OMA DRM v2.0 Extensions for Broadcast Support

Candidate Version 1.0 – 26 Feb 2008

Open Mobile Alliance

OMA-TS-DRM_XBS-V1_0-20080226-C

Contents

Figures

Tables

1. Scope

Open Mobile Alliance (OMA) specifications are the result of continuous work to define industry-wide interoperable mechanisms for developing applications and services that are deployed over wireless communication networks.

The scope of OMA "Digital Rights Management" [DRM-v2] is to enable the consumption of digital content in a controlled manner. The content is consumed on authenticated devices per the usage rights expressed by the content owners. The OMA DRM work addresses the various technical aspects of this system by providing appropriate specifications for content formats, protocols, and the rights expression language.

The scope for this specification is the application of the OMA "Digital Rights Management" specifications in a typical broadcast environment in which devices might only be capable of receiving information broadcast over a shared medium. It refers to the general OMA "Digital Rights Management" [DRM-v2] documents as its foundation. The causes defined in this document take precedence over those specified by the foundation documents, thus creating a broadcast interpretation of the OMA Digital Rights Management standard.

This specification is used by the OMA BCAST enabler in [BCAST10-ServContProt].

2. References

2.1 Normative References

[AES_WRAP] "AES Key Wrap Specification", National Institute of Standards and Technology (NIST), 16 November 2001

[BCAST10-ServContProt] "Service and Content Protection for Mobile Broadcast Services", Open Mobile Alliance™, OMA-TS-BCAST_SvcCntProtection-V1_0,
URL: http://www.openmobilealliance.org/

[BCAST10-SG] "Service-Guide for Mobile Broadcast Services", Open Mobile Alliance™, OMA-TS-BCAST_SerivceGuide-v1_0,
URL: http://www.openmobilealliance.org/

[DRM20-Broadcast-Extensions-OMADD-XSD] "Mobile Broadcast Services - XML schema for OMA DRM 2.0 Extensions for BCAST (XBS) - Data Dictionary", Open Mobile Alliance™, OMA-SUP-XSD_drm_dd_xbs-V2_0,
URL: http://www.openmobilealliance.org

[DRM20-Broadcast-Extensions-RISD-XSD] "Mobile Broadcast Services - XML schema for OMA DRM 2.0 Extensions for BCAST (XBS) - Rights Issuer Service Data", Open Mobile Alliance™, OMA-SUP-XSD_drm_risd_V1_0,
URL: http://www.openmobilealliance.org

[DRM20-Broadcast-Extensions-ROAP-XSD] "Mobile Broadcast Services - XML schema for OMA DRM 2.0 Extensions for BCAST (XBS) - ROAP", Open Mobile Alliance™, OMA-SUP-XSD_drm_roap_extensionhooks-V2_0,
URL: http://www.openmobilealliance.org

[DRMCF-v2] "DRM Content Format", Open Mobile Alliance™, OMA-DRM-DCF-V2_0,
URL: http://www.openmobilealliance.org/

[DRMREL-v2] "DRM Rights Expression Language", Open Mobile Alliance™, OMA-DRM-REL-V2_0,
URL: http://www.openmobilealliance.org/

[DRM-v2] "Digital Rights Management", Open Mobile Alliance™, OMA-DRM-DRM-V2_0,
URL: http://www.openmobilealliance.org/

[EUROCRYPT] EN 50094:1992 - CLC/TC 206 Access control system for the MAC/packet family: EUROCRYPT, 1992

[FIPS 197] FIPS 197, " Advanced Encryption Standard - AES", National Institute of Standards and Technology (NIST), November 26, 2001

[FIPS 198] FIPS 198, "The Keyed-Hash Message Authentication Code (HMAC)", Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8900, March 6, 2002

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1_0-20031030-A,
URL: http://www.openmobilealliance.org/

[ISO14496-12:2005/Amd1] "Information technology – Coding of audio-visual objects – Part 12: ISO Base Media File Format", International Organisation for Standardisation, ISO/IEC 14496-12, Amendment 1 to Second Edition (2005), April 2007

[ISO14496-12] "Information technology – Coding of audio-visual objects – Part 12: ISO Base Media File Format", International Organisation for Standardisation, ISO/IEC 14496-12, Second Edition, April 2005

[OCSP-MP] OMA Online Certificate Status Protocol (profile of [OCSP] V1.0, Open Mobile Alliance™, OMA-OSCP-V1_0-20040127,
URL: http://www.openmobilealliance.org/

[OSCP] RFC 2560, "Internet X.509 Public Key Infrastructre: Online Certificate Status Protocol – OCSP", M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, June 1999,
URL: http://www.ietf.org/rfc/rfc2560.txt

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

[RFC 1305] RFC 1305, "Network Time Protocol (Version 3) Specification, Implementation and Analysis", David L. Mills, March 1992,
URL: http://www.ietf.org/rfc/rfc1305.txt

