a warning message that the device needs to update its device chain. An example might look like: "Do not power off device. Device will perform update during xx:yy h".

The device will be powered up and down in timeslot xx:yy h to pick up the message to update the RI certificate chain (notably the OCSP response).

B.7 Status and Error Message Handling

This clause describes the status and error values for use in the 1-pass protocols for broadcast devices.

The Status field is a binary value. Upon receipt of a message for which Status is not "Success", the default behaviour, unless explicitly stated otherwise below, is that both the RI and the Device SHALL immediately close the connection and terminate the protocol. RI systems and Devices are required to delete any session-identifiers, nonces, keys, and/or secrets associated with a failed run of the protocol.

When possible, the Device SHOULD present an appropriate error message to the user.

NOTE: It is the sequence of the defined values that is specified. The use of dashes as the delimiter is shown with an example placement to be consistent with the examples used elsewhere in this specification. The text portion of this screen is shown as an example only; there is no implied requirement to duplicate the exact wording or formatting shown. The numeric fields SHALL be included as defined above (please note: the short UDN will only be displayed after the first registration, when that data is available for display).

```
The service cannot continue due to an error.
Please contact customer service at:
XXXX-XXX-XXXXXXX

and notify the short UDN:
XXXX XXXX
with following errorcode
XXX
```

An example dialogue showing an error

Figure B.61: sample notification display

NOTE: The error codes should be displayed as a three digit decimal number. Refer to table B.122 for an overview of possible error codes.

Table B.122: Status / error codes

Status / Error	value$_{(h)}$	comment
Success	0x00	
UnknownError	0x01	
Abort	0x02	
NotSupported	0x03	
AccessDenied	0x04	
NotFound	0x05	
MalformedRequest	0x06	
UnknownRequest	0x07	
UnsupportedVersion	0x08	
NoCertificateChain	0x09	
SignatureError	0x0A	
DeviceTimeError	0x0B	
NotRegistered	0x0C	
InvalidDomain	0x0D	
DomainFull	0x0E	
TokenConsumptionMessageError	0x0F	
NoTokenConsumptionMessage	0x10	
ForceInteractiveChannel	0x11	
ForceOobChannel	0x12	
Reserved for future use	0x13 to 0xFF	

UnknownError: indicates an internal RI system error.

Abort: indicates that the RI rejected the Device's request for unspecified reasons.

NotSupported:indicates the Device made a request for a feature currently not supported by the RI.

AccessDenied*:* indicates that the Device is not authorized to contact this RI.

NotFound: indicates that the requested object was not found.

MalformedRequest: indicates that the RI failed to parse the Device's request.

UnknownRequest: indicates that the RI did not recognize the request type.

UnsupportedVersion: indicates that the Device used a ROAP protocol version not supported by the RI.

NoCertificateChain: indicates that the RI could not verify the signature on a Device request due to not having access to the Device's certificate chain.

SignatureError: indicates that the RI could not verify the Device's signature.

DeviceTimeError: indicates that Rights Issuer request a Device to set the Device DRM Time with a new value and report the time drift to the Rights Issuer.

NotRegistered*:* indicates that the Device tried to contact an RI with which it has not completed a valid registration. The RI SHOULD include the ***RI ID*** attribute in the response message in which this error code is sent. The Device SHOULD perform the offline device registration protocol until a retry limit but SHALL have user consent to start.

InvalidDomain*:* indicates that the request was invalid due to an unrecognized Domain Identifier.

DomainFull: indicates that no more Devices are allowed to join the Domain.

TokenConsumptionMessageError: indicates that the RI did receive a token consumption message, but that it was erroneous and that the device should redo the last token consumption message.

NoTokenConsumptionMessag: indicates that the RI did not receive a token consumption message yet, but was expecting one, because the present date/time is later than the last latest_token_consumption_time sent to the device in a token delivery response message.

ForceInteractiveChannel: indicates that the RI forces a mixed mode device to exclusively use its interactive channel and not its OOB channel.

ForceOobChanne: indicates that the RI forces a mixed mode device to exclusively use its OOB channel and not its interactive channel.

B.8 Mapping of messages to DVB-H Time Sliced Bursts

This clause specifies how the various key streams and other messages SHALL be mapped to DVB-H Time Slicing bursts. For definition of such a burst, see [15].

B.8.1 Key Stream Messages

The following requirements apply to Key Stream Messages:

- Key Stream Messages SHALL be carried in the same burst as the content to which the 'current' traffic key they carry applies. Key Stream Messages MAY carry a second traffic key applying to the next crypto period, which MAY apply to content in the next burst.
- A Key Stream Message containing the DRM time field SHOULD be carried in every burst.
- Within a burst, Key Stream Messages SHOULD be broadcast before the content packets which they are used to decrypt.

B.8.2 RI Service Channel

There are no requirements for the mapping of RI Services to DVB-H bursts.

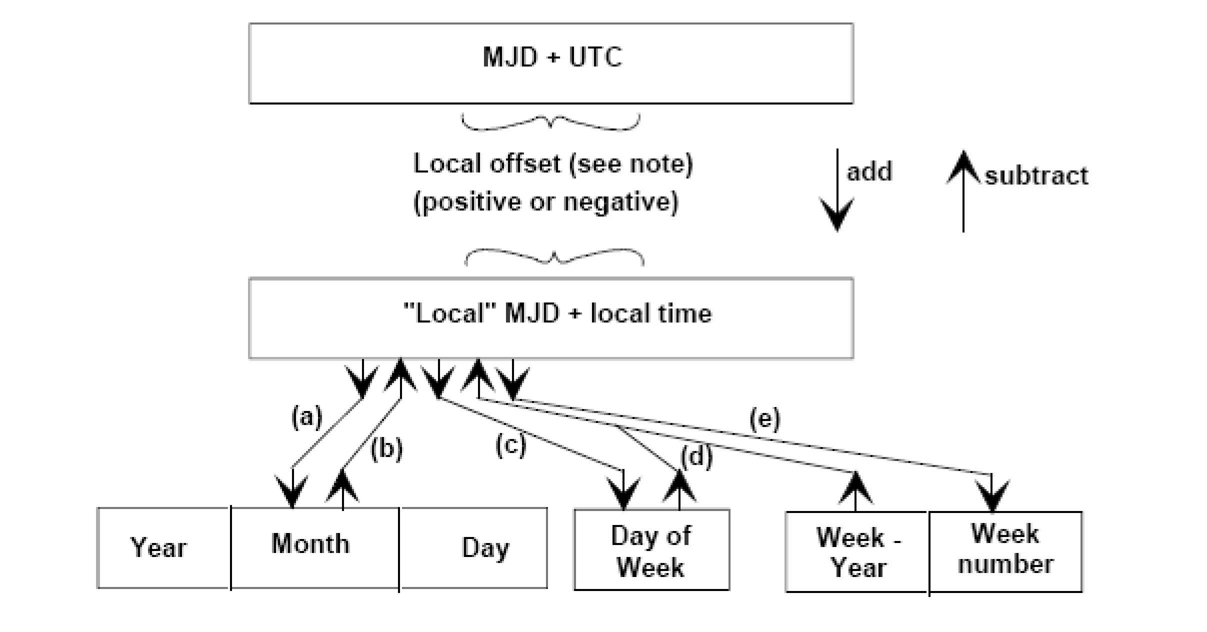
B.8.2.1 In-Band RI Stream

In-Band RI Streams are carried as a separate IP stream within a service. As such, they are broadcast within the bursts of that service.

B.9 Conversion between Time and Date Conventions

NOTE: This text originates from EN 300 468 [14] V1.6.1, but has been slightly modified. The version in EN 300 468 [14] V1.6.1 has a byte alignment problem caused by the time offset polarity field. The version in this text maintains byte alignment everywhere.

The types of conversion which may be required are summarized in figure B.62.



NOTE: Offsets are positive for Longitudes East of Greenwich and negative for Longitudes West of Greenwich.

Figure B.62: Conversion routes between Modified Julian Date (MJD) and Co-ordinated Universal Time (UTC)

The conversion between MJD + UTC and the "local" MJD + local time is simply a matter of adding or subtracting the local offset. This process may, of course, involve a "carry" or "borrow" from the UTC affecting the MJD. The other five conversion routes shown on the diagram are detailed in the formulas below:

Symbols used:

D	Day of month from 1 to 31
int	Integer part, ignoring remainder
K, L ,M', W, Y'	Intermediate variables
M	Month from January (= 1) to December (= 12)
MJD	Modified Julian Date
MN	Week number according to ISO 2015 [57]
mod 7	Remainder (0 to 6) after dividing integer by 7
UTC	Universal Time, Co-ordinated
WD	Day of week from Monday (= 1) to Sunday (= 7)
WY	"Week number" Year from 1900
x	Multiplication
Y	Year from 1900 (e.g. for 2003, Y = 103)

a) To find Y, M, D from MJD
$Y' = \text{int}\,[\,(MJD - 15\,078{,}2) / 365{,}25\,]$
$M' = \text{int}\,\{\,[\,MJD - 14\,956{,}1 - \text{int}\,(Y' \times 365{,}25)\,] / 30{,}6001\,\}$
$D = MJD - 14\,956 - \text{int}\,(Y' \times 365{,}25) - \text{int}\,(M' \times 30{,}6001\,)$
If $M' = 14$ or $M' = 15$, then $K = 1$; else $K = 0$
$Y = Y' + K$
$M = M' - 1 - K \times 12$

b) To find MJD from Y, M, D
If $M = 1$ or $M = 2$, then $L = 1$; else $L = 0$
$MJD = 14\,956 + D + \text{int}\,[\,(Y - L) \times 365{,}25] + \text{int}\,[\,(M + 1 + L \times 12) \times 30{,}6001\,]$

c) To find WD from MJD
$WD = [\,(MJD + 2) \bmod 7\,] + 1$

d) To find MJD from WY, WN, WD
$MJD = 15\,012 + WD + 7 \times \{\,WN + \text{int}\,[\,(WY \times 1\,461 / 28) + 0{,}41]\,\}$

e) To find WY, WN from MJD
   W = int [(MJD / 7) - 2 144,64]
   WY = int [(W × 28 / 1 461) - 0,0079]
   WN = W - int [(WY × 1 461 / 28) + 0,41]
   EXAMPLE: MJD = 45 218 W = 4 315
   Y = (19)82 WY = (19)82
   M = 9 (September) N = 36
   D = 6 WD = 1 (Monday)

   NOTE: These formulas are applicable between the inclusive dates 1900 March 1 to 2100 February 28.

B.9.1 Local Time Offset

This 16-bit field contains the current offset time from UTC in the range between -12 hours and +13 hours at the area which is indicated by the combination of country_code and country_region_id in advance. These 16 bits are coded as 4 digits in 4-bit BCD in the order hour tens, hour, minute tens, and minutes.

The positive or negative offset from the UTC is indicated with the 1 bit local_time_offset_polarity. If this bit is set to '0' the polarity is positive and the local time is advanced to UTC. (Usually east direction from Greenwich). If this bit is set to '1' the polarity is negative and the local time is behind UTC. Please note that the local_time_offset_polarity is represented by the first bit of the first nibble representing the hour tens field. The first nibble of the local_time_offset is therefore encoded as follows.

Table B.123: Local time offset coding

local_time_offset_polarity	Offset hour tens	First nibble
0 (i.e. '+')	0	0000
0 (i.e. '+')	1	0001
1 (i.e. '-')	0	1000
1 (i.e. '-')	1	1001

B.10 Conversion of OMA DRM 2.0 Content Identifiers to Binary Content Identifiers

The CID in OMA DRM 2.0 has the form of a URI. For a binary rights object as defined in clause B.3.3.4.2, this text base id has to be transformed into a binary id. The ID is split into a base_BCI and an extension_BCI.

Definition.

Table B.124

Field	Description/Value	Type
socID	Service Operator Centre ID as signalled in ESG	String
content_id	textual id as used in OMA DRM 2.0 and signalled in the ESG	String
content_type	'#P' for programmes and '#S' for services	String
cid-url	socID || content_type || content_id ||'@'	String

The 64 bit binary content id BCI is then given by:

base_BCI = HMAC-SHA1-64(cid-url)

The extension_BCI is a 32 bit id (see program_CID_extension and service_CID_extension in the KSM).

The BCI is given by:

$$BCI = (base\_BCI << 32) \mid extension\_BCI$$

B.11 Conversion of OMA X509PKISHash Values to Binary Values

Values codes as roap:X509SPKIHash in OMA DRM 2.0 are SHA1-160 hashes. In OMA DRM 2.0 these values are presented as base64 encodings in a XML <hash> element as shown by example below:

```
<keyIdentifier xsi:type='roap:X509SPKIHash'>
    <hash>aXENc+Um/9/NvmYKiHDLaErKofk=</hash>
</keyIdentifier>
```

If such a hash is carried in a binary object e.g. the BCRO or a KSM the 160-bit hash itself without the base64 encoding will be used.

B.12 Limits of the surplus_block()

Following examples show two possible cases: one keyset for standards addressing and one keyset with additional domain addressing.

B.12.1 Standard Keyset

for standard addressing is keyset_block filled with:

- 1 UGK, 9 SGK , 1UDK , 1 UDF , 1 RIAK, 1 UDF.

NOTE: Max subscriber group size is 512, this is supported by 9 SGK.

Table B.125: standard keyset with RSA block size 1 024

Value	Variable	Key size	Key data	Key size	Key data	Key size	Key data
1 024	RSA size						
1	UDF	40	40	40	40	40	40
1	UGK	128	128	128	128	128	128
9	SGK	128	1 152	128	1 152	128	1 152
1	UDK	128	128	128	128	128	128
1	RIAK	128	128	128	128	128	128
0	BDK	128	0	128	0	128	0
0	SBDF	48	0	48	0	48	0
0	LBDF	840	0	840	0	840	0
0	TDK	128	0	128	0	128	0
13	TLF overhead	7	181	7	181	7	181
	keyset_block		1 757		1 757		1 757
1	SK	128	128	192	192	256	256
0	PKCS overhead		0		0		0
	sessionkey_block()	1 024		1 024		1 024	
	room in sessionkey_block() to use for keyset_block		896		832		768
	(remainder of keyset_block in) surplus_block()		861		925		989

Other block sizes with keyset as depicted above produce following results.

Table B.126: standard keyset with other RSA block sizes

block size	SK size	keyset_block	room in sessionkey_block()	surplus block
2 048	128	1 757	1 920	no
2 048	192	1 757	1 856	no
2 048	256	1 757	1 792	no
4 096	128	1 757	3 968	no
4 096	192	1 757	3 904	no
4 096	256	1 757	3 840	no

B.12.2 Extended Keyset

An extended keyset includes keys for standard addressing plus domain addressing. The keyset_block is filled with:

- 1 UGK, 9 SGK , 1UDK , 1 UDF , 1 RIAK, 1 UDF, 1 BDK, 1 SBDF, 1 LBDF (maximum size), 1 TDK.

NOTE: Max subscriber group size is 512, this is supported by 9 SGK.

Table B.127: extended keyset with RSA block size 1 024

Value	Variable	Key size	Key data	Key size	Key data	Key size	Key data
1 024	RSA size						
1	UDF	40	40	40	40	40	40
1	UGK	128	128	128	128	128	128
9	SGK	128	1 152	128	1 152	128	1 152
1	UDK	128	128	128	128	128	128
1	RIAK	128	128	128	128	128	128
1	BDK	128	128	128	128	128	128
1	SBDF	48	48	48	48	48	48
1	LBDF	840	840	840	840	840	840
1	TDK	128	128	128	128	128	128
17	TLF overhead	7	219	7	219	7	219
	keyset_block		2 939		2 939		2 939
1	SK	128	128	192	192	256	256
0	PKCS overhead		0		0		0
	sessionkey_block()	1 024		1 024		1 024	
	room in sessionkey_block() to use for keyset_block		896		832		768
	(remainder of keyset_block in) surplus_block()		2 043		2 107		2 171

Table B.128: extended keyset with other RSA block sizes

block size	SK size	keyset_block	room in sessionkey_block()	surplus block
2 048	128	1 924	1 920	1 019
2 048	192	1 924	1 856	1 083
2 048	256	1 924	1 792	1 147
4 096	128	1 924	3 968	no
4 096	192	1 924	3 904	no
4 096	256	1 924	3 840	No
NOTE: not yet included is the PKCS overhead in the sessionkey_block(), so surplus_block() will be a <u>little</u> larger.				

B.13 Function F_{ZMB}

B.13.1 Definitions

The function for the ZMB system are defined as follows:

a) **node numbering calculation**. The keys in the zero message broadcast encryption key tree are numbered in a breadth-first fashion. The root key has number 0 (NK_0). For any parent key NKi, the left child key is NK2i+1 and the right child key is NK2i+2.

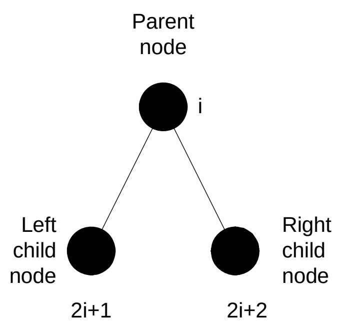


Figure B.63: Node numbering

b) **node / leaf key derivation function**. Given a parent key NKi, the derived child keys NK2i+1 and NK2i+2 are calculated by encrypting a known constant using AES as shown in figure B.64.

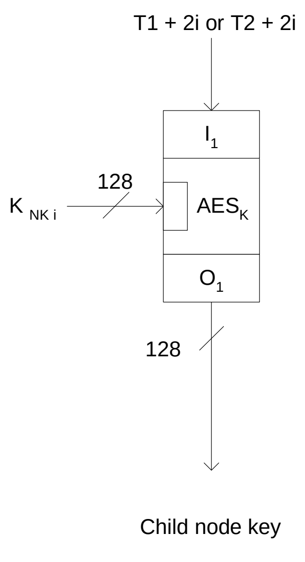


Figure B.64: AES for key derivation

Explaining figure B.64:

Step 1: Initialize the system with following values:

- Let T1 = 0x01010101010101010101010101010101
- Let T2 = 0x02020202020202020202020202020202
- Load $K_{NK\,i}$ (msblf), where $K_{NK\,i}$ = *SGK* (in case of device) or $K_{NK\,i}$ = *Root Key* NK_0 (in case of RI).

Step 2: Calculate the Left Child Node and Right Child Node from an Input Key SGK. Or, Given a parent key NKi (i.e. Root Key or SGK), the derived child keys NK2i+1 (i.e. Left Child Node) and NK2i+2 (i.e. Right Child Node) are specified by following functions:

$$NK2i+1 = AES\{NKi\}(2i+T1)$$

$$NK2i+2 = AES\{NKi\}(2i+T2)$$

NOTE: AES compliant to [17].

c) **computation of all keys.** The rights issuer computes all keys in the tree by recursively applying the relations given by (b) starting from the root key NK0 that is known only to the rights issuer.

d) **leaf number calculation**. In a subscriber group for n devices , these n devices are associated with the leaf keys NKn-1 up to and including NK2n-2. A device with position p (numbered from 0 to n-1) in the group is associated with key NKp+n-1.

e) **determining the (SGK) keyset of a device.** Define the functions sibling(i) and parent(i) as follows:

```
sibling(i) = { i-1 if i is even,
               i+1 if i is odd }.

 parent(i) = { i/2 - 1 if i is even
              (i-1)/2 if i is odd }
```

Then the following algorithm defines the key set to be given to a device with position p in the group:

```
i := p+n-1
KeySet = {}
while i > 0 {
     i := sibling(i)
     KeySet = KeySet union { NKi }
     i := parent(i)
 } end while
```

f) **determining the exclusion keys.** Given an access mask, it is possible to determine the key numbers of the keys used as exclusion keys. Each device given its own keyset can determine these exclusion keys, except if the key associated with its own position is used as an exclusion key. To effectively disallow devices to access a certain asset, the rights issuer derives an IEK by concatenating the device revocation keys and using this concatenation as key for computing a MAC over the Binary Content Identifier BCI as retrieved from the rights object:

$$IEK = HMAC - SHA1\_128\{NK_{ex-1} \| NK_{ex-2} \| \ldots \| NK_{ex-n}\}(BCI)$$

g) **IEK calculation.** The notation HMAC_SHA1(k, s) is used to denote the computation of HMAC [23] keyed by the key "k" over the string "s" with SHA1 [44] as the hash function. HMAC_SHA1_128(k, s) is used to denote the 128 most significant bits of HMAC_SHA1(k, s) output. Note that RFC 2104 [23] specifies that if the key is longer than the output block length of SHA1 then the key is first hashed to the SHA1 block length. This is expected to frequently be the case in this algorithm. A device that has verified that its own key is not part of the exclusion key set, can calculate the IEK using the following algorithm:

```
function CalculateEncryptionKey
{
    IEK = 0
    b := 0
      Temp := ''
    while b < n {
        if bit b in access_mask equals 0 {
              Temp := Temp || CalculateNodeKey(b + n - 1)
        } end if
    } end while
      IEK := HMAC_SHA1_128(Temp, Salt)
    return IEK
}

function CalculateNodeKey(i)
{
    if NKi is in KeySet {
        return NKi from KeySet
    }
```

```
        else {
            parentkey = CalculateNodeKey( parent(i) )
            return AES{ parentkey } ( i )
        }
    }
```

B.13.2 Examples

B.13.2.1 Sample Tree and Keyset

Following picture shows a sample tree with keys for a device at position 7 (i.e. Device D0), as defined in subsection "e" of clause B.13.1.

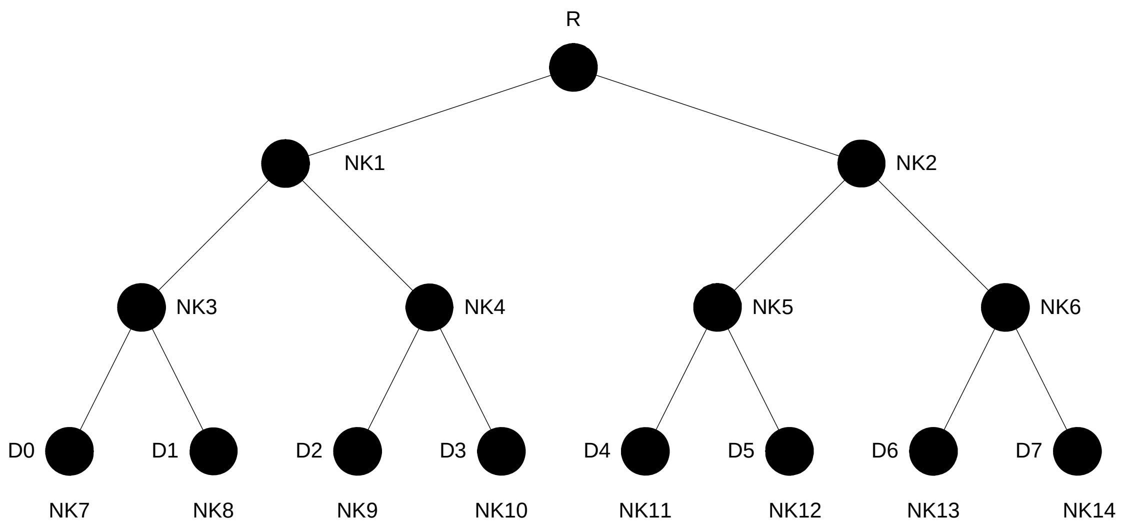


Figure B.65: sample tree with correct node and device numbering

E.g.1: Keyset for D0 = {NK2, NK4, NK8}

E.g.2: Keyset for D4 = {NK1, NK6, NK12}

E.g.3: To effectively disallow devices D1, D5 and D7 to access a certain asset, the rights issuer derives an IEK by concatenating the device revocation keys (NK_8, NK_{12} and NK_{14}) , and using this concatenation as key for computing a HMAC-SHA1-128 over the Binary Content Identifier BCI as retrieved from the rights object:

$$IEK = HMAC - SHA1\_128\{NK_8 \parallel NK_{12} \parallel NK_{14}\}(BCI)$$

B.14 Authentication

For a quick overview of the authentication 'hierarchy' of the system refer to clause B.2.4.1.6.

B.14.1 Authentication for IPsec

IPsec can be used with authentication. In case of authentication with IPsec the authentication data will carry the TAS. The authentication mechanism will create the TAK from the TAS.

To obtain the encrypted traffic key material from the KSM the encrypted traffic key material is decrypted with the SEK or PEK:

$$TAS = D\{SEK\}(traffic\_key\_material)$$

or

$$TAS = D\{PEK\}(traffic\_key\_material)$$

The authentication key is generated from the authentication seed:

$$TAK = f_{auth}\{TAS\}(CONSTANT\_TAK)$$

where:

```
CONSTANT_TAK  = 0x040404040404040404040404040404 (120 bit)
```

Refer to clause B.14.4 for details on f-auth.

The TAK is used in the MAC generation / verification of the IPsec data. Refer to [26] for details.

B.14.2 Authentication for KSMs

A key stream message can contain two MAC fields. The programme MAC field and the service MAC field. If only one MAC field would be used, the authentication key could only be renewed when both SEK and PEK change at the same time. Having two MAC fields and two authentication keys makes it possible to authenticate the message and check for its integrity while only having one key set. The service authentication key (SAK) and the programme authentication key (PAK) will be derived from the service authentication seed and the programme authentication seed respectively which are transmitted together with the encryption keys in the ROs (How this is carried in the BCRO and ICRO in explained in subsequent clauses). A service RO will contain service encryption and authentication keys (SEAK) and a programme RO will contain programme encryption and authentication keys (PEAK).

To obtain the PAS or SAS from the BCRO the SEAK/PEAK is decrypted with the IEK:

$$SAS = LSB_{128}(D\{IEK\}(SEAK))$$

$$PAS = LSB_{128}(D\{IEK\}(PEAK))$$

The authentication key is generated from the authentication seed:

$$SAK = f_{auth}\{SAS\}(CONSTANT\_SAK)$$

$$PAK = f_{auth}\{PAS\}(CONSTANT\_PAK)$$

where :

```
CONSTANT_SAK    = 0x020202020202020202020202020202 (120 bit)
CONSTANT_PAK    = 0x010101010101010101010101010101 (120 bit)
```

Refer to clause B.14.4 for details on f-auth.

The SAK or PAK is used in the MAC generation / verification of the KSM. The algorithm used to calculate the MAC field is HMAC-SHA1-96 according to [45] and RFC 2104 [23], using authentication keys of 160 bit in both cases.

B.14.2.1 Transport of SEAK and PEAK in OMA DRM 2.0 Rights Objects

The encryption keys and authentication keys (SEAK and PEAK), encrypted with a key transport algorithm as specified in [49] (default being AES_wrap [13]), are transported in an ICRO as separate ds:KeyInfo elements in the <asset> fragment of the Rights Object. Hence the <asset> fragment of a service ICRO contains the following (see [50] for a detailed description of the XML elements).

```
<o-ex:asset o-ex:id="asset_ID">

[…]

  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
      <xenc:CipherData>
        <xenc:CipherValue>encrypted_service_encryption_key</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <ds:KeyInfo Id="service_authentication_seed_id" >
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
      <xenc:CipherData>
        <xenc:CipherValue>encrypted_service_authentication_seed</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
</o-ex:asset>
```

encrypted_service_encryption_key =

$$E\{REK\}(SEK) = AES\_wrap\{REK\}(SEK)$$

encrypted_service_authentication_seed =

$$E\{REK\}(SAS) = AES\_wrap\{REK\}(SAS)$$

where SEK and SAS are both an AES key of 128 bits and service_authentication_seed_id is a unique identification of the authentication seed KeyInfo element within the ICRO, constructed as follows:

```
service_authentication_seed_id = service_period_CID || "_authSeed"
```

Similarly, the <asset> element of a programme ICRO contains:

```
<o-ex:asset>

[…]

  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
      <xenc:CipherData>
        <xenc:CipherValue>encrypted_programme_encryption_key</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <ds:KeyInfo Id="programme_authentication_seed_id" >
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
      <xenc:CipherData>
        <xenc:CipherValue>encrypted_programme_authentication_seed</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
</o-ex:asset>
```

encrypted_programme_encryption_key =

$$E\{REK\}(PEK) = AES\_wrap\{REK\}(PEK)$$

encrypted_programme_authentication_seed =

$$E\{REK\}(PAS) = AES\_wrap\{REK\}(PAS)$$

where PEK and PAS are both an AES key of 128 bits and programme_authentication_seed_id is a unique identification of the authentication seed KeyInfo element within the ICRO, constructed as follows:

```
programme_authentication_seed_id = programme_CID || "_authSeed"
```

B.14.2.2 Transport of SEAK and PEAK in BCROs

The encryption keys and authentication keys (SEAK and PEAK) are transported in a BCRO by concatenating the encryption key and the authentication seed and then protecting the resulting field with AES CBC.

encrypted_service_encryption_authentication_key =

$$E\{IEK\}(SEAK) = AES\_CBC\{IEK\}(SEK << 128) \parallel SAS)$$

where SEK and SAS are both an AES key of 128 bits.

and encrypted_programme_encryption_authentication_key =

$$E\{IEK\}(PEAK) = AES\_CBC\{IEK\}(PEK << 128) \parallel PAS)$$

where PEK and PAS are both an AES key of 128 bits.

B.14.3 Authentication of BCROs

BCROs contain one MAC field which is used to authenticate the message and to protect the integrity of the message.

The authentication key is generated from the RIAK:

$$BAK = f_{auth}\{RIAK\}(CONSTANT\_BCRO)$$

where:

```
CONSTANT_BCRO = 0x030303030303030303030303030303 (120 bit)
```

NOTE: To obtain the RIAK the device needs to have been equipped with a valid keyset. Refer to clause B.3.4.3 for details.

Refer to clause B.14.4 for details on f-auth.

The BAK is used in the MAC generation / verification of the BCRO. The algorithm used to calculate the MAC field is HMAC-SHA1-96 according to [45] and RFC 2104 [23], using an authentication key of 160 bit.

B.14.4 General Authentication Mechanism

The function F-auth consists of several steps:

1) Denote by PRF{key}(text) as the AES-XCBC-MAC-PRF with output block size 128 bits as defined by IPsec WG in IETF. Please note:

   - Refer to [28] for the AES-XCBC-MAC-PRF based key generation function.

   - Refer to [31] for the requirement NOT to truncate the generated key material.

2) Apply the generated input key according to ideas of IKEv2 to generate authentication key. Define a key generator function f-kg{key}(constant). Keying material will always be derived as the output of the negotiated PRF algorithm.. PRF$^+$ describes the function that outputs a pseudo-random stream of n blocks based on the inputs to a PRF as follows:

$$T1 = AES\_XCBC\_MAC\_PRF\{AS\}(CONSTANT \parallel 0x01)$$

$$T2 = AES\_XCBC\_MAC\_PRF\{AS\}(T1 \parallel CONSTANT \parallel 0x02)$$

....

$$Tn = AES\_XCBC\_MAC\_PRF\{AS\}(T1 \| CONSTANT \| n)$$

where *AS* is the appropriate authentication seed (be it TAS, PAS, SAS or RIAK) and *CONSTANT* is the appropriate constant as described in preceding clauses. The amount of blocks to derive is defined by the amount of key material needed, i.e. n is the amount of needed key bits divided by 128 and rounded up.

This means that if 160 bits were needed then $PRF^{+}()$ would be computed as:

$$T1 \| T2 = PRF^{+}\{K\}(S)$$

3) The 160 bit authentication key is taken from the generated key material as follows:1

$$AK = MSB_{160}(T1 \| T2)$$

The generated authentication key is applied as described in preceding clauses.

B.14.5 Authentication of Token Delivery Response Messages

Token delivery response messages contain one MAC field which is used to authenticate the message and to protect the integrity of the message.

The authentication key is generated from the RIAK:

$$TDRMAK = f_{auth}\{RIAK\}(CONSTANT\_TDRM)$$

where:

```
CONSTANT_TDRM = 0x050505050505050505050505050505  (120 bit)
```

NOTE: To obtain the RIAK the device needs have been equipped with a valid keyset. Refer to clause B.3.4.3 for details.

Refer to clause B.14.4 for details on f-auth.

The TDRMAK is used in the MAC generation / verification of the token delivery response message. The algorithm used to calculate the MAC field is HMAC-SHA1-96 according to [45] and RFC 2104 [23], using an authentication key of 160 bit.

B.15 Checksum Algorithms

According to empirical research by [52] the likelihood of errors is expressed as:

Table B.129

nr	error	representation	relative likelihood in %
1	single substitution	a => b	60 to 95
2	single adjacent transpositions	ab => ba	10 to 20
3	twin errors	aa => bb	0,5 to 1,5
4	jump transpositions (Longer jumps are even rarer)	acb => bca	0,5 to 1,5

5	phonetic errors (phonetic, because in some languages the two have similar pronunciation, e.g., thirty and thirteen)	a0 => 1a
		where a={2,..,9}
			0,5 to 1,5

6	adding or omitting digits		10 to 20
Key: a < > b, while c can be any decimal digit.			

The most common errors are therefore errors 1, 2 and 6. Error 6 is easily detected. Following clauses explain a method to detect other errors.

B.15.1 UDN Checksum

Definition

The checksum on the UDN is calculates by *F-UDN*

We use codes over Zp, the integers modulo p, where p=11. That is to say, codewords are strings with entries from for $\{0,1,....p-1\}$. We consider codes of length n defined by r parity equations: a string $(c1, c2..., cn)$ with elements from Zp is a codeword if and only if it satisfies the following equations:

for $$i = 1,2,...r, \sum_{j=1}^{n} aj^{(i)} cj \equiv 0(\text{mod } p)$$

We now describe a [20, 17] code, that is defined over 20 symbols from Z11 using the three following check equations as described in the matrix H3 below:

Take *n*=17, *r*=3 and *p*=11. We consider the code defined by the *r*=3 following check equations:

```
0*c1 + 1*c2 + 0*c3 +...+ 1*c18 = 0 (modulo 11)
 1*c1 + 0*c2 + 1*c3 +...+ 1*c19 = 0 (modulo 11)
10*c1 + 1*c2 + 9*c3 +...+ 1*c20 = 0 (modulo 11)
```

In other words, a string (*c*1, *c*2,..., *c*20) with elements from Z11is a codeword if and only if it has inner product zero (modulo 11) with the rows of the following matrix *H3*:

	n1	**n2**	**n3**	**n4**	**n5**	**n6**	**n7**	**n8**	**n9**	**n10**	**n11**	**n12**	**n13**	**n14**	**n15**	**n16**	**n17**	**n18**	**n19**	**n20**
	1	0	1	0	1	0	1	0	1	0	1	2	3	4	5	7	8	1	0	0
H3	0	1	0	1	0	1	0	1	0	1	0	1	2	3	4	6	7	0	1	0
	10	1	9	2	8	3	7	4	6	5	4	5	7	10	3	2	8	0	0	1

Error detection simply takes place by checking if the received word r = (r1, r2 to r20) satisfies the three parity check equations.

Encoding can for example be done as follows: Choose c1, c2,..., c17 in any way. If we define

```
c18 = - ( 1*c1 + 0*c2 + 1*c3 +...+ 8*c17) modulo 11
c19 = - ( 0*c1 + 1*c2 + 0*c3 +...+ 7*c17) modulo 11
c20 = - (10*c1 + 1*c2 + 9*c3 +...+ 8*c17) modulo 11
```

then (c1,c2,... c20) is a codeword. We can view c18, c19 and c20 as parity check digits. Note that we may restrict c1, c2,..., c17 to be any of the numbers 0, 1, 2,..., 9. Any of the three parity check digits can be "10". This "10" can be represented by an alphanumerical character different from 0, 1,..., 9, for example X or Z.

Decoding is done by:

```
s18 = ( 1*c1 + 0*c2 + 1*c3 +...+ 1*c18) modulo 11
s19 = ( 0*c1 + 1*c2 + 0*c3 +...+ 1*c19) modulo 11
s20 = (10*c1 + 1*c2 + 9*c3 +...+ 1*c20) modulo 11
```

Summarizing, the code defined with H3 detects all errors of any of the following types:

- Single and double substitution errors.
- Single and double transposition errors.
- Any combination of a single substitution error and a single transposition error.
- All three consecutive substitution errors.

where a transposition is ab => ba and a substitution is a => b.

EXAMPLE:

NOTE: Following example illustrates the use of the algorithm on valid UDN as input number.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
position (n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
input number	8	5	6	2	8	7	0	1	2	1	5	3	2	9	5	6	7				choose a digit (0..9)
matrix H3	1	0	1	0	1	0	1	0	1	0	1	2	3	4	5	7	8	1	0	0	line for C18 and S18
	0	1	0	1	0	1	0	1	0	1	0	1	2	3	4	6	7	0	1	0	line for C19 and S19
	10	1	9	2	8	3	7	4	6	5	4	5	7	10	3	2	8	0	0	1	line for C20 and S20
coding																					checkdigit = -sum(n1..n17) mod 11
C18	8	0	6	0	8	0	0	0	2	0	5	6	6	36	25	42	56				***9***
C19	0	5	0	2	0	7	0	1	0	1	0	3	4	27	20	36	49				***10***
C20	80	5	54	4	64	21	0	4	12	5	20	15	14	90	15	12	56				***2***
codeword	8	5	6	2	8	7	0	1	2	1	5	3	2	9	5	6	7	9	10	2	
decoding																					checkdigit = +sum(n1..n18 or n19 or n20) mod 11
S18	8	0	6	0	8	0	0	0	2	0	5	6	6	36	25	42	56	9	0	0	***0***
S19	0	5	0	2	0	7	0	1	0	1	0	3	4	27	20	36	49	0	10	0	***0***
S20	80	5	54	4	64	21	0	4	12	5	20	15	14	90	15	12	56	0	0	2	***0***

Please take note that the value "10" of checksum digit C19 can be represented by an alphanumerical character different from {0,1,...,9}, for example X or Z.

B.15.2 ARC Checksum

Definition

The checksum on the ARC is calculated by *F-SDN*

Take *n*=12, *r*=2 and *p*=11. We consider the code defined by the *r*=2 following check equations:

```
8*c1 + 8*c2 + 6*c3 +...+ 1*c11 = 0 (modulo 11)
3*c1 + 6*c2 + 4*c3 +...+ 1*c12 = 0 (modulo 11)
```

In other words, a string (*c*1, *c*2 *to c*12) with elements from Z11is a codeword if and only if it has inner product zero (modulo 11) with both rows of the following matrix *H1*:

	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	n11	n12
H1	8	8	6	5	10	5	6	4	1	4	1	0
	3	6	4	2	6	8	2	1	2	4	0	1

Error detection simply takes place by checking if the received word r = (r1, r2,...,r12) satisfies the two parity check equations.

Encoding can for example be done as follows: choose c1, c2,...,c10 in any way. If we define

```
c11 = - ( 8*c1 + 8*c2 + 6*c3 +...+ 4*c10) modulo 11
c12 = - ( 3*c1 + 6*c2 + 4*c3 +...+ 4*c10) modulo 11
```

then (c1, c2,...,c12) is a codeword. We can view c11 and c12 as parity check digits. Note that we may restrict c1, c2 to c10 to be any of the numbers 0, 1, 2,...,9. Any of the two parity check digits can be "10". This "10" can be represented by an alphanumerical character different from 0, 1,...,9, for example X or Z.

Decoding is done by:

```
c11 = ( 8*c1 + 8*c2 + 6*c3 +...+ 1*c11) modulo 11
c12 = ( 3*c1 + 6*c2 + 4*c3 +...+ 1*c12) modulo 11
```

From this table, we draw the following conclusions.

- All single and double substitution errors are detected.
- All single and double transposition errors are detected.