[RFC 1738] RFC 1738, Uniform Resource Locators (URL), T. Berners-Lee, L. Masinter, M. McCahill, December 1994,
URL: http://www.ietf.org/rfc/rfc1738.txt

[RFC 2045] RFC 2045, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed, N. Borenstein, November 1996,
URL: http://www.ietf.org/rfc/rfc2045.txt

[RFC 2104] RFC 2104, "HMAC: Keyed-Hashing for Message Authentication", H. Krawczyk, M. Bellare, R. Canetti. February 1997,
URL: http://www.ietf.org/rfc/rfc2104.txt

[RFC 2406] RFC 2406, "IP Encapsulating Security Payload (ESP)", S. Kent, R. Atkinson. November 1998,
URL: http://www.ietf.org/rfc/rfc2406.txt

[RFC 2560] RFC 2560, "X.509 Internet Public Key Infrastructure. Online Certificate Status Protocol – OCSP", M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, June 1999,
URL: http://www.ietf.org/rfc/rfc2560.txt

[RFC 3174] RFC 3174, "US Secure Hash Algorithm 1 (SHA1)", D. Eastlake 3rd, P. Jones. September 2001,
URL: http://www.ietf.org/rfc/rfc3174.txt

[RFC 3280] RFC 3280, "Internet X.509 Public Key Infrastructure. Certificate and Certificate Revocation List (CRL) Profile", R. Housley, W. Polk, W. Ford, D. Solo, April 2002,
URL: http://www.ietf.org/rfc/rfc3280.txt

[RFC 3566] RFC 3566, "The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec", S. Frankel, H. Herbert, September 2003,
URL: http://www.ietf.org/rfc/rfc3566.txt

[RFC 3629] RFC 3629, "UTF-8, a transformation format of ISO 10646", F. Yergeau, November 2003,
URL: http://www.ietf.org/rfc/rfc3629.txt

[RFC 3664] RFC 3664, "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)", P. Hoffman, January 2004,
URL: http://www.ietf.org/rfc/rfc3664.txt

[RFC 768] RFC 768, "User Datagram Protocol", J. Postel, August 28, 1980,
URL: http://www.ietf.org/rfc/rfc768.txt

[RFC2119] RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997,
URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC2234] RFC 2234, "Augmented BNF for Syntax Specifications: ABNF", D. Crocker, Ed., P. Overell, November 1997,
URL: http://www.ietf.org/rfc/rfc2234.txt

[RFC4234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. October 2005,
URL:http://www.ietf.org/rfc/rfc4234.txt

[SCHNEIER] "Applied Cryptography, Second Edition: protocols, algorithms, and source code in C", Bruce Schneier

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures,
URL:http://www.openmobilealliance.org/

[VERHOEF_1969] "Error detecting decimal codes", J. Verhoef, Mathematical Centre Tract 29, The Mathematical Centre, Amsterdam, 1969.

[XC14N] "Exclusive XML Canonicalization: Version 1.0", John Boyer, Donald E. Eastlake 3rd and Joseph Reagle, W3C Recommendation July 18, 2002.
URL: http://www.w3.org/TR/xml-exc-c14n/

[XMLEnc] "XML Enryption Syntax and Processing", D. Eastlake, J. Reagle, Takeshi Imamura, Blair Dillaway, Ed Simon, W3C Recommendation, December 10, 2002,
URL: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[XMLSchema] "XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney and Noah Mendelsohn. W3C Recommendation, May 2, 2001.
URL: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

2.2 Informative References

[DRMARCH-v2] "OMA DRM Architecture Overview", Open Mobile Alliance™, OMA-DRM-ARCH-V2-0, URL: http://www.openmobilealliance.org/

[ETSI 102 474] ETSI TS 102 474 v1.1.1 (2006-04), "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Service Purchase and Protection", URL: http://portal.etsi.org/

[FIAT_NAOR] "Broadcast Encryption", A. Fiat, M. Naor, Advances in Cryptology - CRYPTO '93, Lecture Notes in Computer Science, Vol. 773, 1994, pp. 480 – 491

[ISO/IEC 13818-1] ISO/IEC 13818-1, "Information technology - Generic coding of moving pictures and associated audio information - part1: Systems"

[NAOR02] "Revocation and Tracing Schemes for Stateless Recievers", D. Naor, M. Naor, J. Lotspiech, June 2002

[NIST 800-38A] NIST 800-38A: "Recommendation for Block Cipher Modes of Operation; Methods and Techniques", 2001

[OFT] "Key establishment in large dynamic groups using one-way function trees", A.T. Sherman, D.A. McGrew, IEEE Transactions on Software Engineering, Volume 29, Issue 5, May 2003, pp. 444 – 458

[OMADICT] "Dictionary for OMA Specifications", Version x.y, Open Mobile Alliance™, OMA-ORG-Dictionary-Vx_y, URL:http://www.openmobilealliance.org/

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

This is an informative document, which is not intended to provide testable requirements to implementations.