- Any combination of a substitution error in position 12, and transposition error in positions not involving position 12 is detected.
- A substitution error not in position 12 'matches' exactly one transposition error. About 1 % not detected.

where a transposition is ab => ba and a substitution is a => b.

EXAMPLE:

NOTE: Following example illustrates the use of the algorithm on valid ARC as input number.

position (n)	1	2	3	4	5	6	7	8	9	10	11	12	
input number	1	6	6	0	8	7	3	1	0	1			choose a digit (0..9)
matrix H1	8	8	6	5	10	5	6	4	1	4	1	0	line for C11 and S11
	3	6	4	2	6	8	2	1	2	4	0	1	line for C12 and S12
coding													checkdigit = -sum(n1..n10) mod 11
C11	8	48	36	0	80	35	18	4	0	4			***9***
C12	3	36	24	0	48	56	6	1	0	4			***9***
codeword	1	6	6	0	8	7	3	1	0	1	9	9	
decoding													checkdigit = +sum(n1..n11 or n12) mod 11
S11	8	48	36	0	80	35	18	4	0	4	9	0	***0***
S12	3	36	24	0	48	56	6	1	0	4	0	9	***0***

B.16 RSA Signatures under PKCS#1

RSA signatures are made as described by the implementation guidelines of [PKCS #1] v2.1: RSA Cryptography Standard, *RSA Laboratories, June 14, 2002.*

The scheme is RSA + SHA1. There are two choices described in the [21] as they are RSASSA-PSS and RSASSA-PKCS1-v1_5.

Since OMA DRM 2.0 is used for interactive mode of operation and uses RSASSA-PSS, the present document will also use RSASSA-PSS to sign the binary messages for broadcast mode of operation.

B.17 C-style Types

Void.

B.18 Tag Length Format for keyset_block

B.18.1 TLF Syntax Definition

A Tag Length Format (TLF) is defined to identify the keyset_items in the keyset_block. A keyset_item is identified by following syntax:

<tag> [optional <clarifier>] <length> <keyset_item>

Following values are defined and SHALL be used:

tag values

This is a 4 bit field (bslbf) indicating the tag that uniquely identifies the keyset item.

Table B.130: Defined tag values

Keyset_item	Tag $_{(b)}$	remark
UGK	0000	
SGK	0001	
UDK	0010	
UDF	0011	
BDK	0100	
SBDF	0101	shortform_domain_id
LBDF	0110	
RIAK	0111	
TDK	1 000	
reserved for future use	1 001 to 1 111	not used in the present document.

NOTE:

- The keyset items SHALL be included in the order of table B.130.
- The keyset SHALL include at most one instance of the following keys: UGK, UDK, UDF, RIAK and TDK.
- If included the SGKs (8 or 9) SHALL follow in fashion SGK1..n.
- The keyset MAY include zero or more domain sets (BDK, SBDF, LBDF). If included the SBDF SHALL follow the BDK it belongs to, followed by the optional LBDF that belongs to the aforementioned SBDF.

clarifier (optional)

This is a 10 bit field (bslbf) can be used to indicate the following possible values:

- In case the preceding <tag> value indicates a SGK, this field represents the position of a SGK in the Fiat Naor tree.
- In case the preceding <tag> value indicates a LBDF this field represents the length on the LBDF.
- Future extensions to the present document shall always use tag + clarifier + length + keyset_item.

describing the use of the clarifier field for position of SGK

If keyset_item == 001 (i.e. SGK) then the optional field 'clarifier' SHALL indicate the position of the SGK as a node in the [FIAT NAOR] tree. When m = group size, then n = 2 log (m), where n is number of SGKs in tree. Possible positions for the SGKs in the tree are $2^{(n+1)}$ -1 . Therefore parameter 'position' is expressed with 10 bits to express 1 023 nodes in a tree. First MSB left will be used as binary indicator to indicate if the SGK position is a node (0, zero) or a leaf (1, one). Bit positions 2..10 (from left to right LSB) are used in binary format as an indication of the node and leaf position. Nodes and leafs SHALL be numbered according to following picture B.66:

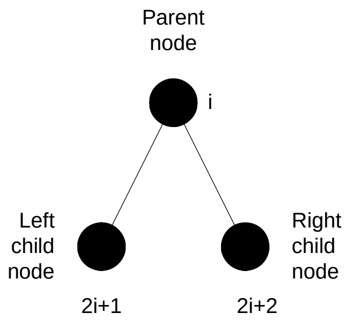


Figure B.66: node numbering

Key:

The root key R is numbered zero. Node keys NK are sequentially numbered per 'level' in a breadth-first manner from left to right, starting from the root node with number 0.

describing the use of the clarifier for length of LBDF

If LBDF is included the optional field 'clarifier' describes the variable length of the LBDF in bytes, as described in clause B.18.3.

length values

This is a 3 bit field (bslbf) indicating the length of a keyset item.

Table B.131: Defined length values

(key)length prescriber	Length $_{(b)}$	Remark
128 bit AES	000	
192 bit AES	001	
256 bit AES	010	
5 byte Eurocrypt [16]	011	
6 byte	100	SBDF
reserved for future use	101 to 111	not used in the present document

NOTE: In case of the LBDF there is no extra length field, since the length value is indicated by the clarifier.

B.18.2 TLF Examples

E.g.1: A 5 byte Eurocrypt [16] address implementing the UDF will be coded like:

```
<0011> <011> <UDF>
```

E.g.2: A 48 bits SBDF address will be coded like:

```
<0101> <100> <SBDF>
```

E.g.3: A LBDF address of 105 bytes will be coded like:

```
<0110> <0001101001> <LBDF>
```

E.g.4: A 128 but AES key implementing the UGK will be coded like:

```
<0000> <000> <UGK>
```

E.g.5: A 128 bit AES key implementing the SGK on node position NK5 in figure B.65 will be coded like:

```
<0001> <0000000101> <000> <SGK>
```

E.g.5: A 128 bit AES key implementing the SGK on node position NK7 (i.e. D0) in figure B.65 will be coded like:

```
<0001> <1000000000> <000> <SGK>
```

B.18.3 LBDF Syntax

In OMA DRM 2.0 the domain ID can be 1 to 17 characters (any) followed by 3 digit characters.

The string that forms the identifier is encoded normally in ROAP messages using UTF-8. UTF-8 character encoding for ASCII characters is 'efficient' with 1 byte per character. On the other hand, there are characters that are encoded using 6 bytes (Asian languages).

The 17 XML UTF-8 characters are translated into bytes as follows:

Longest OMA DRM 2.0 domain identifier encoded as bytes is 6×17+3 bytes = 105 bytes.

Shortest domain identifier is 4 bytes.

B.19 Message_tag Overview

The messages that are defined in the present document SHALL use following message_tag values.

Table B.132: message_tag overview

Message name	message_tag	Described in clause
BCRO()	0x20	B.3.3.4
device_registration_response()	0x01	B.3.4.3.7.2
re_register_msg()	0x11	B.3.4.3.7.3
update_ri_certificate_msg()	0x12	B.3.4.3.7.4
update_drmtime_msg()	0x13	B.3.4.3.7.5
update_contact_number_msg()	0x14	B.3.4.3.7.6
domain_registration_response()	0x02	B.3.4.4.4.1
domain_update_response()	0x03	B.3.4.4.4.2
join_domain_msg()	0x15	B.3.4.4.4.3
leave_domain_msg()	0x17	B.3.4.4.4.4
token_delivery_response()	0x30	B.3.4.5.4.1

B.20 Authentication of the tokens_consumed Field in the Token Consumption Data

Devices SHALL authenticate the tokens_consumed field and the device_nonce of the token consumption report, see clause B.3.4.3.3.4 in the way specified in this clause. The hash function used here is one of the four secure hash functions from [51], page 449 (the upper left one in figure 18.9).

The maximum amount of tokens that can be reported as consumed is 9 999. The amount of tokens, with the before mentioned restriction, is represented with a 14 bit uimsbf number and called tokens_consumed. The value of device_nonce can be in value between 0 and 9 and is represented as a 4 bit uimsbf number. The 14 bit number tokens_consumed is right concatenated with the 4 bit number device_nonce. The resulting 18 bit number is right padded with 0x1 and right padded again with 109 binary zeroes (so 2^{109}). The resulting 128 bit number is used as the input for a single AES block. The Report Authentication Key, as obtained with the token delivery response message, see clause B.3.4.5.4.1, is used as the key input for the AES block. The 128-bit output of the AES block is EXOR-ed with the 128-bit input of the AES block. The left-most 43 bits of the result of this EXOR operation are taken as the report_authentication_code. The 13 digit decimal representation of these 43 bits, including any leading zeroes is used as the report_authentication_code in the token consumption report. See also the next figure.

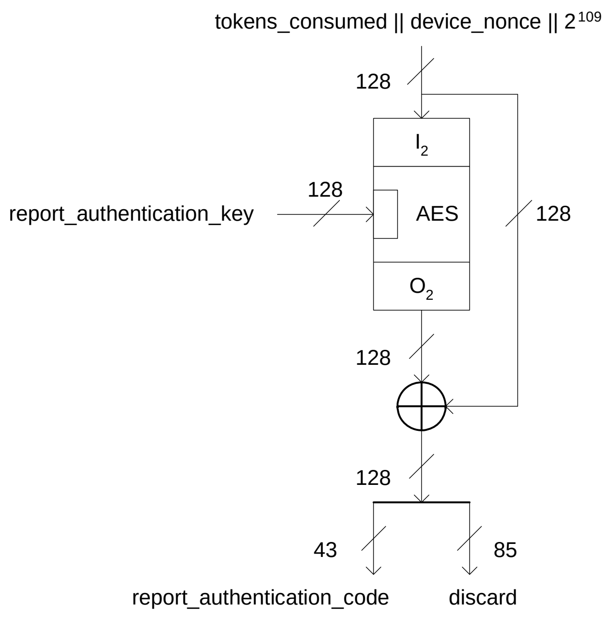

tokens_consumed || device_nonce || 2^109
128
I2
128
report_authentication_key
AES
128
O2
128
128
43
85
report_authentication_code
discard

Figure B.67: Computation of the report_authentication_code

B.21 Management of Tokens by RIs and Devices

B.21.1 Token Management by RIs

There are two business models for the use of tokens.

The first business model is that all tokens ordered by the user are paid for by the user. These tokens are pre-paid tokens. The second business model is that a user orders tokens, but only wants to pay for the ones he actually consumes. These tokens are post-paid tokens.

The tools to support these two business models are the token delivery response message, see clause B.3.4.5.4.1 and the token reporting protocol, see clause B.3.4.5.3. The next clauses describe how these tools can be used by an RI to support the above two business models and how one can switch from one business model to the other.

B.21.1.1 Pre-paid Token Business Model

Setting the token_reporting_flag in the token delivery response message to 0x0 will signal to the device that it does not now nor in the future have to report anymore on the consumption of any of the tokens received so far from this RI.

Therefore in the pre-paid token business model, where the user has agreed to be billed for the delivery of the tokens and their consumption need not be reported, the RI will set the token_reporting_flag to 0x0.

Tokens that are delivered from an RI to a device with a token delivery response message which token_reporting_flag has been set to 0x0 can be called pre-paid tokens.

B.21.1.2 Post-paid Token Business Model

Setting the token_reporting_flag in the token delivery response message to 0x1 will signal to the device that it SHALL report on the consumption of these tokens.

NOTE: Although a broadcast device can only display the token consumption message to the user and must rely on the user to report this message to the RI, the wording in this section is as if the device does the reporting.

Therefore in the post-paid token business model, where the user has to be billed for the actual consumption of the tokens and their actual consumption SHALL be reported by the device, the RI will set the token_reporting_flag to 0x1.

Tokens that are delivered from an RI to a device with a token delivery response message which token_reporting_flag has been set to 0x1 can be called post-paid tokens.

In the post-paid token business model, the RI can limit its risk, by making sure that a device at all times only contains post-paid tokens up to a certain maximum, the so called credit-limit. Furthermore, the RI can set a date/time limit in the device after which the device is not allowed to consume post-paid tokens any more. This can be done as follows:

1) The RI sends in the first token delivery response message a number of tokens equal to the credit-limit. Furthermore, the RI sets the token_reporting_flag in the token delivery response message to 0x1 and sets the latest_consumption_time to a suitable date/time.

2) The RI waits for the reception of a token consumption message.

3) If the RI receives a token consumption message, it SHALL check the authenticity of the tokens_consumed field. If the authentication fails, go to step 2, otherwise continue with step 4.

4) For reasons explained in clause B.21.1.3, if the reported number of consumed tokens is higher than the credit-limit, the RI SHALL assume that only a number of post-paid tokens equal to the credit-limit have been consumed by the device.

5) The RI bills the user for the amount of post-paid tokens consumed with a maximum equal to the credit-limit.

6) The RI sends a token delivery response message a number of tokens equal to the amount of post-paid tokens consumed with a maximum equal to the credit-limit. Furthermore, the RI sets the token_reporting_flag in the token delivery response message to 0x1 and sets the latest_consumption_time to a suitable date/time.

7) Go to step 2.

Note that in the above, the use of the response_flag, device_nonce, earliest_reporting_time, latest_reporting_time and other fields has not been included.

Note further that the RI can force the creation of a token consumption message by sending a token delivery response message with its status field set to 'TokenConsumptionMessageError' or to 'NoTokenConsumptionMessage'.

B.21.1.3 Switching from the Pre-paid Token Business Model to the Post-paid Token Business Model

When at a certain point of time, the user asks the RI to switch from the use of pre-paid tokens to the use of post-paid tokens and the RI agrees, the RI starts at step 1 in the previous clause. The device will report the actual consumption of tokens that will delivered to it in step 1 and in all steps 6. However, at the time of executing step 1, the device MAY still have some pre-paid tokens. Based on the implementation of the device, these pre-paid tokens MAY also be reported as consumed by the device, see clause B.21.2. Because the RI knows that a device never holds more post-paid tokens than the credit limit, the RI SHALL assume that at most an amount of tokens equal to credit_limit have been consumed by the device. Hence step 4 in clause B.21.2.

B.21.1.4 Switching from the Post-paid Token Business Model to the Pre-paid Token Business Model

When at a certain point of time, the user asks the RI to switch from the use of post-paid tokens to the use of pre-paid tokens, and the RI agrees, the RI has a few options

One option is that the RI bills the user for the amount of post-paid tokens that were left in the device at the time of the last token consumption message. These tokens have in effect then become pre-paid tokens. The RI will set the token_reporting_flag in the next token delivery response message to 0x0 and set the value of token_quantity to zero or to the amount of tokens that the user wished to purchase in addition to the amount of post-paid tokens left in the device (encrypting token_quantity yields the encrypted_token_quantity field).

Another option, useful e.g. when the previous option turns out to be expensive, is that the RI performs the actions described in clause B.21.1.5 for clearing the post-paid tokens left in the device. After clearing the post-paid tokens, the RI can start sending pre-paid tokens to the device if the user wishes to purchase these.

B.21.1.5 Stopping the Post-paid Token Business Model

When at a certain point of time, the user informs the RI that he does no longer wish to use post-paid tokens, not even the ones that are still in his device, the RI can do the following. The RI sends the device a token delivery response message with:

- the value of token_quantity set to zero (encrypting token_quantity yields the encrypted_token_quantity field), or set the token_quantity_flag to 0x0;
- the token_reporting_flag field set to 0x1;
- the latest_token_consumption_time set to a date/time in the past;
- the status field to 'NoTokenConsumptionMessage'.

This forces the device to generate a token consumption report. Using this, the RI determines how many post-paid tokens are still in the device. The RI sends the device a token delivery response message with:

- the value of token_quantity set to minus the amount of post-paid tokens left in the device (encrypting token_quantity yields the encrypted_token_quantity field);
- the token_reporting_flag field set to 0x1;
- the latest_token_consumption_time set to a date/time in the past;
- the status field to 'Success'.

The above message MAY be repeated several times. After reception of this token delivery message, the device will have no post-paid tokens left. Any remaining pre-paid tokens can still be consumed by the device.

B.21.2 Token Management by Devices

Each RI context in a device SHALL at a minimum contain:

- A token purse, which is incremented with the tokens received from the RI and which is decremented with the amount of tokens required for each requested consumption of metered protected content from the corresponding RI. If a decrement would yield an accumulator value of less than zero, the device SHALL deny the requested consumption of metered protected content from the corresponding RI.
- A token consumption accumulator, which initially starts at zero.
- The token purse initially starts at zero.

Whenever a device receives from an RI a token delivery response message that has its token_reporting_flag set to 0x0, the device sets the token consumption accumulator associated with the RI to zero and SHALL consider all tokens in the token purse associated with the RI as pre-paid tokens.

In case of the reception of a (series of) token delivery response message with the token_reporting_flag set to 0x1, there are 2 possible implementations of token consumption registration.

A device with a simple implementation of token consumption registration would do the following:

1) Whenever tokens are required and available for consumption, then in addition to decrementing the token purse associated with the RI, the device also increments the token consumption accumulator associated with the RI with the same amount.

2) When a device receives a token delivery response message with status 'Success' and a token_reporting_flag set to 0x1, the device SHALL schedule the creation of a token consumption report at a suitable time, keeping in mind the value in the field latest_consumption_time and possibly the values in the fields earliest_reporting_time and latest_reporting_time.

   If the response_flag was set to 0x1, the device SHALL decrement the token consumption accumulator associated with the RI with the amount of tokens reported in the token consumption report that this token delivery response message was a response to. The device MAY delete that token consumption report and all others sent before that token consumption report was sent.

The device SHALL increment the token purse associated with the RI with the number of tokens indicated in the encrypted_token_quantity field of this token delivery response message.

3) When a device receives a token delivery response message with status 'TokenConsumptionMessageError' or with status 'NoTokenConsumptionMessage', the device SHALL immediately create a token consumption report.

4) When a device creates a token consumption report, it uses a device_nonce which is one higher than the previously used device_nonce. If the previously used device_nonce was 9, the device uses 0 as the next device_nonce.

5) When a device creates a token consumption report, it uses the value of the token consumption accumulator associated with the RI as the amount of tokens to report as consumed.

6) Token consumption reports SHALL be stored by the device, at least until the device receives a token delivery response message indicating that they have been successfully been processed by the RI or indicating that later created token delivery messages have been successfully been processed by the RI.

7) The device SHALL stop with the execution of the above actions when it receives a token delivery response message with the token_reporting_flag set to 0x0. The device SHALL then set the token consumption accumulator associated with the RI to zero and MAY delete all created token consumption reports.

The device actions above imply that the RI will consider the first credit_limit tokens reported as consumed to be post-paid tokens, even though the device might still have had pre-paid tokens. Furthermore, if the current date/time is past the date/time set in the latest_token_consumption_time field, a device is not allowed to use tokens any more. Since a device according to the above rules does not keep track separately of the pre-paid tokens, it cannot use tokens anymore even though the device might still have had pre-paid tokens.

With a slightly more complex implementation, the above two disadvantages can be solved. The device can then first consume all pre-paid tokens before consuming any post-paid tokens. To this end, the device would implement two token purses per RI, a pre-paid token purse and a post-paid token purse for tokens that have been delivered to the device with token delivery response messages with the token_reporting_flag set to 0x1. The device can first consume all tokens in the pre-paid token purse. Consumption of tokens from the pre-paid token purse will not influence the value of the token consumption accumulator and this consumption will therefore not be reported by the device. Whenever the device consumes tokens from the other token purse, the token consumption accumulator SHALL be incremented with the same amount. A device according to this implementation, upon the reception of a token delivery response message with the token_reporting_flag set to 0x0, SHALL increment the pre-paid token purse with the tokens in the post-paid token purse, SHALL set the post-paid token purse as well as the token consumption accumulator to zero.

B.22 ESG and CDP Signalling

The present document requires certain information to be signalled in the ESG and CDP data delivered alongside protected services. This clause describes the formatting of this data.

The ESG is fully described in [35], in clause B.5.4 and in this clause.