3.2 Definitions

Adapted PDCF	The PDCF file format from [DRMCF-v2], including adaptations as specified in this document.
Broadcast Device	A device that provides functionality for receiving unprotected or protected broadcast services over the broadcast channel, without using does not support an interactive communication channel. and cannot communicate with other entities except using the broadcast channel. Note that a Broadcast Device can still have an implicit return channel: it may present information, triggers and dialogs to the user who may “"implement”" the interaction channel in various ways (e.g. telephone, web portal, service desk). Note further that a device MAY either be a Broadcast Device, an Interactive Device or a Mixed-Mode Device.
Broadcast Rights Object	This is a Rights Object used by DRM profile of the Service and Content Protection. BCRO is delivered over broadcast channel. Encoding of the BCRO is specified in Section 8 of this specification [XBS DRM extensions-v1.0].
Data Carousel	System used in broadcast environments for transmitting a set of data in a repeating pattern, allowing data to be pushed from a broadcaster to multiple receivers. This mechanism allows a device to reconstitute the transmitted set of data tuning anytime tot the channel during at least the carousel period.
Generalised Rights Object	This term is used in this document as a more generic term whenever an RO or a BCRO is meant.
Inferred Encryption Key	Refers to the key used for encrypting or decrypting the CEK/SEK/PEK. The Inferred Encryption Key is derived from the UGK, the DEK, the UDK or the BDK. The Inferred Encryption Key is only used for BCROs.
Interactive Device	A device that supports provides functionality for receiving unprotected or protected broadcast services over the broadcast channel and interaction channel, using an interactive communication channel and that can communicate with other entities without using the broadcast channel for the communication. For example, an Interactive Device can execute interactive protocols, like the DRM 2.0 ROAP protocol or HTTP towards a Rights Issuer. Note that a device MAY either be a Broadcast Device, an Interactive Device or a Mixed-Mode Device.
Mixed-mode Device	A Device that is both a Broadcast Device and an Interactive Device, i.e. a device that supports an interactive communication channel and also provides functionality for receiving unprotected or protected broadcast services over the broadcast channel, without using an interactive communication channel.
Mixed-mode-operation	The operation of a Rights Issuer that can handle both Interactive Devices and Broadcast Devices.
Mobile Broadcast Service Provider	The Mobile Broadcast Service Provider provides Broadcast Services to the End-User. The Mobile Broadcast Service Provider may use the facilities of a Mobile Broadcast Network Operator to distribute the Mobile Broadcast Services to the End-User.
Rights Issuer Service	Service that carries Broadcast Rights Objects, registration data and other messages from a Rights Issuer over a Broadcast Channel.
Rights Object	This is a Rights Object used by DRM profile of the Service and Content Protection. RO is delivered over interaction channel. Encoding of the RO is specified in [DRMDRM-v2.0].

3.3 Abbreviations

AES	Advanced Encryption Standard
ARC	Action Request Code
BAK	BCRO Authentication Key
BCD	Binary Coded Decimal
BCI	Binary Content Identifier
BCRO	Broadcast Rights Object
BDK	Broadcast Domain Key
BSD/A	Broadcast Service Distribution/Adaptation Center
BSM	BCAST Subscription Management
CA	Certification Authority
CIEK	Content Item Encryption Key
CRL	Certificate Revocation List
DEK	Deduced Encryption Key
DK	Device Key
DRD	Device Registration Data
DRM	Digital Rights Management
DVB	Digital Video Broadcasting
ECT	Efficient Coding Table
ESP	Encapsulating Security Payload
FSGK	Flexible Subscriber Group Key
GRO	Generalised Rights Object
HMAC	Hashed Message Authentication Code
ID	Identification
IEK	Inferred Encryption Key
IPsec	IP Security
IV	Initialization Vector
LBDF	Longform Broadcast Domain Filter (a.k.a. longform_domain_id)
MAC	Message Authentication Code
MJD	Modified Julian Date
MTU	Maximum Transmission Unit
NDD	Notification of Detailed Data
NK	Node Key
NSD	Notification of Short Data
OBEX	Object Exchange
OCSP	Online Certificate Status Protocol
OFT	One-way Function Tree