The CDP is fully described in [36]

B.22.1 Data Mapping

B.22.1.1 Signalling of Service Operation Centre

All Service Information Centre data is included in *PurchaseChannel* fragment, in the Private Data element, using *ServiceOperationCentreType*.

```
    <complexType name="ServiceOperationCentreType">
        <complexContent>
            <extension base="esg:PrivateDataType">
```

```
                <sequence>
                    <element name="socID" type="anyURI"/>
                    <element name="socName" type="mpeg7:TextualType" maxOccurs="unbounded"/>
                    <element name="socKeyURL" type="anyURI"/>
                    <element name="socInfoURL" type="anyURI"/>
                    <element name="socRiProxyURL" type="anyURI" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

In order to signal the socID using a XML type of anyURI the actual socID is concatenated to the following URI: "http://www.18crypt.com/socID:".
Given a socID of e.g. 1234 the socID SHALL be signalled as follows:
<c18:socID>http://www.18crypt.com/socID:1234</c18:socID>

B.22.1.2 Signalling of Customer Operation Centre

All Customer Operation Centre data is included in *PurchaseChannel* fragment, in the Private Data element, using *CustomerOperationCentreType*.

```
    <complexType name="CustomerOperationCentreType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="cocID" type="anyURI"/>
                    <element name="cocName" type="mpeg7:TextualType" maxOccurs="unbounded"/>
                    <element name="cocLocalFlag" type="boolean"/>
                    <element name="cocRekeyingSafetyWindow" type="integer"/>
                    <element name="cocPurchaseURL" type="anyURI" minOccurs="0"/>
                    <element name="cocPortalURL" type="anyURI" minOccurs="0"/>
                    <element name="cocContactInfo" type="mpeg7:TextualType"/>
                    <element name="cocRiID" type="roap:Identifier"/>
                    <element name="cocRiURL" type="anyURI"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

B.22.1.3 Signalling of Service

Service information is given in the *Service* fragment. Service id, name and description are signalled with the respective fields in the Service fragment. ServiceTypeEnum is signalled with the ServiceType element with ServiceTypeCS.

The serviceBaseCID is signalled in Private Data element, using *HorizontalServiceInformationType*.

```
    <complexType name="HorizontalServiceInformationType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="serviceBaseCID" type="anyURI" minOccurs="1"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
```

B.22.1.4 Signalling of ScheduleItem

ScheduleItem information is given in the *ScheduleEvent* fragment, except for name and description which are given in the associated *Content* fragment. The schedule item id SHALL be instantiated.

B.22.1.5 Void

B.22.1.6 Signalling of PurchaseItem and PurchaseData

PurchaseItem information is given in *Purchase* fragment, using *HorizontalPurchaseItemType*. The purchase item id, version, name and description are given in pItemId, pItemName, pItemVersion, and pItemName fields, respectively. The purchase fragment has a reference to a ServiceBundle fragment, which is the sellable item.

pItemServiceId, pItemScheduleId and pItemContentId are not part of this fragment. Instead, service can be sold by instantiating a ServiceBundle fragment that refers just to that service. A scheduleItem can be sold by instantiating a service that contains just one scheduleItem, and selling that service.

SubscriptionTypeEnum values are:

- 1 for continuous subscription;
- 2 for one-time subscription;
- 3 for preview.

```
    <simpleType name="subscriptionTypeEnumType">
        <restriction base="integer">
            <enumeration value="1"/>
            <enumeration value="2"/>
            <enumeration value="3"/>
        </restriction>
    </simpleType>
    <complexType name="HorizontalPurchaseItemType">
        <complexContent>
            <extension base="esg:PurchaseDataBaseType">
                <sequence>
                    <element name="pItemName" type="mpeg7:TextualType" maxOccurs="unbounded"/>

                    <element name="pItemVersion" type="integer"/>
                    <element name="purchaseOption">
                        <complexType>
                            <sequence>
                                <element name="SubscriptionUnit" nillable="true" minOccurs="0">
                                    <complexType>
                                        <attribute name="subscriptionTypeEnum"
                                        type=" c18:subscriptionTypeEnum" use="required"/>
                                        <attribute name="subscriptionDuration"
                                        type="duration" use="required"/>
                                    </complexType>
                                </element>
                                <element name="UnitText" type="mpeg7:TextualType"
                                minOccurs="0" maxOccurs="unbounded"/>
                            </sequence>
                        </complexType>
                    </element>
                </sequence>
                <attribute name="pItemId" type="unsignedInt" use="required"/>
            </extension>
        </complexContent>
    </complexType>
    <simpleType name="subscriptionTypeEnum">
        <union memberTypes="c18:subscriptionTypeEnumType integer"/>
    </simpleType>
```

B.22.2 Rights Issuer Services

B.22.2.1 Signalling of Rights Issuer Services

Rights Issuer Services are signalled using the ServiceType scheme. The ServiceType element is used to identify the Service as an RI Service by referencing "RI Service" in the ClassificationScheme urn:dvb:ipdc:esg:cs:ServiceTypeCS. Additional data for the RI Service is signalled using the RIServiceDataType which is used as an instance of the PrivateData element in ServiceType.

```
<complexType name='RIServiceDataType'>
    <complexContent>
        <extension base='esg:PrivateDataType'>
            <sequence>
                <element name='riID' type='roap:Identifier'/>
                <element name='scheduled' type='boolean'/>
                <element name='drmID' type='anyURI' minOccurs='0'/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

B.22.2.2 Signalling of Scheduled Rights Issuer Service Schedule

Rights Issuer Services can be addressed to certain groups or devices. In order to allow this the PrivateData element of the ContentType is used to add group addressing. Scheduling of these ContentTypes is done by using the ScheduleEventType in the usual way.

```
<complexType name='RIAddressingType'>
    <complexContent>
        <extension base='esg:PrivateDataType'>
            <sequence>
                <element name='EuroCryptGroup' type='unsignedLong'/>
                <element name='EuroCryptGroupMask' type='unsignedLong'/>
                <element name='Device' type='byte' minOccurs='0' maxOccurs='255'/>
                <element name='DeviceAddress' type='unsignedByte' minOccurs='0'/>
                <element name='DeviceAddressMask' type='unsignedByte' minOccurs='0'/>
                <element name='CertificateChainUpdate' type='boolean'/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

B.22.3 Event Scheduling

The present document requires some extra permissions data to be carried related to event scheduling. The following structure is defined.

```
<complexType name='PermissionScheduleEventType'>
    <complexContent>
        <extension base ='esg:ScheduleEventType'>
            <sequence>
                <element name='PermissionCategory' type='unsignedShort' minOccurs='0'/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

B.22.4 Acquisition Data

The key stream is signalled in the ESG acquisition data, as follows:

```
<complexType name='esg:KeyStreamType'>
    <attribute name='IPDCKMSId' type='unsignedShort' use='required' />
    <attribute name='IPDCOperatorId' type='string' use='required' />
</complexType>
```

The *IPDCOperatorId* attribute is mapped *socID*, as defined in the present document.

Furthermore, an additional parameter to the SDP `fmtp` description MAY be added:

Table B.133

Parameter	Mand / Opt	Type	Comments
serviceBaseCID	O	AnyURI	Base ID of the service

B.22.5 XML schema definition

The schema definition below defines the namespace http://www.18crypt.com/2005/XMLSchema and all XML elements used in an ESG.

The schema SHALL take precedence over XML elements declared previously in this clause.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:esg="urn:dvb:ipdc:esg:2005"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001" xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
xmlns:c18="http://www.18crypt.com/2005/XMLSchema"
targetNamespace="http://www.18crypt.com/2005/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <import namespace="http://www.w3.org/XML/1998/namespace"/>
    <import namespace="urn:mpeg:mpeg7:schema:2001"/>
    <import namespace="urn:dvb:ipdc:esg:2005"/>
    <import namespace="urn:oma:bac:dldrm:roap-1.0"/>
    <complexType name="ServiceOperationCentreType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="socID" type="anyURI"/>
                    <element name="socName" type="mpeg7:TextualType" maxOccurs="unbounded"/>
                    <element name="socKeyURL" type="anyURI"/>
                    <element name="socInfoURL" type="anyURI"/>
                    <element name="socRiProxyURL" type="anyURI" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="CustomerOperationCentreType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="cocID" type="anyURI"/>
                    <element name="cocName" type="mpeg7:TextualType" maxOccurs="unbounded"/>
                    <element name="cocLocalFlag" type="boolean"/>
                    <element name="cocRekeyingSafetyWindow" type="integer"/>
                    <element name="cocPurchaseURL" type="anyURI" minOccurs="0"/>
                    <element name="cocPortalURL" type="anyURI" minOccurs="0"/>
                    <element name="cocContactInfo" type="mpeg7:TextualType"/>
                    <element name="cocRiID" type="roap:Identifier"/>
                    <element name="cocRiURL" type="anyURI"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="HorizontalServiceInformationType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="serviceBaseCID" type="anyURI"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <simpleType name="subscriptionTypeEnumType">
        <restriction base="integer">
            <enumeration value="1"/>
            <enumeration value="2"/>
            <enumeration value="3"/>
        </restriction>
    </simpleType>
    <complexType name="HorizontalPurchaseItemType">
        <complexContent>
            <extension base="esg:PurchaseDataBaseType">
                <sequence>
```

```
                    <element name="pItemName" type="mpeg7:TextualType" maxOccurs="unbounded"/>
                    <element name="pItemVersion" type="integer"/>
                    <element name="purchaseOption">
                        <complexType>
                            <sequence>
                                <element name="SubscriptionUnit" nillable="true" minOccurs="0">
                                    <complexType>
                                        <attribute name="subscriptionTypeEnum"
                                        type="c18:subscriptionTypeEnum" use="required"/>
                                        <attribute name="subscriptionDuration" type="duration"
                                        use="required"/>
                                    </complexType>
                                </element>
                                <element name="UnitText" type="mpeg7:TextualType" minOccurs="0"
                                maxOccurs="unbounded"/>
                            </sequence>
                        </complexType>
                    </element>
                </sequence>
                <attribute name="pItemId" type="unsignedInt" use="required"/>
            </extension>
        </complexContent>
    </complexType>
    <simpleType name="subscriptionTypeEnum">
        <union memberTypes="c18:subscriptionTypeEnumType integer"/>
    </simpleType>
    <complexType name="RIServiceDataType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="riID" type="roap:Identifier"/>
                    <element name="scheduled" type="boolean"/>
                    <element name="drmID" type="anyURI" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="RIAddressingType">
        <complexContent>
            <extension base="esg:PrivateDataType">
                <sequence>
                    <element name="EuroCryptGroup" type="unsignedLong"/>
                    <element name="EuroCryptGroupMask" type="unsignedLong"/>
                    <element name="Device" type="byte" minOccurs="0" maxOccurs="255"/>
                    <element name="DeviceAddress" type="unsignedByte" minOccurs="0"/>
                    <element name="DeviceAddressMask" type="unsignedByte" minOccurs="0"/>
                    <element name="CertificateChainUpdate" type="boolean"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
    <complexType name="PermissionScheduleEventType">
        <complexContent>
            <extension base="esg:ScheduleEventType">
                <sequence>
                    <element name="PermissionCategory" type="unsignedShort" minOccurs="0"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

B.23 The Use and Precedence of Programme ROs, Service ROs and the permissions_category

An operator can give a user access to a service using a Service RO. The Service RO is typically valid for a relatively long period, e.g. a month. A Service RO contains permissions and constraints for all programmes in the service.

In the simplest case, the Service RO contains one set of permissions and constraints, which means that all programmes in the service have the same permissions and constraints for users that have received such a Service RO.

An operator might wish to treat certain programmes of a service in a special way. For example, the operator may wish to sell additional permissions to users for specific programmes, or give different restrictions to certain programmes for users who just have the subscription (e.g. more restrictions for a new movie, less restrictions for preview type content). The present document provides two mechanisms to achieve such a requirement.

B.23.1 Use of Multiple Programme ROs in addition to a Service RO

The first mechanism the present document provides for having different permissions and constraints for programmes is to have a Programme RO for each programme for which the permissions and constraints are different from the ones in the encompassing Service RO. Therefore, a Programme RO of a programme SHALL have precedence over a Service RO of the service for which the programme is a part.

These Programme ROs can be broadcast to broadcast only devices and they can be sent over the interactivity channel to devices that have an interactivity channel. The distribution of these Programme ROs can consume much bandwidth in case there are many programmes for which the permissions and constraints are different from the Service RO.

B.23.2 Use of the permissions_category and Service ROs

The second mechanism the present document provides for having different permissions and constraints for programmes is by using the permissions_category functionality. In this second mechanism, only Service ROs are distributed, which can save considerable bandwidth in comparison with the first mechanism, where multiple Programme ROs are distributed.

For BCROs, it is possible to have more than one set of permissions and constraints for each asset in the Service RO. Each of these sets is identified by an 8-bit number which is called the permissions_category. For Service ICROs, the permissions_category is indicated in their Service_CID (see specification of permissions_category, clause B.3.2.1.3). The Key Stream Message can contain a permissions_category number. This number in the Key Stream Message indicates which of the sets of permissions and constraints in the Service BCRO and which Sevice ICRO applies to the programme that is broadcast at the time of reception of the permissions_category number in the Key Stream Message. The permissions_category SHOULD remain constant over one programme.

Using the permissions_category number functionality, it is possible to have a service with programmes with different permissions and constraints by just distributing a Service RO containing multiple sets of permissions and constraints, each set indexed by permissions_category.

The RO as defined in OMA DRM 2.0 does not have the permissions_category functionality. Therefore, the present document contains an enhancement that defines this functionality.

B.23.3 Use of Programme ROs without a Service RO

An operator can give access to the user on a programme by programme basis using Programme ROs, without using a Service RO. This may be useful in, for example, Pay-Per-View business models.

B.23.4 Precedence of Permissions and Constraints in Programme and Service ROs

The permissions and constraints in a Programme RO for a programme in a service SHALL have precedence over any set of permissions and constraints in any Service RO applicable to that service. This is independent of whether or not a set of permissions and constraints is indicated for that programme in the Key Stream Message by the permissions_category field.

In the absence of a Programme RO for a programme in a service, the set of permissions and constraints that is indicated for that programme in the Key Stream Message by the permissions_category field SHALL have precedence over any other set of permissions and constraints in any Service RO that is applicable to the service of which that programme is a part.

If the above two cases do not apply, all permissions and constraints in a Service RO that is applicable to the service of which the programme is a part, and which permissions and constraints do not have a permissions_category field, or have a permissions_category field with the value 0, SHALL have precedence over any other set of permissions and constraints in any Service RO that is applicable to the service of which that programme is a part.

If the above three cases do not apply, all permissions and constraints in the Service RO that is applicable to the service of which the programme is a part apply for that programme i.e. the device can select any permission together with its constraint.

B.24 Conformance Table

The following table summarizes the mandatory and optional requirements for different types of devices, and for broadcasters and Rights Issuers.

Note that this table indicates whether support for each method is mandated. Please see the referenced clause for details of whether the use of a method is mandated in specific circumstances, or whether alternatives are available.

Table B.134

	Reference	**Device with Interactivity channel only**	**Device with Broadcast channel only**	**Mixed-mode device**
Interactive mode registration	B.2.3.2	M	-	M
Broadcast mode registration	B.2.3.3	-	M	M
Mixed-mode registration	B.2.3.4	-	-	M
Addressing a complete group	B.2.6.2.1	-	M	M
Addressing a privileged subset	B.2.6.2.2 and B.2.6.3	-	M	M
Addressing a unique device	B.2.6.2.3	-	M	M
Addressing a broadcast domain	B.2.6.2.4	-	M	M
IPsec stream delivery	6.1	M	M	M
IPsec message authentication	6.1.4	M	M	M
ISMACryp stream delivery	6.2	M	M	M
SRTP stream delivery	6.3	M	M	M
SRTP message authentication	6.3.3	M	M	M
Key stream delivery	B.3.2	M	M	M
Timestamp field in KSM	B.3.2.1.3	M	M	M
Programme key layer in KSM	B.3.2.1.3	M	M	M
Service key layer in KSM	B.3.2.1.3	M	M	M
Access criteria in KSM	B.3.2.1.3	M	M	M
Parental rating in KSM	B.3.2.1.3	M	M	M
Permissions category in KSM	B.3.2.1.3	M	M	M
Access criteria descriptors in KSM	B.3.2.1.3	O	O	O
KSM authentication	B.3.2.1.3	M	M	M
Contents of service and programme RO	B.3.3.1, B.3.3.2	M	-	M
Use of 'datetime' constraint in service and programme ROs	B.3.3.1, B.3.3.2	M	-	M
Scheduling of RO renewals according to datetime constraints	B.3.3.1	O	-	O
Common constraints for all assets in one service RO	B.3.3.1	-	-	-
Post-acquisition usage rules in service or programme RO according to [50]	B.3.3.1, B.3.3.2	M	-	M
Precedence of KSM permissions over RO permissions	B.3.3.1, B.3.3.2	M	-	M
ICRO delivery with ROAP [49]	B.3.3.3	M	-	M
BCRO delivery	B.3.3.4	-	M	M
Parsing of BCRO	B.3.3.4.2	-	M	M
Support for asset object	B.3.3.4.2.1	-	M	M
Support for permission object	B.3.3.4.2.2	-	M	M
Support for action object	B.3.3.4.2.3	-	M	M
play action	B.3.3.4.2.3	-	M	M
display action	B.3.3.4.2.3	-	M	M

	Reference	Device with Interactivity channel only	Device with Broadcast channel only	Mixed-mode device
execute action	B.3.3.4.2.3	-	M	M
print action	B.3.3.4.2.3	-	M	M
export action	B.3.3.4.2.3	-	M	M
access action	B.3.3.4.2.3	-	M	M
Support for constraint objects	B.3.3.4.2.4	-	M	M
Support for count_constraint_descriptor	B.3.3.4.2.4.1	-	M	M
Support for timed_count_constraint_descriptor	B.3.3.4.2.4.2	-	O	O
Support for datetime_constraint_descriptor	B.3.3.4.2.4.3	-	O	O
Support for interval_constraint_descriptor	B.3.3.4.2.4.4	-	O	O
Support for accumulated_constraint_descriptor	B.3.3.4.2.4.5	-	O	O
Support for individual_constraint_descriptor	B.3.3.4.2.4.6	-	O	O
Support for system_constraint_descriptor	B.3.3.4.2.4.7	-	M	M
Support for metering_constraint_descriptor	B.3.3.4.2.4.8	-	O	O
Notify offline requests with detection of vocal errors	B.3.4.3.3	-	M	M
Inform device on forced actions	B.3.4.3.5 and B.3.4.3.7.3 and B.3.4.4.4.3 and B.3.4.4.4.4	-	M	M
Device identification per UDN	B.3.4.3.6	-	M	M
Broadcast device registration data	B.3.4.3.7.2	-	M	M
Broadcast RI certificate update	B.3.4.3.7.4	-	M	M
Broadcast DRM time update	B.3.4.3.7.5	-	M	M
Broadcast contact number update	B.3.4.3.7.6	-	M	M
Joining a broadcast domain	B.3.4.4	-	M	M
Leaving a broadcast domain	B.3.4.4	-	M	M
Broadcast domain registration data	B.3.4.4.4.1	-	M	M
Token handling	B.3.4.5	O	O	O
Reception of Rights Issuer Services	B.4	-	M	M
Broadcast of Rights Issuer Services	B.4	-	-	-
Provision of RI Service Schedule	B.4.2	-	-	-
Scheduling of RI Service reception	B.4.2	-	O	O
Purchase via interactivity channel	B.5.1	O	-	O
Support for all defined transactions if purchase via interactivity channel is supported	B.5.1	M	-	M
Support for join-domain trigger in ROAP registration response	B.5.1.1.5	M	-	M
Support for additional join-domain trigger in purchase response	B.5.1.1.6	M	-	M
Random distribution of renewal requests over time	B.5.1.1.7	M	-	M
Support for HTTP over the interactivity channel	B.5.1.2	M	-	M
Support for HTTPS over the interactivity channel	B.5.1.2	M	-	M
Signing of XML messages using RSASSA-PSS [21]	B.5.1.3	M	-	M
Canonicalization of XML messages according to [33]	B.5.1.3	M	-	M
Announcement of device capabilities in XML requests	B.5.1.3.1.2	M	-	M
Usage of previously announced device capabilities if they are not included in a request	B.5.1.3.1.2	O	-	O
Usage of DVB platform ID as socID	B.5.4.1	-	-	-

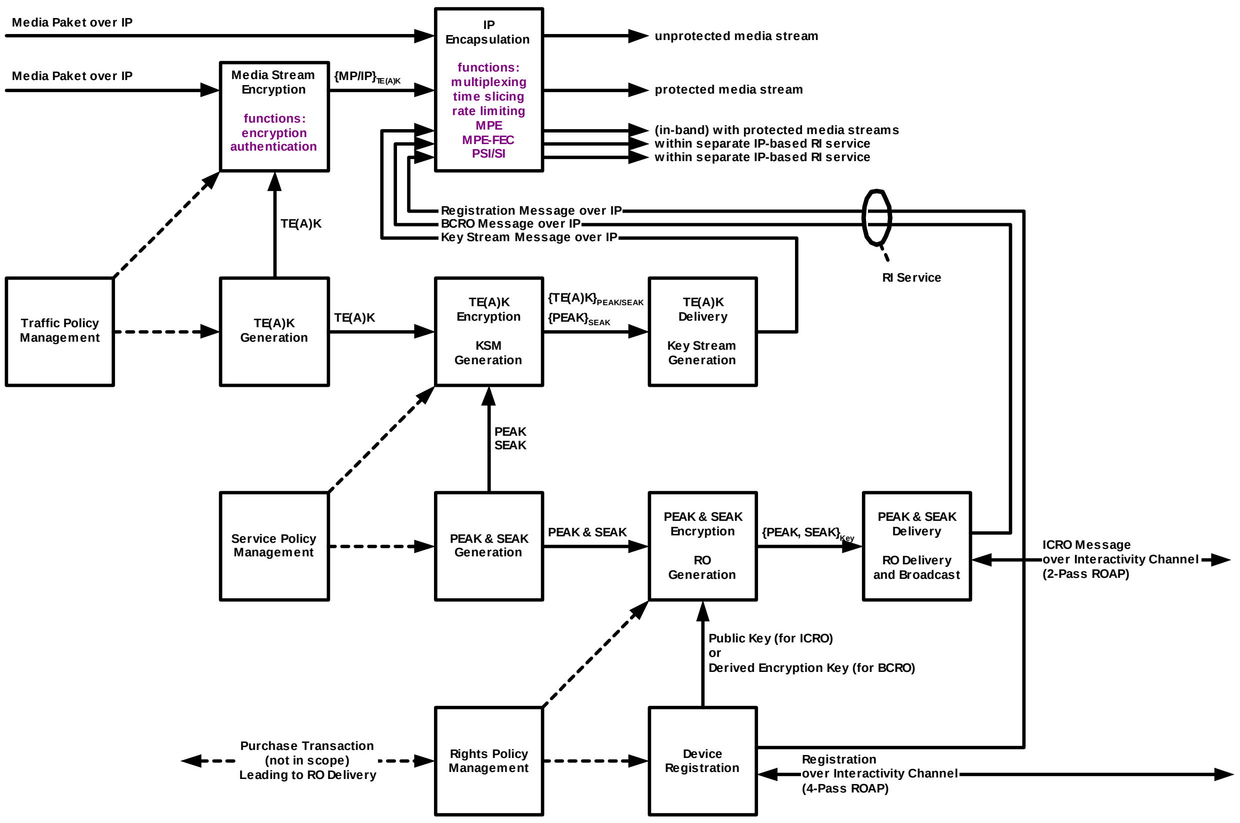
	Reference	Device with Interactivity channel only	Device with Broadcast channel only	Mixed-mode device
Support for all purchase transactions if Purchase URL is announced in service guide	B.5.4.2	-	-	-
Support for out-of-band transactions if Portal URL is announced in service guide	B.5.4.2.	-	-	-
Inclusion of COC data in service guide for each COC with whom the SOC has an agreement	B.5.4.2	-	-	-
Inclusion of purchase availability information in service guide for local COCs	B.5.4.2, B.5.4.7	-	-	-
Inclusion of price information in service guide for local COCs	B.5.4.2, B.5.4.7	-	-	-
Handling weak keys	B.6.1	-	-	-
Handling OCSP grace period	B.6.2	-	M	M
Signalling of Rights Issuer Services	B.22.2.1	-	M	M
Signalling of Rights Issuer Service schedule	B.22.2.2	-	O	O
Event scheduling	B.22.3	M	M	M
Acquisition data	B.22.4	M	M	M
Purchase data	B.22.1.6	M	M	M
Key: M: Support is mandatory O: Support is optional -: Not applicable				

Annex C (informative): 18Crypt

C.1 Head-end for Service Protection

The normative part of the present document defines building blocks that can be very flexibly combined into a real system deployment. A head-end architecture for service protection is proposed in this clause, and some considerations are given that justify the architecture, but this architecture is by no means the only possible one. Therefore, this whole clause is informative in its entirety.

C.1.1 Function Blocks



Media Paket over IP
Media Paket over IP
IP Encapsulation
functions: multiplexing time slicing rate limiting MPE MPE-FEC PSI/SI
unprotected media stream
protected media stream
(in-band) with protected media streams
within separate IP-based RI service
within separate IP-based RI service
Media Stream Encryption
functions: encryption authentication
{MP/IP}TE(A)K
Registration Message over IP
BCRO Message over IP
Key Stream Message over IP
RI Service
TE(A)K
Traffic Policy Management
TE(A)K Generation
TE(A)K
TE(A)K Encryption
KSM Generation
{TE(A)K}PEAK/SEAK
{PEAK}SEAK
TE(A)K Delivery
Key Stream Generation
PEAK SEAK
Service Policy Management
PEAK & SEAK Generation
PEAK & SEAK
PEAK & SEAK Encryption
RO Generation
{PEAK, SEAK}Key
PEAK & SEAK Delivery
RO Delivery and Broadcast
ICRO Message over Interactivity Channel (2-Pass ROAP)
Public Key (for ICRO) or Derived Encryption Key (for BCRO)
Purchase Transaction (not in scope) Leading to RO Delivery
Rights Policy Management
Device Registration
Registration over Interactivity Channel (4-Pass ROAP)

Figure C.1: Function Blocks of Service Protection Head-End

The function blocks are arranged according to layers in the 4-layer model for service protection, and probably do not require further description, except for the policy management functions.

Traffic Policy Management: defines which media streams need encryption, which streams are 'bundled' to share their traffic key material, and the length of the crypto period.

Service Policy Management: defines Services and Programmes and their lifetimes, which 'bundles of media flows' (corresponding to a key stream) they contain, and the access criteria to each key stream. Specifically, this functional block also manages the PEAKs and SEAKs. In DVB terminology, this includes Event Information Scheduling.

Rights Policy Management: defines purchasable items, which are 'bundles of services and programmes'. It also defines to which device or domain a RO is bound and what rights the device (or its owner) has.

C.1.2 Considerations

The traditional security architecture of the DVB head-end, with cell-local key generation for both ECMs (corresponding to KSMs) and EMMs (corresponding to ROs), does not sufficiently cater for the needs of mobile devices, and can therefore not be used one-to-one for DVB-H.

Device mobility adds the requirement that ROs SHALL be valid for the whole network, to avoid the need for RO acquisition upon cell change. Hence, the PEAKs and SEAKs need to be generated centrally, in order for them to be packaged into ROs that have network-global validity.

TE(A)K generation can in principle still be cell-local, but this might have negative effects on cell hand-over, necessitating that, before traffic can be decrypted in the new cell, a KSM needs to be acquired and processed leading to new security associations to be set up. Local TE(A)K generation has furthermore the disadvantage that PEAKs and SEAKs need to be distributed to all encryptors, whereas with central TE(A)K generation, only the TE(A)Ks need to be distributed (optimizations are possible, such as synchronized pseudo-random generators). Therefore, in the proposal below, TE(A)K generation is shown as a central function.

The Media Stream Encryption function can very easily be separated from the IP Encapsulation. It is also possible to combine the Encryption with the Media Stream Generation function, or to separate Media Stream Encryption into its own network element. There are advantages and disadvantages with all options.

- Combining encryption with the generation of Media Streams offers end-to-end protection, but may be costly to deploy, because the Encryptor has to be integrated with all possible Generators used in the network.
- Combining encryption with the encapsulation of Media Streams is very easy to deploy, but then the up-stream traffic has to be protected by different means (this is not such a big down-side, since the threat model is different upstream of the Encapsulator).
- Separating encryption into its own network element gives most flexibility, but then the Encryptor will have to have many functions (e.g. IP routing, multiplexing, possibly rate limiting) in common with the Encapsulator, so that both elements will turn out to be very similar.

In the proposal below, the Media Stream Encryption is combined with the IP Encapsulation into a single network element, but this is by no means a requirement of the architecture.

C.1.3 Proposed Network Elements

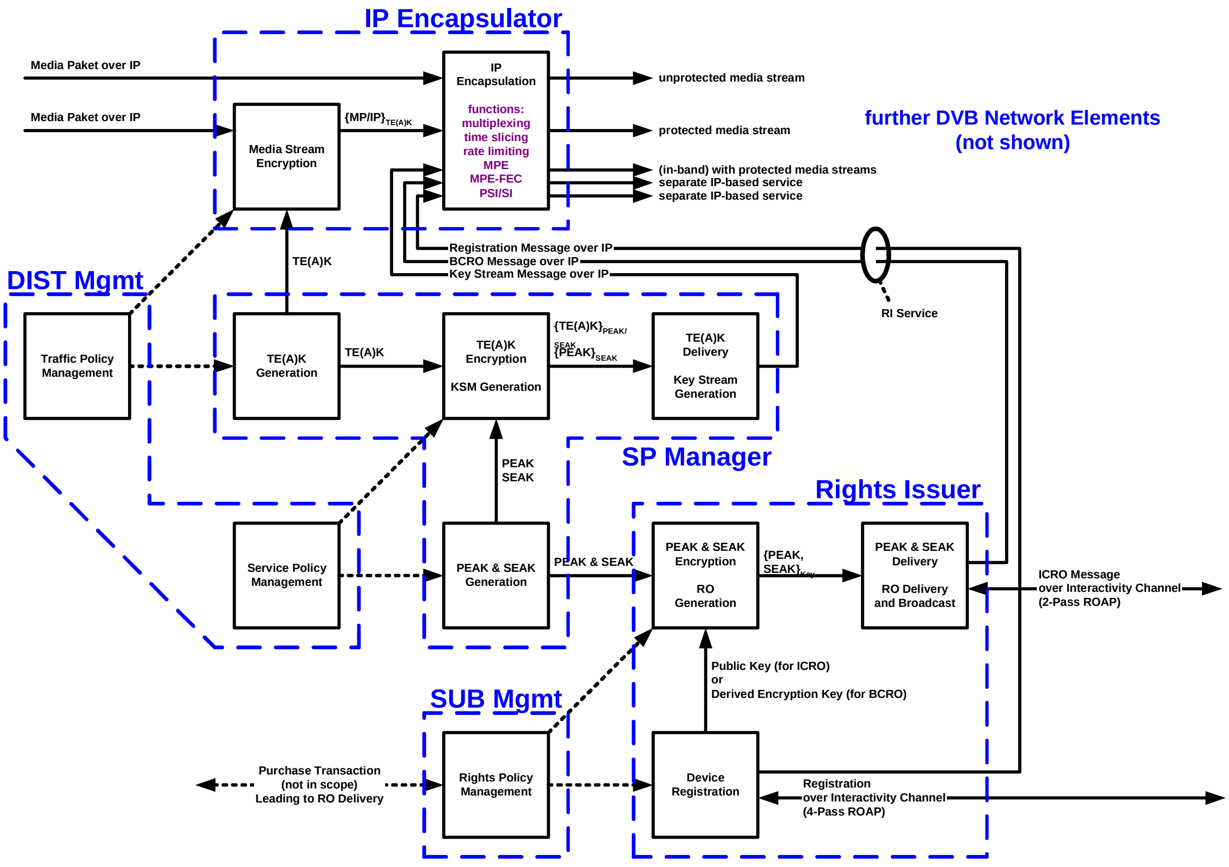

IP Encapsulator
Media Paket over IP
Media Paket over IP
Media Stream Encryption
{MP/IP}TE(A)K
IP Encapsulation
functions: multiplexing time slicing rate limiting MPE MPE-FEC PSI/SI
unprotected media stream
protected media stream
(in-band) with protected media streams
separate IP-based service
separate IP-based service
further DVB Network Elements (not shown)
Registration Message over IP
BCRO Message over IP
Key Stream Message over IP
RI Service
TE(A)K
DIST Mgmt
Traffic Policy Management
TE(A)K Generation
TE(A)K
TE(A)K Encryption
KSM Generation
{TE(A)K}PEAK/SEAK
{PEAK}SEAK
TE(A)K Delivery
Key Stream Generation
PEAK SEAK
SP Manager
Rights Issuer
Service Policy Management
PEAK & SEAK Generation
PEAK & SEAK
PEAK & SEAK Encryption
RO Generation
{PEAK, SEAK}
PEAK & SEAK Delivery
RO Delivery and Broadcast
ICRO Message over Interactivity Channel (2-Pass ROAP)
Public Key (for ICRO) or Derived Encryption Key (for BCRO)
SUB Mgmt
Purchase Transaction (not in scope) Leading to RO Delivery
Rights Policy Management
Device Registration
Registration over Interactivity Channel (4-Pass ROAP)

Figure C.2: Systems and Network Elements of Service Protection Head-End

The proposed head-end architecture for service protection defines the following systems and network elements:

The **IP Encapsulator (IPE)** is a network element that can be deployed cell-local, but also (in case an ASI distribution network is used) network-global, or something in between. The IPE can become a 'box' with well-defined standard interfaces. Upstream, the interface needs to support IP routing functionality, and the physical interfaces can be those of an IP router. Downstream, the interface needs to support MPEG-2 transport streams, and the physical interfaces can be ASI. To what extent existing DVB head-end interface specifications are re-usable for the management interfaces (especially for putting in place TE(A)Ks) needs to be further studied.

The **Service Protection Manager** and the **Rights Issuer** are network-global elements. They may be 'boxes': the functionality is well-defined, and the interfaces are minimal.

The **Distribution Management System** and the **Subscription Management System** are true systems, with differentiation based on their functionality. There are already standards for some of the interfaces, and some more of the interfaces will eventually become standardized (e.g. the interfaces needed for roaming).

C.2 Deployment and Roaming Scenarios

The deployment and roaming scenarios described in this clause build on the purchasing mechanisms (see clause B.5) and rights issuer mechanisms (see clause B.4), and the interfaces these mechanisms offer.

Whereas said mechanisms are very generic and allow a variety of business models to be implemented, roaming cannot be described without making assumptions about deployment options that result from particular business models. It is not possible to capture all possible deployment options in this clause, which is therefore informative in its entirety.

Two scenarios are described that show how roaming can be implemented for both interactive and broadcast mode of operation.

The placement of the Rights Issuer (RI) with respect to the Service Operation Centre (SOC) and the Customer Operation Centre (COC) is the main difference between the two described scenarios: in the first scenario, SOC combines Service Distribution Management (DIST Mgmt) and RI, in the second scenario, COC combines Service Subscription Management (SUB Mgmt) and RI.

In the description of the scenarios, the following important concepts need further explanation:

Locality: A SOC and a COC can be considered local to each other, if:

- - the COC's complete pricing information is included in the SOC's service guide
- - the COC keeps its complete purchasing information current through 'bulk downloads'
- - (if the COC includes an RI) the COC keeps the keys current through 'bulk downloads'

Whereas Locality is not needed from a functionality point of view (a non-local COC can offer the same functionality as a local COC), it can be assumed that thanks to the better user experience and also less network traffic, Locality will be implemented at various degrees.

Home COC: The COC with which the owner of the device has a contractual relationship (aka. 'Subscription').This relationship allows that the owner is charged for the services consumed. In the case where the Home COC includes a Rights Issuer (RI), it can be assumed that the device is already registered and has a valid RI context.

Local COC: A COC that is local to a SOC.

Home SOC: The SOC which is local to the Home COC. In the case where the Home SOC includes an RI, it can be assumed that the device is already registered and has a valid RI context.

Local SOC: A SOC that is local to a COC.

Visited SOC: The SOC from which the device (at home or roaming) receives the Broadcast Services it wants to purchase.

NOTE: The case where the visiting owner of a device enters a contractual relationship with a COC that is local to the Visited SOC is not considered roaming for the sake of this clause (the case is identical to the local scenarios described in this clause; 'only' interoperability between devices and networks is needed to enable this functionality).

C.2.1 Deployment Option A (RI as part of SOC)

The first roaming scenario is based on the deployment option A, where the SOC combines DIST Mgmt and RI.

C.2.1.1 Local scenario

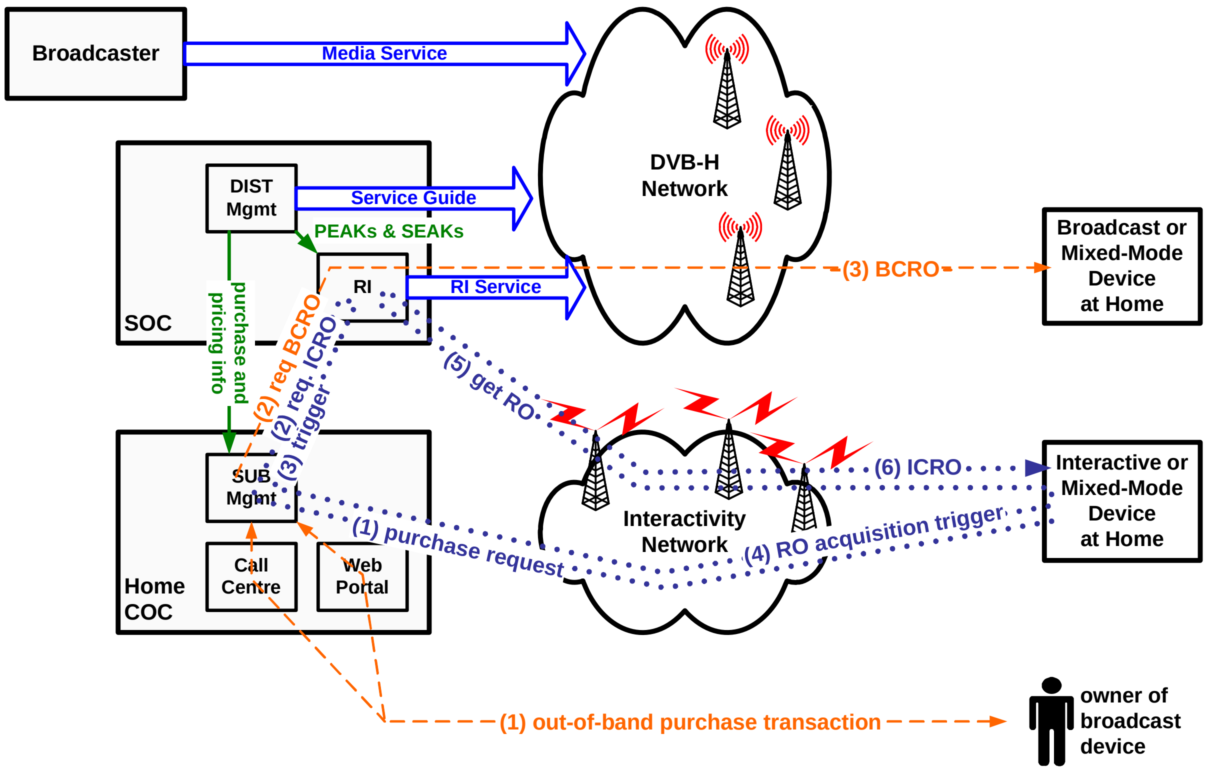


Figure C.3: Deployment Option A (Combining DIST Mgmt and RI in SOC) - Local Scenario

In the **interactive mode of operation**, the device (1) issues a purchase request to the SUB Mgmt of the Home COC. If successful, the SUB Mgmt (2) requests the RI of the SOC to generate an ICRO related to the purchase, and (3) gets an RO acquisition trigger back, which it (4) forwards to the device. The device (5) uses the trigger to request the ICRO directly from the RI, which (6) sends the RO to the device.

In the **broadcast mode of operation**, the owner of the device (1) performs an out-of-band purchase transaction with the SUB Mgmt of the Home COC (e.g. through a call centre, or a web portal). If the transaction is successful, the SUB Mgmt (2) requests the RI of the SOC to generate and broadcast the BCRO related to the purchase. If this is successful, the RI (3) broadcasts the BCRO on the Ad-hoc RI Stream for a while, and then continues repeating it in the Scheduled RI Stream during the whole lifetime of the purchased item.

In the **mixed-mode of operation**, the device (1, in the interactive scenario) issues a purchase request to the SUB Mgmt of the Home COC. If successful, the flow can continue with the interactive mode of operation, or the broadcast mode of operation described above (2, in the respective scenario).

C.2.1.2 Roaming scenario

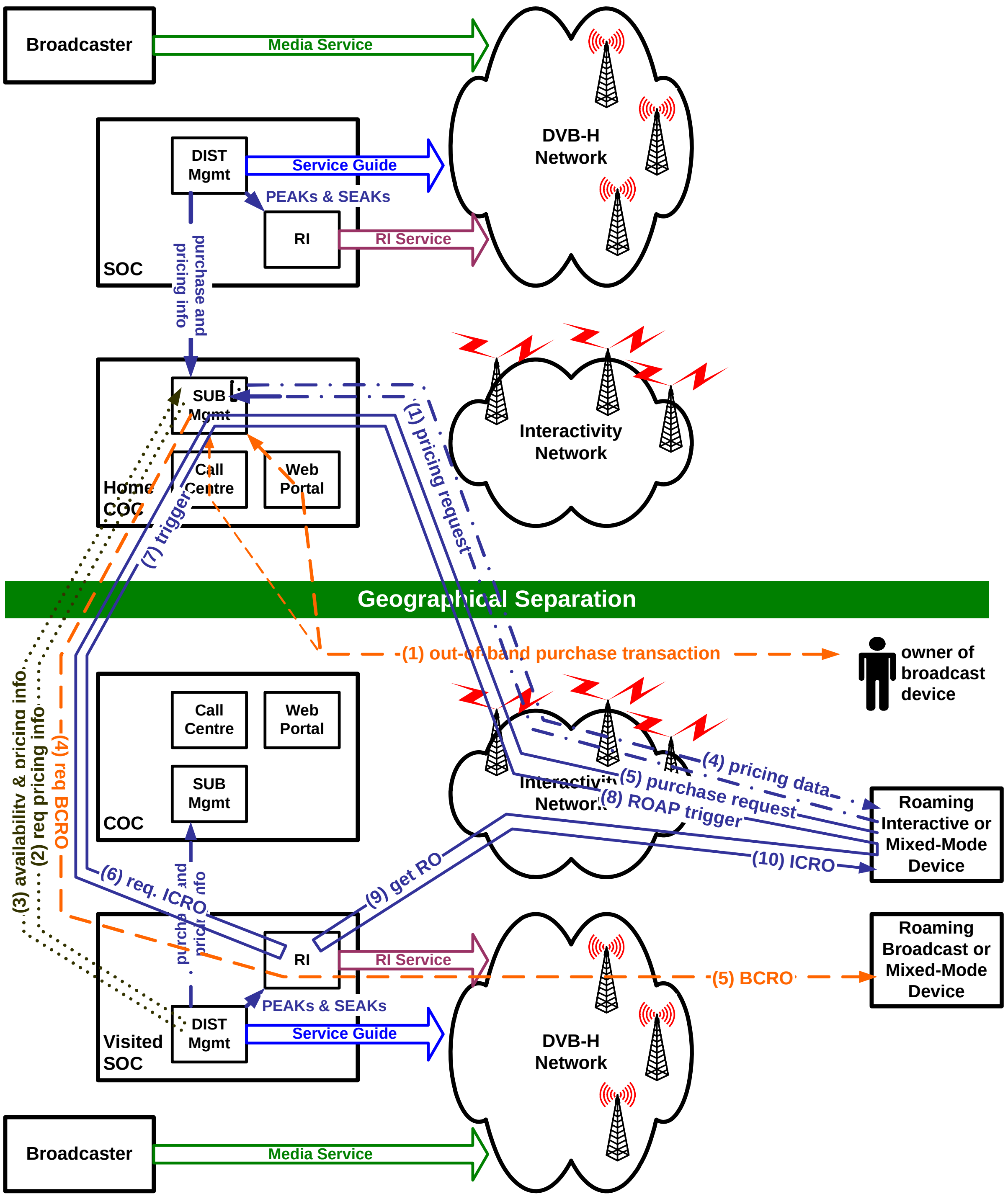


Figure C.4: Deployment Option A (Combining DIST Mgmt and RI in SOC) - Roaming Scenario

In the **interactive mode of operation**, the device (1) first issues a pricing request to the SUB Mgmt of the Home COC, which is now remote to the Visited SOC. The SUB Mgmt of the Home COC (2) requests the availability and pricing information from the Visited SOC, which (3) returns it through the COC (4) to the device. Only now has the device all the information available that enables it (5) to make a purchase request to the SUB Mgmt of the Home COC. If successful, the SUB Mgmt (6) requests the RI of the Visited SOC to generate an ICRO related to the purchase, and (7) gets an RO acquisition trigger back, which it (8) forwards to the device. The device (9) uses the trigger to request the ICRO directly from the RI of the Visited SOC, which (10) sends the RO to the device.

In the **broadcast mode of operation**, the owner of the device (1) performs an out-of-band purchase transaction with the SUB Mgmt of the Home COC (e.g. through a call centre, or a web portal). During the purchase transaction, the SUB Mgmt (2) requests the availability and pricing information from the Visited SOC, which (3) returns it for immediate use in the purchase transaction. If the transaction is successful, the SUB Mgmt of the Home COC (4) requests the RI of the Visited SOC to generate and broadcast the BCRO related to the purchase. If this is successful, the RI of the Visited SOC (5) broadcasts the BCRO on the Ad-hoc RI Stream for a while, and then continues repeating it in the Scheduled RI Stream during the whole lifetime of the purchased item.

In the **mixed-mode of operation**, the device (1) first issues a pricing request to the SUB Mgmt of the Home COC, which is now remote to the Visited SOC. The SUB Mgmt of the Home COC (2) requests the availability and pricing information from the Visited SOC, which (3) returns it through the COC (4) to the device. Only now has the device all the information available that enables it (5) to make a purchase request to the SUB Mgmt of the Home COC. If successful, the flow can continue with the interactive mode of operation (6), or the broadcast mode of operation (4) described above.

C.2.1.3 Conclusion to Deployment Option A

The advantage of this deployment option is that the programme encryption and authentication keys (PEAKs) and service encryption and authentication keys (SEAKs) never leave the Visited SOC.

The differences between the local and the roaming scenario are:

- the additional request for pricing information (the Home COC might cache this information for subsequent usage).

The needs for non-local communication between Visited SOC and Home COC in the roaming scenario are the following:

- retrieval of pricing information;
- request for ICRO creation and retrieval of the corresponding acquisition trigger (interactive mode of operation);
- request for BCRO creation and broadcasting within an RI service (broadcast mode of operation).

C.2.2 Deployment Option B (RI as part of COC)

The second roaming scenario is based on the deployment option B, where the COC combines SUB Mgmt and RI.

C.2.2.1 Local scenario

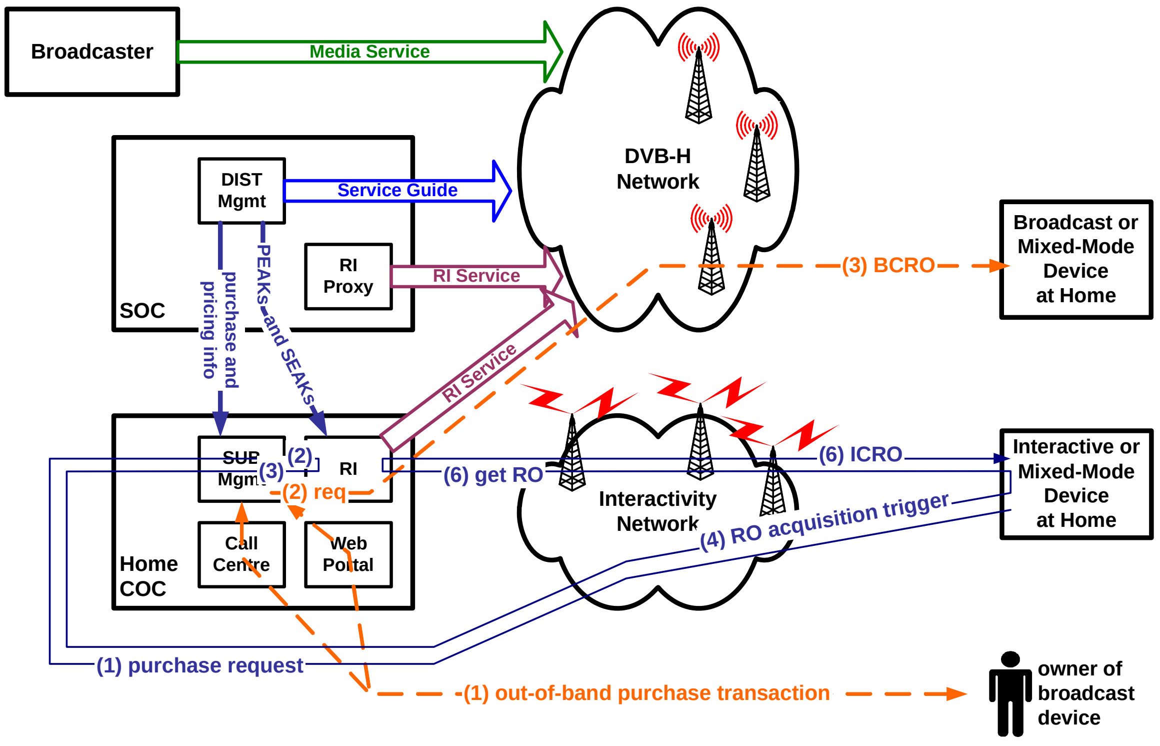


Figure C.5: Deployment Option B (Combining SUB Mgmt and RI in COC) - Local Scenario

In the **interactive mode of operation**, the device (1) issues a purchase request to the SUB Mgmt of the Home COC. If successful, the SUB Mgmt (2) requests its own RI to generate an ICRO related to the purchase, and (3) gets an RO acquisition trigger back, which it (4) forwards to the device. The device (5) uses the trigger to request the ICRO directly from the RI, which (6) sends the RO to the device.

In the **broadcast mode of operation**, the owner of the device (1) performs an out-of-band purchase transaction with the SUB Mgmt of the Home COC (e.g. through a call centre, or a web portal). If the transaction is successful, the SUB Mgmt (2) requests its own RI to generate and broadcast the BCRO related to the purchase. If this is successful, the RI (3) broadcasts the BCRO on the Ad-hoc RI Stream for a while, and then continues repeating it in the Scheduled RI Stream during the whole lifetime of the purchased item.

In the **mixed-mode of operation**, the device (1, in the interactive scenario) issues a purchase request to the SUB Mgmt of the Home COC. If successful, the flow can continue with the interactive mode of operation, or the broadcast mode of operation described above (2, in the respective scenario).

C.2.2.2 Roaming scenario

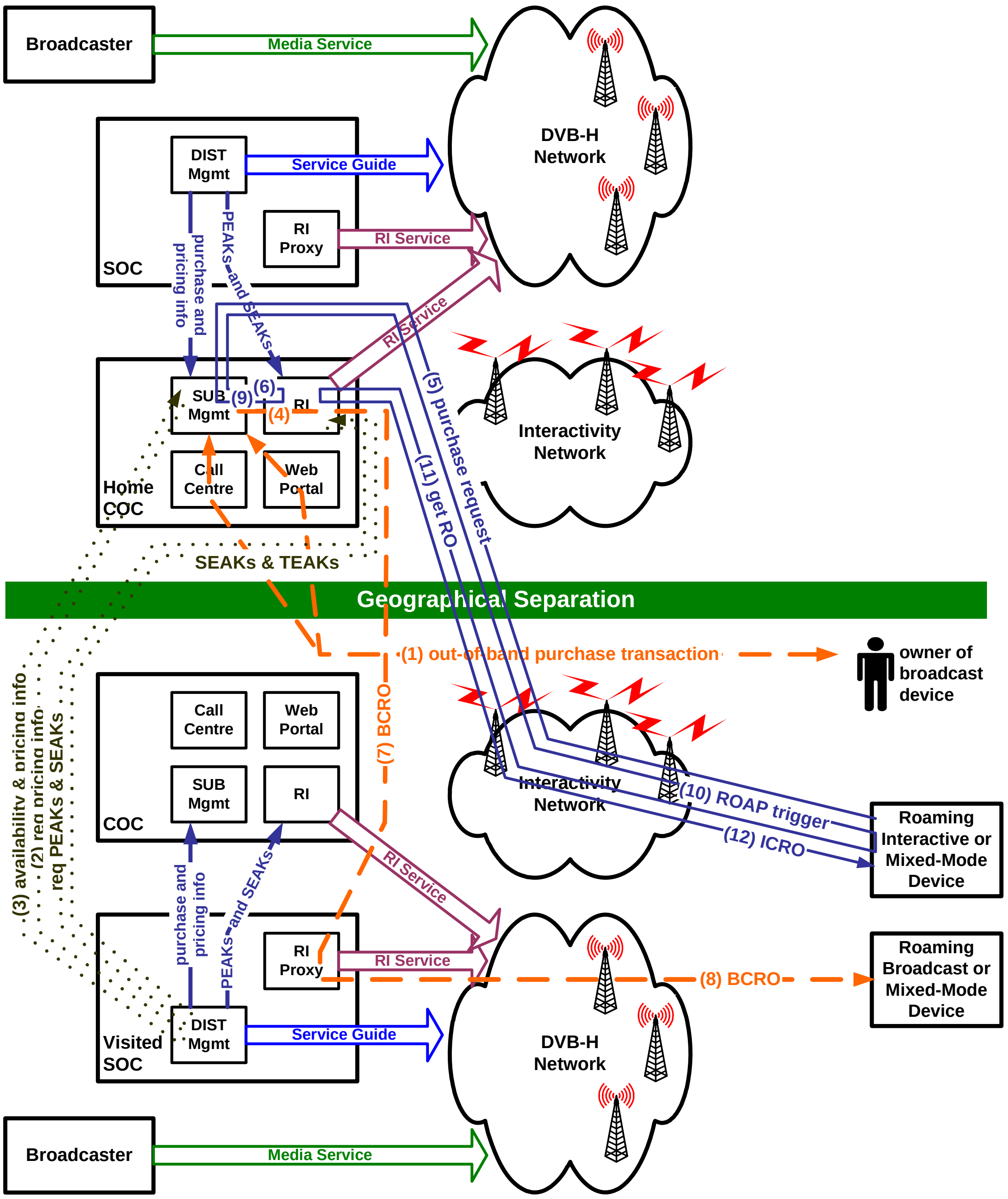


Figure C.6: Deployment Option B (Combining SUB Mgmt and RI in COC) - Roaming Scenario

In the **interactive mode of operation**, the device (1, not shown in figure) first issues a pricing request to the SUB Mgmt of the Home COC, which is now remote to the SOC. The SUB Mgmt of the Home COC (2) requests the availability and pricing information from the Visited SOC, which (3) returns it through the Home COC (4, not shown in the figure) to the device. Only now has the device all the information available that enables it (5) to make a purchase request to the SUB Mgmt of the Home COC. If successful, the SUB Mgmt of the Home COC (6) requests its own RI to generate an ICRO related to the purchase. The RI of the Home COC (7, not shown in the figure) requests the corresponding PEAKs and SEAKs from the DIST Mgmt of the Visited SOC, which (8, not shown in the figure) sends them back to the RI. The RI of the Home COC now generates the ICRO, and (9) returns an RO acquisition trigger back to the SUB Mgmt, which the SUB Mgmt (10) forwards to the device. The device (11) uses the trigger to request the ICRO directly from the RI of the Home COC, which (12) sends the RO to the device.

In the **broadcast mode of operation**, the owner of the device (1) performs an out-of-band purchase transaction with the SUB Mgmt of the Home COC (e.g. through a call centre, or a web portal). During the purchase transaction, the SUB Mgmt of the Home COC (2) requests the availability and pricing information from the Visited SOC, which (3) returns it, for immediate use in the purchase transaction. If the transaction is successful, the SUB Mgmt of the Home COC (4) requests the RI of the Visited SOC to generate and broadcast the BCRO related to the purchase. The RI of the Home COC (5, not shown in the figure) requests the corresponding PEAKs and SEAKs from the DIST Mgmt of the Visited SOC, which (6, not shown in the figure) sends them to the RI of the Home COC. The RI now generates the BCRO, and (7) requests the Visited SOC's RI Proxy to include the BCRO in its RI Service. The RI Proxy (8) broadcasts the BCRO on the Ad-hoc RI Stream for a while, and then continues repeating it in the Scheduled RI Stream during the whole lifetime of the purchased item.

In the **mixed-mode of operation**, the device (1, not shown in figure) first issues a pricing request to the SUB Mgmt of the Home COC, which is now remote to the SOC. The SUB Mgmt of the Home COC (2) requests the availability and pricing information from the Visited SOC, which (3) returns it through the Home COC (4, not shown in the figure) to the device. Only now has the device all the information available that enables it (5) to make a purchase request to the SUB Mgmt of the Home COC. If successful, the flow can continue with step (6) of the interactive mode of operation, or step (4) of the broadcast mode of operation described above.

C.2.2.3 Conclusion to Deployment Option B

The advantage of this deployment option is that the roaming device does not have to perform a RI registration when roaming, but that the ROs will be issued by the RI that is part of the Home COC.

This comes at the cost of:

- the Visited SOC having to make the programme encryption and authentication keys (PEAKs) and service encryption and authentication keys (SEAKs) available to the Home COC;
- the Visited SOC having to operate an RI Proxy to run an RI service that will broadcast BCROs on behalf of the Home RI.

The differences between the local and the roaming scenario are:

- the additional request for pricing information that the Home COC makes from the Visited SOC (the Home COC might cache this information, for subsequent usage with other devices);
- in the local scenario, the Home COC has the choice to either use the SOC's RI Proxy, or to run its own SI Service (this is in principle also possible in the roaming scenario, but not desirable).

The reasons why there is a need for non-local communication between Visited SOC and Home COC in the roaming scenario are the following:

- retrieval of pricing information;
- retrieval of PEAKs and SEAKs.

C.2.3 Conclusion to roaming scenarios

The deployment and roaming scenarios presented in this clause are made up from simple building blocks that can be flexibly combined. Whereas particular business models have been chosen to illustrate how 'roaming' can be implemented, the building blocks are not limiting the deployment options.

C.3 Service Purchase Scenarios

Clause C.3 describes several service purchase scenarios.

C.3.1 Free-to-Air (Unscrambled)

An unscrambled Free-to Air service does not use service protection as defined in the present document. If there is no Key Stream Message signalled for a service, it means that the service is an unscrambled Free-to-Air service.

C.3.2 Free-to-View (Scrambled)

A Free-to-View service uses service protection as defined in the present document. The ROs associated with a Free-to-View service are sent to each registered device free of charge. The RI should schedule the sending of ROs associated with a Free-to-View service such that registered devices can always access Free-to-View service, except perhaps for a short time after device switch-on or device registration.

C.3.3 Subscription-based Services

When a user wants to subscribe to a service, he performs a subscription transaction. The way a subscription transaction and subsequent actions may be carried out depends on which channel is used.

C.3.3.1 Using the Interactivity Channel

When an Interactivity Channel is available, the subscription transaction may be carried out over the Interactivity Channel, e.g. by filling in forms on a Web page. In this subscription transaction, the user somehow indicates this service and his device to the RI and possibly a payment method. After successful completion of this subscription transaction, the RI sends to the device of the user the ROs that govern the access to the service that the user subscribed to.

The delivery of ROs over the Interactivity Channel is specified in clause B.3.3.3.

C.3.3.2 Using the Broadcast Channel

When there is no Interactivity Channel available, the user will have to perform the subscription transaction using a different communication channel, e.g. the telephone. During the subscription transaction, the user communicates the following to the Customer Operation Centre of the Service Provider:

- The service(s) the user wants to subscribe to, possibly in the form of a Human Friendly Purchase ID. A Human Friendly Purchase ID is a Service ID suitable for humans to communicate and it is unique at least within the domain of a particular Service Provider. The Service Provider has to map Human Friendly Service IDs to Service IDs.

- The Human Friendly Device ID, either in the form of the shortform_udn or longform_udn(), as specified in clause B.3.4.3.7.2, or a Domain ID, as specified in clauses B.3.3.4.2 and B.3.4.4.

- A payment method.

After successful completion of this subscription transaction, the RI of the Service Provider sends to the device of the user the ROs that govern the access to the service(s) that the user subscribed to.

The delivery of ROs over the Broadcast Channel is specified in clause B.3.3.4.

C.3.4 One-Off Purchases

In a One-Off purchase, one typically purchases the right to view a certain service not for the full duration of that service, but only for a limited amount of time, e.g. the time that a particular movie is being broadcast.

In the present document, it is an option to divide a service into several programmes. If a service is divided into several programmes, the access control can be on a programme by programme basis, see e.g. clause B.2.4.1.2.

There are 2 variations for a One-Off purchase:

- PPV: the user pays for a complete programme.

- IPPV: the user pays for part of a programme, because he decided to do the One-Off purchase when the programme already had started.

The difference in between the above two can be in the pricing.