OMA	Open Mobile Alliance
OOB	Out Of Band
PAK	Program Authentication Key
PAS	Program Authentication Seed
PDR	Push Device Registration
PEAK	Program Encryption / Authentication Key
PKC	Public Key Certificate
PKC-ID	PKC Identifier: the hash of the Public Key Certificate
PKCS	Public Key Cryptography Standard
PKI	Public Key Infrastructure
PPV	Pay Per View
PRF	Pseudo Random Function
PSI	Program Specific Information
RI	Rights Issuer
RIAK	Right Issuer Authentication Key
RO	Rights Object
ROT	Root Of Trust
RSA	Rivest-Shamir-Adelman public key algorithm
RTP	Real Time Protocol
SAK	Service Authentication Key
SAS	Service Authentication Seed
SBDF	Shortform Broadcast Domain Filter (a.k.a. shortform_domain_id)
SEAK	Service Encryption / Authentication Key
SGK	Subscriber Group Key
SHA-1	Secure Hash Algorithm
SI	Service Information
SK	Session Key
TAK	Traffic Authentication Key
TAS	Traffic Authentication Seed
TDK	Token Delivery Key
TEK	Traffic Encryption Key
TKM	Traffic Key Message
UDF	Unique Device Filter
UDK	Unique Device Key
UDN	Unique Device Number
UDP	User Datagram Protocol
UGK	Unique Group Key
UTC	Universal Time Clock

3.3 Notations

E{K}(M)	Encryption of message '*M*' using key '*K*'
D{K}(M)	Decryption of message '*M*' using key '*K*'
A{K}(M)	Authentication of message '*M*' with key '*K*'
V{K}(M)	Verification of message '*M*' with key '*K*'
A | B	Bitwise OR of *A* and *B*
A & B	Bitwise AND of *A* and *B*
A || B	Concatenation of *A* and *B*
A / B	*A* and/or *B*
A << B	Bitwise shift left of *A* by *B* bits. The *B* most significant bits of *A* are discarded, whilst the *B* least significant bits after the shift contain zeros.
A >> B	Bitwise shift right of *A* by *B* bits. The *B* least significant bits of *A* are discarded, whilst the *B* most significant bits after the shift contain zeros.
AES-128-ENCRYPT{K}(M)	Encrypts the message *M* with AES, using the 128-bit key *K*.
ceil(X)	Rounds up the real value *X* to the lowest integer *N* such that $X<N$.
floor(X)	Rounds down the real value *X* to the highest integer *N* such that $X>N$.
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*.
HMAC-SHA1-t	The HMAC-SHA1 computation truncated to the most significant *t* bits, i.e. MSB*t*(HMAC-SHA1)
HMAC-SHA1{K}(M)	HMAC-SHA1 computation of message '*M*' using key '*K*'
SHA1-t	The SHA1 computation truncated to the most significant *t* bits, i.e. MSB*t*(SHA1)

4. Introduction

Digital Rights Managementdefines the mechanisms to deliver DRM Content and Rights Objects to a consuming device. In the existing specification suite, devices are assumed to be capable of two-way interaction with other entities, such as a Rights Issuer. In a typical broadcast environment, this may not be the case and devices may exists that can only receive information broadcast over a shared medium.

4.1 Version 1.0

In BCAST 1.0 ERP, the need for adaptations, extensions and guidelines has been identified for the following OMA Digital Rights Management [DRM-v2] items:

- ROAP Protocol

  The ROAP protocol is specified assuming a bi-directional communication mechanism between Device and Rights Issuer. A broadcast (i.e. uni-directional) equivalent for the functionality provided by the ROAP protocol is required. Bandwidth usage is very important in broadcast and protocol messages should be optimised for size.

- Rights Expression Language

  There is a need for additional types of usage that are typical to the broadcast model, e.g. time-shift, record, edit. These may also have non-standard constraints such as impulse-pay-per-view, prepaid.

  NOTE: Impulse pay-per-view is a content purchase model where participating receiving client devices are sufficiently physically secure that they are trusted with the pay-per-view program keys in advance. Each client device tracks locally which pay per view (PPV) programs a user actually chooses to view and then periodically reports these purchases to a billing system that charges the user. This purchase model allows a further increase in scalability for PPV programs, since a purchase is made instantly, locally in the device, and the infrastructure equipment is only responsible for periodically collecting cumulative purchase reports from receiving client devices. This purchase model requires protection of the cryptographic keys, purchase information, purchase recording and reporting software. To support impulse pay-per-view for receiving client devices that do not necessarily have a return path capability, the devices can pre-purchase credit from a kiosk. Once that credit is used, the subscriber can return to a kiosk, to report back purchases and to buy more credit.

- Subscription Group Addressing

  - This is a feature that allows – per instance of content protection – to define the exact group of broadcast receivers that will be capable of accessing the protected content. It is required for fine-grained management of broadcast subscription services.

- Authentication of Broadcast Rights Objects and broadcast content

  - The bandwidth efficiency requirements of broadcast systems may necessitate a broadcast specific authentication scheme for BCROs and content.