The process of a One-Off purchase is equal to the process of a subscription to a service as defined in clause C.3.3, except that the ROs sent to the device of the user contain PEAKs instead of SEAKs (see also clause B.2.4.1.2).

C.3.5 Bundling with a Mobile Subscription

The protection technology defined by the present document does not offer any specific features for bundling mobile and IPDC subscriptions, nor does it put any requirements on whether a subscription is bundled or not with a mobile subscription.

C.3.6 Online and Offline Control of Content

The present document offers a transparent channel for protected content to be downloaded to the device. The possibilities are indicated in clause B.1.

C.3.7 Billing and Charging Types

C.3.7.1 On Line

'On line' indicates that the requested rights to consume a service are purchased with a direct connection between the authorization server and the device (e.g. through interactivity channel).

An example of how an RO could be purchased on line is shown in clause C.3.3.1. Independent of the way an RO is purchased, clauses C.3.3 and C.3.4 show ways how the purchased ROs for a service (clause C.3.3) or part thereof in the form of a programme, see clause C.3.4, can be transmitted to the device.

The consumption of content can also be paid for using tokens stored in the device, see clause C.3.7.5. This clause also shows how tokens can be purchased on line.

C.3.7.2 Off Line

'Off line' indicates that the requested rights to consume a service are purchased without a direct and connected link between the authorization server and the device.

An example of how an RO could be purchased off line is shown in clause C.3.3.2. Independent of the way an RO is purchased, clauses C.3.3 and C.3.4 show ways how the purchased ROs for a service (clause C.3.3) or part thereof (clause C.3.4) can be transmitted to the device.

The consumption of content can also be paid for using tokens stored in the device, see clause C.3.7.5. This clause also shows how tokens can be purchased off line.

C.3.7.3 Pre-Paid

'Pre-paid' indicates that the requested rights to consume a service or programme are charged before this service or programme is delivered and is available for consumption. The examples of consumption purchases in clause C.3.3 and clause C.3.4 are examples of pre-paid consumption if the sending of ROs commences after the user has been charged for these ROs.

Pre-paid can also mean that the content is purchased without any ongoing subscription with the RI. The example of the consumption of content using tokens in clause C.3.7.5 is an example of this type of pre-paid purchasing model if the tokens are delivered to the device after the user has been charged for these tokens.

Note that the device always has to be registered with the RI of the Customer Operation Centre as specified in clause B.2.3, otherwise no ROs of any kind can be sent to the device. However, whether the Customer Operation Centre also wishes to obtain personal information of the user is outside the scope of the present document.

C.3.7.4 Post-Paid

'Post-paid' indicates that the requested rights to consume a service or programme are charged after this service or programme is delivered and is available for consumption. The examples of consumption purchases in clauses C.3.3 and clause C.3.4 are examples of post-paid consumption if the sending of ROs commences already before the user has been charged for these ROs.

Post-paid can also mean that the content is purchased without any ongoing subscription with the RI. The example of the consumption of content using tokens in clause C.3.7.5 is an example of this type of post-paid purchasing model if the tokens are delivered to the device before the user has been charged for these tokens.

Yet another example of post-paid consumption in clause C.3.7.5 is that a device consumes contents using tokens up to a certain limit for the maximum number of tokens and up to a certain date/time limit and that the device has to report the actual consumption in tokens before this date/time limit.

Note that the device always has to be registered with the RI of the Customer Operation Centre as specified in clause B.2.3, otherwise no ROs of any kind can be sent to the device. However, whether the Customer Operation Centre also wishes to obtain personal information of the user is outside the scope of the present document.

C.3.7.5 Tokens

"Token": A consumption unit used for purchasing services. To avoid on-line purchase of service, a token may e.g. be purchased or granted in advance, in which case it would typically be stored in the terminal.

An RI can send tokens to a device as specified in clause B.3.4.5.

The RI can send tokens to a device:

- Free of charge in order e.g. to pre-load a device with tokens in a post-paid purchasing model.
- After a user has ordered tokens using an Interactivity Channel. Billing of the tokens may be before or after delivery of the tokens.
- After a user has ordered tokens using another channel (e.g. by a phone call). Billing of the tokens may be before or after delivery of the tokens.

When sending tokens to a device free of charge, as indicated above, the RI may indicate that the device has to report the actual consumption of tokens to the RI before a certain date/time, see clause B.3.4.5.3. The device can only continue the consumption of tokens after the set date/time limit when it has received a new date/time limit from the RI. Note that RIs may wish not to support this type of token consumption for devices which do not have an Interactivity Channel, because the user cannot be forced to report the actual use of the last period.

Whether or not a device needs to use tokens for the consumption of content and if so how many, is indicated using the metered constraint, as specified in clause B.3.3.4.2.4.8. The metered constraint can be present in an RO (see clause B.3.3.4) as well as in the KSM (see clause B.3.2).

C.3.8 Bundles of Services

Bundles of services can be handled using the tools in the present document in e.g. the following ways:

- When a user subscribes to a bundle of services, he will be sent one RO per service with the SEK and SES of that service.
- When a user subscribes to a bundle of services, he will be sent one RO with the SEKs and SES-es of all services from the bundle.
- Any combination of the above.

C.3.9 Free Preview

Services may offer a 'free preview' to users. Various scenarios for the offering of free previews and possible implementations are informatively described below. However, free previews may be implemented in any way that is compliant with the present document.

C.3.9.1 Free-to-Air Preview

The simplest form of free preview is to allow any user to watch a particular service without restriction for some period of time.

This can be implemented by making a part of the service 'free-to-air' and broadcasting it unencrypted, as described in clause B.2.1.2.1.

C.3.9.2 Free Preview with Rights Objects

Free preview could be implemented in the same way as free-to-view service (see clause C.3.2), where the service encrypted and a key stream is also broadcast. A Rights Object is delivered enabling users to receive the preview service, or perhaps all preview services of a particular Rights Issuer. The key stream and the Rights Object will carry a content ID which allows them to be matched.

C.3.9.3 Free Preview with Access Criteria

The above scheme could be used to restrict access to the preview service according to access criteria in the key stream, as described in clause B.3.2. For example, this could restrict access to registered users above a certain age.

C.3.9.4 Timed Preview

Users can be given the opportunity to receive a service for an arbitrary interval, duration, accumulated duration or with any other restriction that can be described using the OMA DRM 2.0 Rights Expression Language.

This can be implemented by delivering Rights Objects to all the users who are to be offered the preview. These Rights Objects would contain the necessary SEAKs or PEAKs required to the receive the channel, along with the relevant restrictions. Note that the choice of supplying a SEAK or a PEAK and the relevant service or programme CID in the Rights Object will be influenced by how long the preview opportunity will last.

When a device receives a Rights Object that allows 'timed preview', it should indicate to the user that the preview is available.

These Rights Objects may be delivered to any or all registered users to fulfil any requirements a Rights Issuer may have.

C.3.9.5 Preview with Cryptographically-Enforced Period

Rights Issuers could divide a programme up into a series of sub-programmes, each with its own PEAK and CID. Rights Objects for the preview are made available to users containing only the PEAK and CID for the relevant sub-programme, so that the preview will last for the sub-programme length. The ESG will indicate a number of CIDs for the entire programme.

Alternatively, it may be the case that the preview lasts for exactly one programme, or more than one programme, in which case, the mapping of PEAKs and CIDs to programmes can be adjusted accordingly.

C.3.9.6 Advertising Previews via the ESG

In order that users can easily identify that a service offers a preview of some sort, it should be indicated in the ESG.

C.4 Operative Scenarios

C.4.1 Revocation

In certain cases, a need to revoke Device Private Keys may occur. Revoking a Device Private Key means that the certificate certifying the accompanying Device Public Key is included in the Certificate Revocation List (CRL).

When there is an Interactivity Channel, the device will obtain ROs using the ROAP protocol. This ensures that the RI will have to check the CRL any time it sends an RO. For a service requiring a monthly update of the RO, the RI will check the CRL monthly.

When there is no Interactivity Channel with a device, the RI will only use the Device Public Key during registration or re-registration.

The RI should check for the certificates of all devices that are registered on a regular basis whether or not they are on the CRL. If a certificate is on the CRL, the RI

- stops using all secrets (SEAKs, PEAKs, Subscriber Group Keys, etc.) that have been sent to the device to which that certificate belongs; and
- stops sending new ROs to the device to which that certificate belongs; and
- provides all other non-revoked devices that used these secrets with new UGKs, SGKs, ROs, using newly generated secrets.

C.4.2 Expiration

The present document defines two ways of operation, the interactive mode of operation and the broadcast mode of operation, see clauses B.2.5 and B.2.6.

The present document makes use of functionality defined in OMA DRM 2.0 such as e.g. the ROAP protocol and ROs in the interactive mode of operation. In the interactive mode of operation, expiration is defined by OMA DRM 2.0, see [OMA_DRM_DRM].

In the broadcast mode of operation, the present document defines several objects that control expiration. This clause highlights some of these.

C.4.2.1 Expiration of RI Context

During registration, a device receives the RI Context in a device_registration_response() message, see clause B.3.4.3.7.2. This message contains two objects which control expiration of the RI Context.

The first object is the certificate chain of the RI, as indicated in the certificate_chain field of the device_registration_response() message. Certificates in this certificate chain may contain expiration dates.

The second object is a time stamp, as indicated in the time_stamp field of the device_registration_response() message. The time_stamp field may be present in the device_registration_response() message.

Whenever any certificate in the certificate chain expires or when the device_registration_response() message expires based on the field time_stamp, the device stops using the access rights from all ROs that the device received and was able to access using the data (unique group key, subscriber group keys or unique device key) from this device_registration_response() message for making content received after the expiry time available for any purpose.

C.4.2.2 Expiration of a BCRO

The BCRO contains a field called refresh_time, see clause B.3.3.4.2. The date/time in this field indicates to the device that it is advised to try and acquire a newer version of the BCRO, because e.g. one or more end times of constraints in the BCRO are approaching. The refresh_time therefore does not indicate expiration of the BCRO.

C.4.3 Moving from One Group to Another

In order to save bandwidth when sending BCROs to devices, devices may be combined into groups. If several devices in one group require the same BCRO, then when using the group addressing feature of the present document, this BCRO only needs to be sent once and will be received by all addressed devices in the group, see clause B.2.6.2.

In order to optimize the bandwidth for sending BCROs to devices, it may be advantageous to move a device from one group to another.

Moving from one group to another can be done by sending a re-registration trigger to the device, see clause B.3.4.3.5. As part of the re-registration process, the RI provides the device with the credentials of the new group, using the device_registration_response() message, see clause B.3.4.3.7.2.

C.5 Use of the Interoperability Point

The interoperability point, or short i-point, allows multiple DRM implementations to be used simultaneously in an IPDC infrastructure. Rights to one and the same service can be sold via multiple Rights Issuers using multiple rights management systems and multiple DRM implementations. A Rights Issuer in this context does not imply an OMA DRM 2.0 Rights Issuer, it merely indicates an entity issuing Rights Objects independent of the DRM system used.

Given a service *A* from service provider *SP1*. *SP1* will encrypt the content and broadcast it together with a key stream. The base content ID of the service and the socID is signalled as part of the ESG and the service extension ID is signalled as part of the KSM.

The ESG will contain purchase information for each RI selling rights to a particular service or programme as shown in figure C.7.

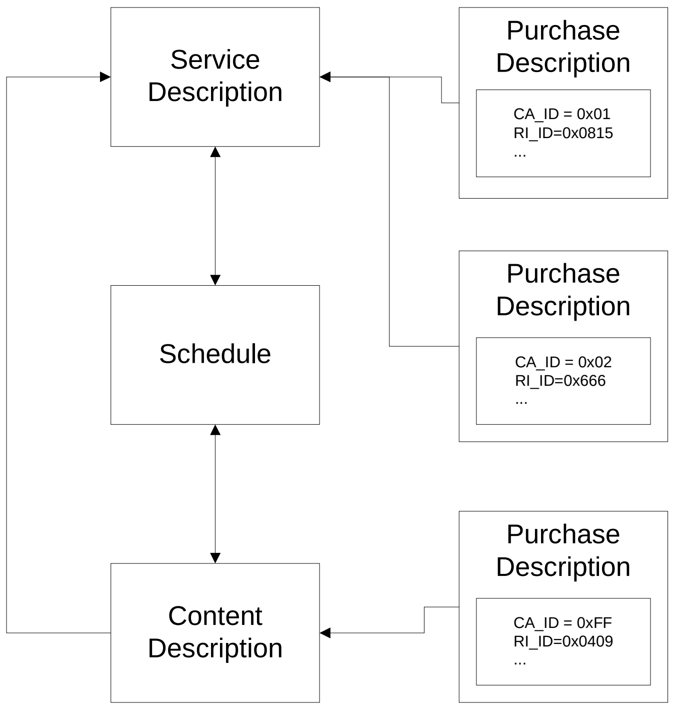


Figure C.7: Use of multiple purchase descriptions in the ESG

The CA_ID is a unique ID identifying the DRM system used above the i-point. The values 0x00 and 0x01 are reserved. The value 0x01 indicates that the DRM system used above the i-point is the DRM system specified by the present document. The assignment of values to specific DRM system is outside of the scope of the present document.

The combination of the socID, content_id, content_type and the service extension ID0 uniquely identifies (see clause B.10) the service and the keys used to encrypt the TEK. *SP1* provides details about the service (socID, content_id, content_type) together with key information (SEK/SAK or PEK/PAK, service_extension_ID) to all interested RIs. The RIs create either compliant ROs as defined in the present document or proprietary "ROs" intended

for other DRM implementations. These ROs get send to interested devices, either by means specified in the present document or by any other means in case of proprietary rights management systems. The present document does not specify proprietary ROs but in order to use the interoperability point, it needs to be possible to associate proprietary ROs to the unique service_CID and they have to carry the key material (SEK and SAK or PEK and PAK).

A device, which supports multiple DRM implementations is assumed to check with each implementation whether it has any ROs for a specific service_CID. A possible algorithm for checking multiple DRM implementations in a device is shown in figure C.8.

When a user selects a service or programme the device acquires the socID, content_type (programme or service) and the service_base_ID from the ESG, together with the information on how to receive the KSM. Once the device has received the first KSM it can create the CID/BCI. Given the CID/BCI the device can query all DRM systems available in that device whether any of the DRM systems has a RO for that particular CID/BCI. The default DRM system to check should be the system specified in the present document. As an optimization the device might only check with DRM systems listed in the purchase information of the ESG.

If none of the DRM systems in that device has a RO for that service/programme the access to that service/programme fails and the user could be notified that there are no entitlements to view that service/programme.

If at least one DRM system has a RO for that particular service/programme the DRM system is instructed to decrypt the KSM, check for its integrity and deliver the resulting TEK to the IPsec, SRTP or ISMACryp implementation. Access permissions listed in the KSM for that particular RI_ID SHALL be adhered to by the DRM system.

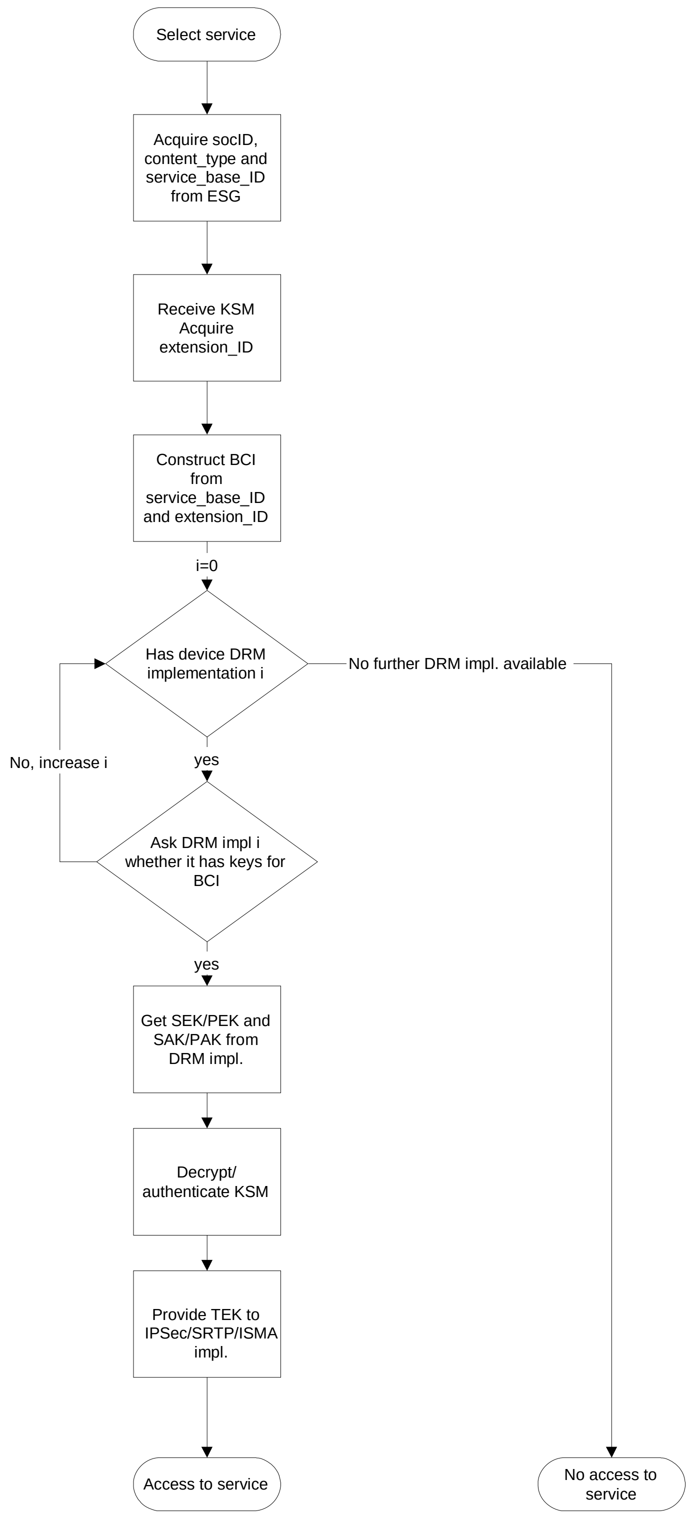

Select service
Acquire socID, content_type and service_base_ID from ESG
Receive KSM Acquire extension_ID
Construct BCI from service_base_ID and extension_ID
i=0
Has device DRM implementation i
No further DRM impl. available
No, increase i
yes
Ask DRM impl i whether it has keys for BCI
yes
Get SEK/PEK and SAK/PAK from DRM impl.
Decrypt/ authenticate KSM
Provide TEK to IPSec/SRTP/ISMA impl.
Access to service
No access to service

Figure C.8: Process for service decryption when multiple DRM implementations are available in device

C.6 Security Considerations

This clause explains the assumptions behind the security of the system. The system can be deployed either in interactive mode or broadcast mode, and can be used as the basis for service or content protection.

The host receiving the key stream messages, BCROs and IPsec, SRTP or ISMACryp traffic must take care to protect itself from maliciously constructed key stream messages and rights objects. This is protection that must be afforded to the host and the user of the host. To protect the host the following conditions shall have to be true:

- Key stream messages shall not instantiate outbound IPsec security associations.
- An IPsec security association, SRTP cryptographic context or ISMACrypKey instantiated based on a key stream message shall not interfere with any negotiated or user-configured IPsec security association.
- The amount of resources devoted to IPDC IPsec security associations, SRTP cryptographic contexts and ISMACrypKeys shall be limited so that a denial-of-service attack is not possible.
- The number of key stream messages under processing shall be limited so that a denial-of-service attack is not possible against the host. This is essentially critical as the key stream message authentication key is stored in multiple systems and they may not be trustworthy.
- The amount of Rights Objects under processing shall be limited so that a denial-of-service attack is not possible against the host. This is especially critical as the authentication of Rights Objects may not be reliable (a symmetric key is used in the case of BCROs) and-or they may not be protected.
- For each IPsec security association, SRTP cryptographic context and ISMACrypKey, there must be one key stream that is authorized to modify the security association, cryptographic context or key. Modification attempts by unauthorized key streams shall be silently ignored.

To assure a level of protection for hosts against other malicious hosts in the system is based on the security of the authentication keys. This security is ultimately rooted in the tamper-resistance of the hosts holding the authentication keys. The following conditions shall have to be true:

- The BCRO authentication keys shall not be readily available in a receiving host.
- The key stream message authentication keys shall not be readily available in a receiving host.
- The IPsec security association authentication keys shall not be readily available in a receiving host.
- The SRTP cryptographic context authentication keys shall not be readily available in a receiving host.

Note that the authentication of key stream messages and BCRO objects is not reliable as the system does not guarantee status information about key validity, especially in broadcast mode. The message authentication codes of key stream messages and BCROs provide security in direct proportion to the weakest tamper-resistant hardware that has access to the used message authentication keys.

The protection that must be afforded for the decryption keys and the broadcast content is required for the security of the service and content protection. This protection will ultimately be rooted in tamper-resistance of the receiving hosts. If the system is deployed for service protection then the assets that must be protected from unauthorized access are the long-term secret keys on the device and short-term decryption keys for the broadcast data.

The keys are divided into two domains. Long-term and short-term. Long-term secrets are essentially secrets whose lifetime spans that of several rights objects. Short-term secrets are secrets that are changed frequently over time, at least changing per rights object, if necessary.

The long-term domain of keys is:

- SGKs, BDKs, UDK, TDK and UGK.
- Intermediate keys derived when deriving an IEK from a set of SGKs.
- OMA DRM 2.0 decryption and signing keys.
- OMA DRM 2.0 domain keys.
- RIAK.

The short-term domain contains all the other keys in the system.

The trustworthiness of a receiving host in terms of service protection can be indicated as a tuple of the following parameters:

1) Level of protection afforded long-term decryption secrets and asymmetric keys.

2) Level of protection afforded short-term secrets.

3) Level of protection afforded long-term symmetric authentication keys.

Parameter 1 defines the capability of a host to compromise content and decryption not intended for it. Parameter 2 defines the capability of a host to compromise content intended for it. Parameter 3 defines the capability of a host to pose as a legitimate rights issuer or broadcaster to other hosts in the system.

The present document does not as such define robustness requirements for implementations, but the recommendation is that long-term secrets should be afforded hardware-based protection, such that their storage and handling is performed using tamper-resistant hardware. It is probably a reasonable to rely on software protection (such as OS security) for short-term secrets, especially if the integrity of the handling software is assured.

If content protection is desired then the system must be hardened further. In this only authorized applications that heed post-acquisition and usage rules have access to decrypted keys and decrypted content. Note that SRTP does NOT authenticate any enveloping headers such as UDP ports. It is recommended that authentication be used in cases where content protection is used to prevent a malicious party from modifying the cipher text. This could result in redirecting decrypted traffic to another application. It is trivial to redirect traffic to another application if SRTP is used as SRTP does not authenticate the enveloping header. In this case it must be ensured that these unauthorized applications do not have access to the decryption keys. If IPsec is used, then it must be ensured that an unauthorized application does not receive the decrypted traffic.

If content protection is required then also the authentication keys must be secured at least as strongly as any decryption keys as they are used to prove authenticity and integrity of usage rules.

The trustworthiness of a receiving host for content protection can be measured in terms of the service protection trustworthiness parameters and the ability to limit decrypted content and keys to applications that heed the usage rules and security offered stored content.

The content if cached/stored on the local device may be stored in a multitude of ways, and it is beyond the scope of the present document to define how that is to be done. It is recommended that if content protection is desired the content be stored in such a manner that unauthorized applications cannot decrypt stored protected content or change the usage rules pertaining to stored protected content.

C.6.1 Threat Mapping

This clause contains a simplified threat mapping of the system to highlight why the different cryptographic assets must be protected. This mapping is given in terms of assets, their function, the consequence of unauthorized use and possible mitigation techniques. The full transitive chain of consequences of the compromise of one asset leading to the compromise of another asset is not explicitly given here, but the cause-consequence relationships should be clear enough that the reader can work these out. This clause does not consider how any decrypted content must be handled within the system, as that is a function of the robustness of the system, and outside the scope of the present document.

This clause intends to provide input to the design of implementation, especially how to structure the manipulation of different secrets inside a possible device. This clause does not aim to be a complete, normative or definitive treatment.

Asset	**Device Private Key**
Function	The device private key is used for registration and decryption of the SGKs, BDKs, UGK and UDK addressed to this device. The key is part of a key pair, and the private key is used for decryption, with the device public key being used for encryption.
Consequence of unauthorized use	All keyset blocks (SGKs, BDKs, UGK, RIAK and UDK) addressed to the device owning the key can be decrypted. These can be used to decrypt the fields in all BCROs addressed to the device encrypted using keys in these keyset blocks.
Mitigation	The device private key must be stored in a tamper-resistant hardware module. Only a robust implementation of the non-interactive mode registration layer must be allowed use of it. In theory, if there is a mechanism for detecting compromised device private keys, the corresponding public keys could be revoked using conventional techniques such as revocation lists.

Asset	**OMA DRM 2.0 Device Private Keys**
Function	These keys are used to authenticate OMA DRM 2.0 client-side messages and decrypt rights objects that are addressed to this device.
Consequence of unauthorized use	The OMA DRM 2.0 Rights Objects addressed to this device can be decrypted. A malicious party could create unauthorized signatures and pretend to the identity of the OMA DRM 2.0 implementation in the device that it could use these keys.
Mitigation	The private keys must be stored in a tamper-resistant fashion and the keys must only be accessible to a robust OMA DRM 2.0 implementation.

Asset	**Unique Group Key**
Function	The Unique Group Key (UGK) is a common symmetric key for all members in a group of devices in non-interactive mode. The group and the key are determined at the registration layer. This key is used to derive the IEK for when a BCRO is addressed to all the members in a non-interactive mode subscriber group.
Consequence of unauthorized use.	The IEK for all BCROs addressed to the corresponding group could be computed.
Mitigation	The Unique Group Key (UGK) should be stored in tamper resistant hardware and only be allowed to be used by a robust implementation of the non-interactive mode rights layer. Further, the HMAC_SHA1() hash function and the SALT input from the BCRO means that the IEK derived from the UGK does not reveal information about the UGK. The rights layer should be allowed access only to IEKs.

Asset	**Subscriber Group Keys**
Function	The Subscriber Group keys (SGKs) are used to derive the IEKs for the subscriber groups in non-interactive mode. They are managed by the registration layer.
Consequence of unauthorized use	The IEKs can be computed for all the BCROs addressed to subscriber subgroups where the recipient has subscribed to. If the SGKs of two different devices in the same subscribergroup can be obtained, then all BCROs addressed to all subscribers in this subscribergroup can be decrypted.
Mitigation	The Subscriber Ggroup Kkeys (SGKs) must be stored in tamper-resistant hardware and only be allowed to be used by a robust implementation of the non-interactive mode registration layer. Further, the HMAC_SHA1() hash function and the SALT input from the BCRO means that the IEK derived from the SGKs does not reveal information about the SGK. The rights layer should be allowed access only to IEKs.

Asset	**Unique Device Key**
Function	The unique device key (UDK) is used to form the IEK used to decrypt BCROs addressed to a single device in non-interactive mode. This key is managed by the registration layer.
Consequence of unauthorized use	BCROs addressed to this device would be decrypted.
Mitigation	The unique device key (UDK) must be stored in tamper-resistant hardware and only be allowed to be used by a robust implementation of the non-interactive mode registration layer. Further, the HMAC_SHA1() hash function and the SALT input from the BCRO means that the IEK derived from the UDK does not reveal information about the UDK. The rights layer should be allowed access only to IEKs.

Asset	**Token Delivery Key**
Function	The Token Delivery Key (TDK) is used to protect the token quantity and the (consumed token) report authentication key in the token delivery response message. This key is managed by the registration layer.
Consequence of unauthorized use	Devices can consume tokens meant and paid for by others. Token consumption reports can be faked.
Mitigation	The Token Delivery Key (TDK) shall be stored in tamper-resistant hardware and only be allowed to be used by a robust implementation of the non-interactive mode registration layer.
Asset	**Broadcast Domain Key**
Function	The broadcast domain keys (BDKs) are used to form the IEK used to decrypt BCROs addressed to a devices that are registered in a broadcast domain in non-interactive mode. The domain and the key(s) are managed by the registration layer.
Consequence of unauthorized use	BCROs addressed the members of a particular broadcast domain would be decrypted.
Mitigation	The Broadcast Domain Keys (BDKs) must be stored in tamper-resistant hardware and only be allowed to be used by a robust implementation of the non-interactive mode registration layer. Further, the HMAC_SHA1() hash function and the SALT input from the BCRO means that the IEK derived from the BDK does not reveal information about the BDK. The rights layer should be allowed access only to IEKs.

Asset	**OMA DRM 2.0 Domain Keys**
Function	These keys are used to decrypt rights objects that are addressed to members of an OMA DRM 2.0 domain.
Consequence of unauthorized use	The OMA DRM 2.0 Rights Objects addressed to a domain can be decrypted.
Mitigation	The OMA DRM 2.0 domain keys should only be accessible and usable by robust implementations of OMA DRM 2.0.

Asset	**RIAK**
Function	The Rights Issuer Authentication Key is used in non-interactive mode to derive the BA, which is used to authenticate BCRO objects. This is a symmetric key. This key is provided as part of the registration data.
Consequence of unauthorized use	Unauthorized use of the RIAK allows a member of a subscriber group to pose as a rights issuer to other members of the same group. This could be used to create denial-of-service attacks or to allow an attacker to gain access to attack vectors that would otherwise have been protected.
Mitigation	The RIAK should be stored in tamper-resistant hardware and it should only be usable and accessible by a robust implementation of the registration layer. Preferably the registration layer would only indicate whether a BCRO is authentic or not to the rights layer. If that is at all possible, then the registration layer should provide only a BAK to the rights layer. Because the RIAK is a symmetric key, the rights layer of a device should not be able to compute MACs, only verify them.

Asset	**SEK**
Function	The service encryption key is the key provisioned in rights objects to devices. This key is used to encrypt TEKs and in-case a pay-per-view model is followed, PEKs.
Consequence of unauthorized use	Unauthorized use of the SEK allows one to decrypt TEKs or PEKs one is not entitled to.
Mitigation	The SEKs should not be revealed by the rights layer. Only decrypted TEKs should be revealed by the rights layer. The rights layer implementation should be robust.

Asset	**PEK**
Function	The program encryption key is the key provisioned in rights objects to devices. This key is used to encrypt TEKs in-case pay-per-view model is followed. This key may be also provisioned in a Key Stream Message, in which it would be encrypted using a SEK.
Consequence of unauthorized use	Unauthorized use of the PEK allows one to decrypt TEKs one is not entitled to.
Mitigation	The PEKs should not be revealed by the rights layer. Only decrypted TEKs should be revealed by the rights layer. The rights layer implementation should be robust.

Asset	**TEK**
Function	The traffic encryption keys are used as the master key for the traffic layer (IPsec, SRTP, or ISMACryp). They are provisioned in Key Stream Messages and are encrypted using a SEK or a PEK.
Consequence of unauthorized use	The IPsec, SRTP or ISMACryp traffic can be decrypted by an authorized party.
Mitigation	The TEK should be changed relatively frequently, on the order of once per a couple of seconds, if possible. The implementation handling the TEKs (the traffic layer) and the key stream layer, should be robust and not reveal the TEK to unauthorized parties nor allow its use for anything than incoming IPsec, SRTP or ISMACryp traffic, as appropriate. Backup/restore or import/export of TEKs by the traffic layer should not be allowed.

Asset	**BAK**
Function	The BAK is a key used to authenticate BCROs. It is derived from the RIAK.
Consequence of unauthorized use	Unauthorized use of the RIAK allows a member of a subscriber group to pose as a rights issuer to other members of the same group. This could be used to create denial-of-service attacks or to allow access to attack vectors that would otherwise have been protected.
Mitigation	The BAK should not be stored, but should be derived as needed by the registration layer. Preferably the registration layer would only indicate whether a BCRO is authentic or not to the rights layer. If that is at all possible, then the registration layer should provide only a BAK to the rights layer. The RIAK used to create the BAK must never be revealed. Because the RIAK is a symmetric key, the rights layer of a device should not be able to compute MACs, only verify them.

Asset	**SAK**
Function	The Service Authentication Key is derived from the encrypted service authentication seed inside a BCRO. This key is used to authenticate key stream messages.
Consequence of unauthorized use	Unauthorized use of the SAK allows a malicious party to create forged key stream messages. Possibly causing denial of service attacks, or allowing access to attack vectors that would not have been available otherwise.
Mitigation	The rights layer implementation should be robust, and it should not reveal the SAK to the key stream layer (or any other party). It should only provide an 'authentic / not authentic' response for key stream messages.

Asset	**PAK**
Function	The Program Authentication Key is derived from the encrypted program authentication seed inside a BCRO. This key is used to authenticate key stream messages.
Consequence of unauthorized use	Unauthorized use of the PAK allows a malicious party to create forged key stream messages. Possibly causing denial of service attacks, or allowing access to attack vectors that would not have been available otherwise.
Mitigation	The rights layer implementation should be robust, and it should not reveal the PAK to the key stream layer (or any other party). It should only provide an 'authentic / not authentic' response for key stream messages.

Asset	**IEK**
Function	The Inferred Encryption Key. This key is used to encrypt and decrypt SEKs or PEKs in BCROs. This key is derived either from the SGKs, the UGK or the UGK and the BCI field from the BCRO. An IEK is specific to a single BCI-value and broadcast sub-group in the non-interactive mode.
Consequence of unauthorized use	Unauthorized use of an IEK allows unauthorized use of a BCRO. An IEK is additionally required to compute a BAK, but is not alone sufficient for that. However, since both the RIAK and SGKs, UGK and UDK reside in the registration layer, one must assume that unauthorized use of an IEK could also allow unauthorized use of a RIAK.
Mitigation	The IEK should be provided by the registration layer to the rights layer, and neither party should reveal it to unauthorized elements in the system. The IEK can also be changed relatively frequently if necessary by changing the BCI in BCROs. This allows a method to recover from the compromise of a single IEK.

Asset	**IPsec SA**
Function	The IPsec SA defines the decryption and encryption of traffic at the traffic layer if IPsec is used. The IPsec SA is instantiated by the Key Stream Layer.
Consequence of unauthorized use	Encrypted plaintext could be decrypted without authorization. A device could send forged packets to other devices. This could lead to a denial-of-service attack or access to attack vectors that would otherwise have been unavailable.
Mitigation	An IPsec implementation should not export or reveal the cryptographic keys related to IP datacast to unauthorized parties. Additionally the IPsec implementation should not allow the redirection of data decrypted using IP datacast SAs to non-robust or compliant applications. Especially if content protection is desired.

Asset	**SRTP Crypto Context**
Function	The SRTP crypto context defines the decryption and encryption of traffic at the traffic layer if SRTP is used. The SRTP Crypto Context is instantiated by the Key Stream Layer.
Consequence of unauthorized use	Encrypted plaintext could be decrypted without authorization. A device could send forged packets to other devices. This could lead to a denial-of-service attack or access to attack vectors that would otherwise have been unavailable.
Mitigation	An SRTP implementation should not export or reveal the cryptographic keys related to IP datacast to unauthorized parties. Additionally the SRTP implementation should not allow the redirection of data decrypted for IP datacast to non-robust or compliant applications. Especially if content protection is desired.

Asset	**ISMACrypKey**
Function	The ISMACrypKey defines the decryption and encryption of traffic at the traffic layer if ISMACryp is used. The ISMACrypKey is instantiated by the Key Stream Layer.
Consequence of unauthorized use	Encrypted plaintext could be decrypted without authorization. A device could send forged packets to other devices. This could lead to a denial-of-service attack or access to attack vectors that would otherwise have been unavailable.
Mitigation	An ISMACryp implementation should not export or reveal the cryptographic keys related to IP datacast to unauthorized parties. Additionally the ISMACryp implementation should not allow the redirection of data decrypted for IP datacast to non-robust or compliant applications. Especially if content protection is desired.

Annex D (informative): Bibliography

IETF RFC 3647: "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework".

IETF RFC 3174: "US Secure Hash Algorithm 1 (SHA1)".

ISO/IEC 13818-1: "Information technology -- Generic coding of moving pictures and associated audio information: Systems".

ETSI TS 131 102: "Universal Mobile Telecommunications System (UMTS); Characteristics of the Universal Subscriber Identity Module (USIM) application (3GPP TS 31.102)".

A.J. Menezes, P.C. van Oorschot, S.A. Vanstone: "Handbook of applied cryptography", CRC Press.

ISO/IEC 14496-15: "Information technology - Coding of audio-visual objects - Part 15: Advanced Video Coding (AVC) file format"

IETF RFC 3550: "RTP: A Transport Protocol for Real-Time Applications".

ISMA 1.0: "Internet Streaming Media Alliance Implementation Specification".

IETF RFC 3984: "RTP payload Format for H.264 Video".

IETF RFC 2327: "SDP: Session Description Protocol".

ETSI TR 101 154 (V1.7.1): "Digital Video Broadcasting (DVB); Implementation guidelines for the use of MPEG-2 Systems, Video and Audio in satellite, cable and terrestrial broadcasting applications".

Java 2 Platform, Micro Edition: "Foundation Profile Version 1.0.1". http://java.sun.com/products/foundation/index.jsp

History

Document history		
V1.1.1	November 2007	Publication