- Broadcast Service Support

- Token Management

This specification specifies the mentioned mechanisms. This specification is not stand-alone; it must be interpreted in the context of the existing OMA DRM v2.0 suite of specifications. Its goal is to provide alternative mechanisms for those parts of the standard that do not comply to the specific constraints of broadcast systems: one-way communication and bandwidth efficiency. Next to that, it also defines support for additional broadcast concepts such as ‘broadcast service’, (frequent) re-keying of broadcast content protection and broadcast usage models.

This specification and the DRM profile related parts of [BCAST10-ServContProt] very closely follow the IPDC over DVB-H 18Crypt profile for service and content protection described in [ETSI 102 474], Annex B. Most, but not all, parts of this

specification are identical to their counterparts in [ETSI 102 474], although they appear in different order. In fact, DRM profile is an extended version of 18Crypt, and 18 Crypt is a backwards compatible subset of the DRM profile.

Technical differences between DVB [ETSI 102 474] and BCAST [BCAST10-ServContProt] [DRM20-Broadcast-Extensions] include, but are not necessarily restricted to, the following:

- BCRO format
    - BCROs may be signed in BCAST (not so in DVB 18Crypt)
    - Subscriber group addressing (two additional addressing modes in BCAST that do not exist in DVB 18Crypt)
- STKM format
    - protection_after_reception (flag is always 0 in DVB 18Crypt)
    - traffic_key_lifetime (3 instead of 4 bits in DVB 18Crypt, with MSB always 0)
    - next_master_key_index_flag (field is always 0 in DVB 18Crypt)
    - next_master_salt_flag (field is always 0 in DVB 18Crypt)
    - master_salt_flag (field is always 0 in DVB 18Crypt)
    - next_master_key_index (field is not present in DVB 18Crypt)
    - master_salt (field is not present in DVB 18Crypt)
    - next_master_salt (field is not present in DVB 18Crypt)
    - DCF encryption (TKM_ALGO_DCF) (does not exist in DVB 18Crypt)
- Protection signalling in SDP
- Global Status Codes used in Server Side Interfaces and Messages (do not exist in DVB 18Crypt)
- Token delivery response message (may be signed in BCAST, not so in DVB 18Crypt)
- Adapted PDCF file format (does not exist in DVB 18Crypt)
- Traffic authentication for ISMACryp (not used in DVB 18Crypt)

The rest of the document is organized as follows. Section 5 describes the processing of keys at the different layers in the 4-layer OMA BCAST service protection architecture [BCAST10-ServContProt]. Section 6 describes the Authentication Hierarchy of the 4-layer OMA BCAST service protection architecture. Section 7 describes the management of domain and devices in broadcast environments. The new format and mechanism for the delivery of rights objects called BCRO (Broadcast Rights Object) are defined in Section 8. The concept of token management and Subscriber group are described in Section 9 and 10 respectively. Section 11 describes the broadcast service support that allows the secure delivery of broadcast stream to a Device. Section 12 describes various Rights Issuer services. The PDCF adaptations for Traffic Encryption Key Streams are described in Section 13.

All sections of this specification apply to the DRM profile, as specified in [BCAST10-ServContProt]. A few sections also apply to the Smartcard profile as specified in [BCAST10-ServContProt]. Which sections apply to the Smartcard profile is specified in [BCAST10-ServContProt].

5. Four-Layer Key Hierarchy For Service Protection

The OMA BCAST service and content protection architecture consists of a four layer key hierarchy [BCAST10-ServContProt]. This section explains the handling and processing of keys at different layers.

5.1 Registration Layer-Layer 1 Keys (Broadcast Mode)

For the Broadcast Mode of operation, a set of keys are delivered to the Device at the registration layer. These keys are used for authentication and decryption purposes.

The keys are delivered to the device in a protected format, called a keyset_block, as part of the device registration data (refer to Section 7.2.2.2 for details).

The RI generates a session key (SK) to protect the keyset_block (UGK, (F)SGK1..n, UDK, BDK, RIAK, UDF, SBDF, LBDF and/or TDK), which carries the keyset described in Section 7.2.2.2.3 (see 5).

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

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 Section 7.2.2.2 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 session key_block. Refer to Section 7.2.2.2 for details.

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

The term Inferred Encryption Key (IEK) is used to encrypt and decrypt the CEK/SEK/PEK in a BCRO. The IEK is "derived" from the UGK, (F)SGK, UDK or BDK to decrypt the BCRO, such that

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

or

$$IEK = HMAC\_SHA1\_128\{NK_i \,||\, ... \,||\, NK_j\}(BCI)$$

where the DKs are the Device Keys ordered according to the index (such that i < j) that are required for creating the key for the desired group. The Device Keys are obtained using the scheme described in Section 10.3.4.4.

or

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

or

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

The BCI parameter is 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.

5.2 Long-Term Key Message Layer-Layer 2 Keys

Keys in this layer can be delivered either over broadcast or interaction channel. The following sections describe the processing of keys both in the broadcast and interaction modes.

5.2.1 Broadcast Mode

The SEK and PEK are transmitted to the device on the Long Term Key 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 Section 10.2) as follows:

- **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 derived from 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 derived from the subscriber group keys ((F)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 derived from 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 derived from 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 [DRM-v2].

5.2.2 Interaction Mode

If a Rights Object delivered via the interaction channel contains a CEK, it will be processed according to [DRM-v2]. If it contains a SEK or PEK, this key is protected in the same way as the CEK in [DRM-v2].

5.3 Short-Term Key Message Layer-Layer 3 Keys

The Traffic Encryption Key (TEK) is transmitted in this layer. The TEK will be encrypted using either a Program 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 sections to be used.

5.3.1 Service Based Subscription

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

If access rights change per program, a program key is used within the Short Term Key Message, but is never delivered separately in a Rights Object. The scheme described in Section 5.3.2 is used.

If access rights do not change per program, a program key is not used and the scheme below is followed.

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

and

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

The SEK is transmitted to devices as part of the Rights Objects on the Long Term Key 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 Devices and Broadcast Devices.

Figure 1 shows the key hierarchy for the case of a service based subscription.

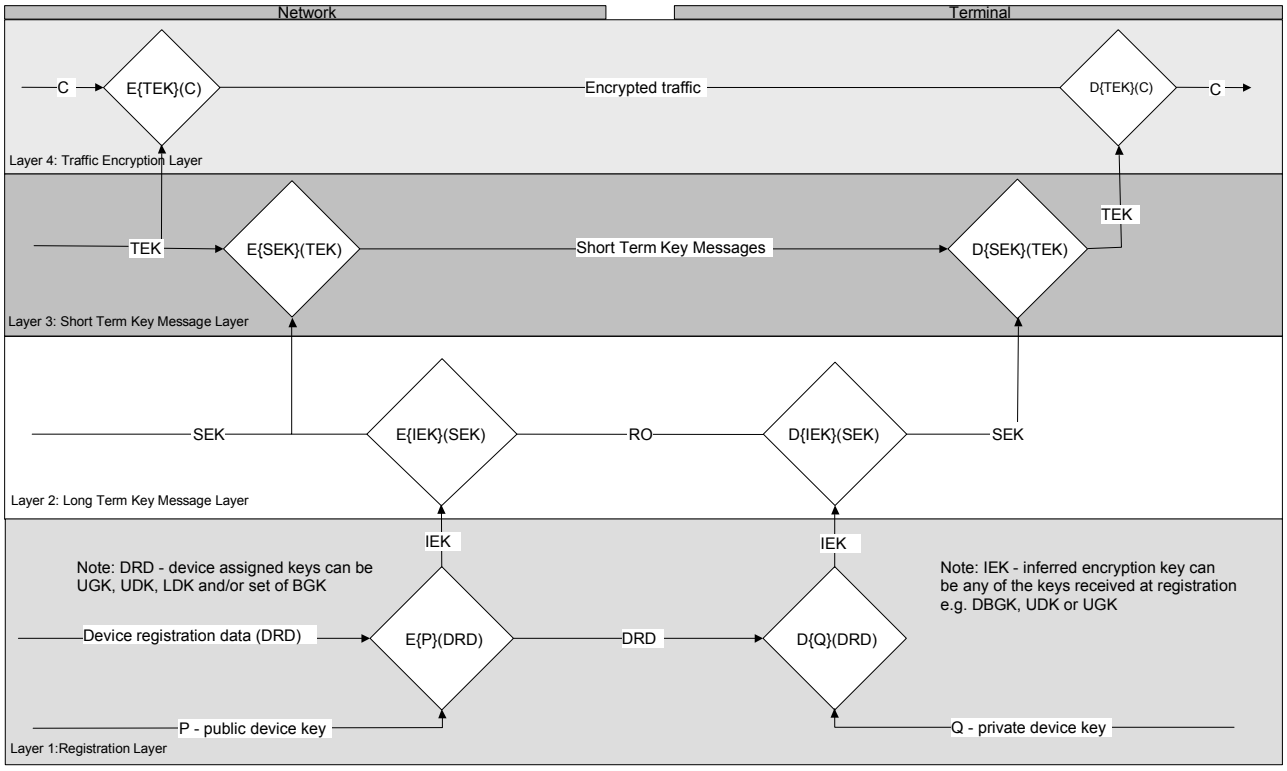


Figure 1: 4-layer key hierarchy - use of SEK only

5.3.2 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 Short Term Key Message. So the STKM 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.

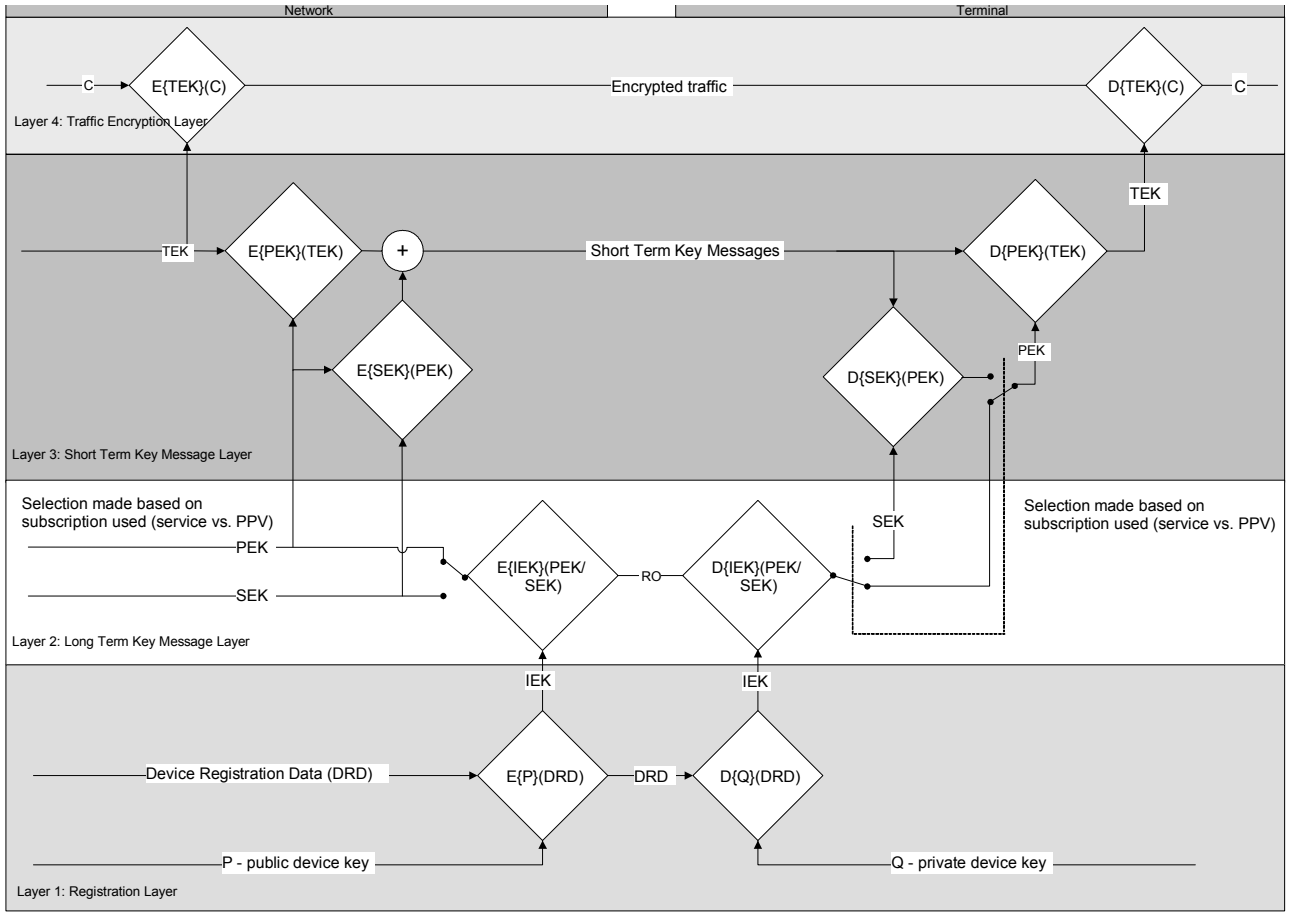


Figure 2: 4-layer key hierarchy - use of PEK and SEK

Figure 2 shows the four layer key hierarchy in the case of service subscription and pay-per-view.

5.4 Traffic Encryption Layer-Layer 4 Keys

On the Layer 4, the data is encrypted using one of IPsec, SRTP or ISMACryp. This layer is called the Traffic Encryption Layer. The key used to encrypt the traffic on this layer is called Traffic Encryption Key, or TEK.

6. Authentication

This section describes the authentication "hierarchy" of the four-layer OMA broadcast service protection architecture for the DRM profile [BCAST10-ServContProt]. Figure 3 illustrates how authentication is handled at the different layers of the 4-layer service protection architecture.

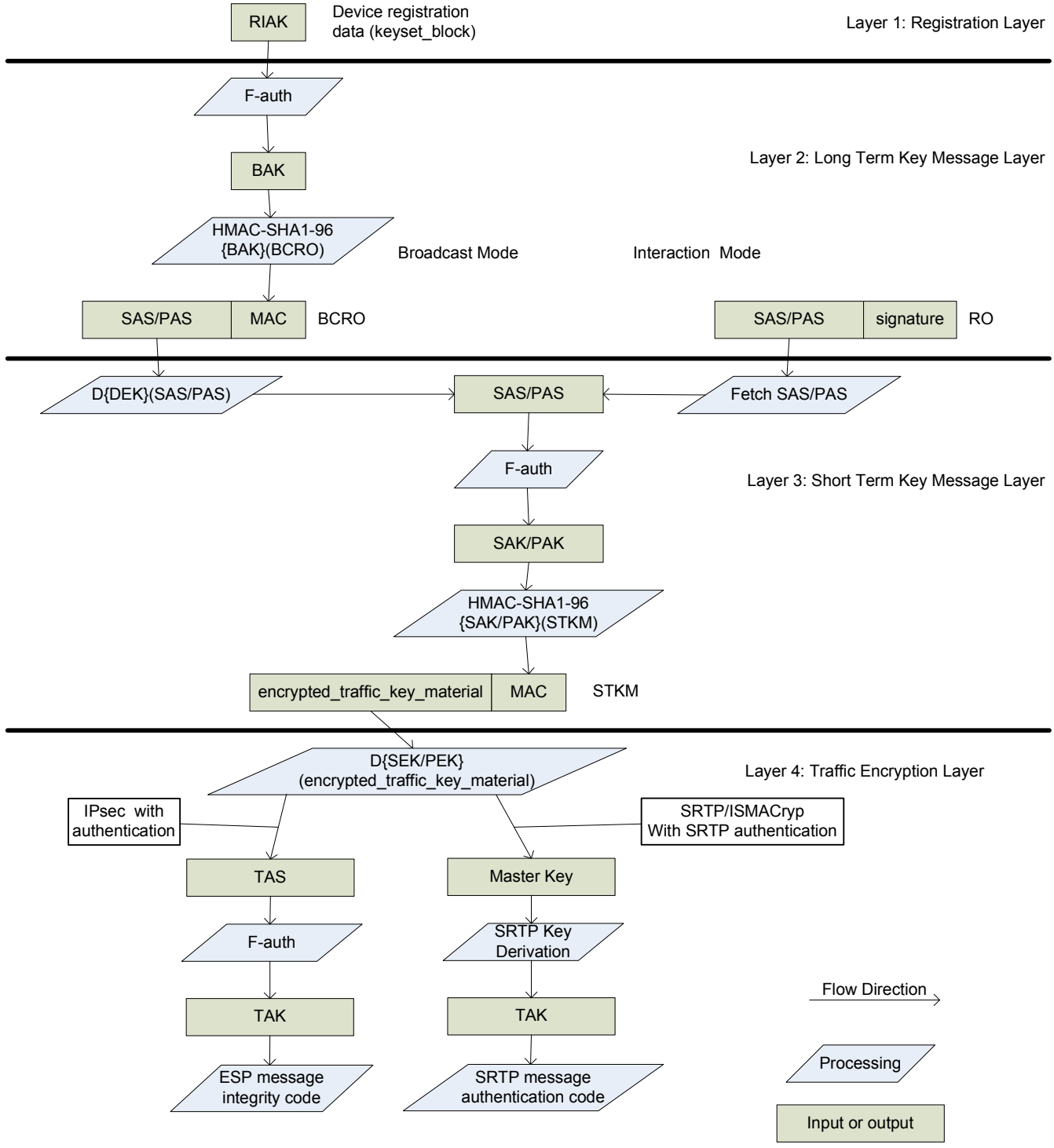


Figure 3: Authentication hierarchy

Where,

F-auth is a general authentication function that is described in Appendix C.14.5.

Note that the STKM, BCRO and RO message structures in Figure 3 only show the relevant parts of the corresponding message structures. In addition to the keying material, these messages also contain other information (for more details see [BCAST10-ServContProt]).

The keys used for authentication at the different layers of the OMA BCAST service protection architecture for DRM profile are described in the following sub-sections:

6.1 Registration Layer-Layer 1 Keys

This layer only has the RI Authentication Key (RIAK). The RIAK is delivered during registration as part of the device_registration_response() message, described in Section 7.1.3.2.

6.2 Long-Term Key Message Layer-Layer 2 Keys

This layer only has BCRO Authentication Key (BAK). The BAK, which is derived from the RI Authentication Key (RIAK), is used to authenticate and verify the integrity of the BCRO message (see Section 8.2.1).

6.3 Short-Term Key Message Layer-Layer 3 Keys

This layer has two authentication keys: the Program Authentication Key (PAK) and the Service Authentication Key (SAK). The PAK is derived from the Program Authentication Seed (PAS) and SAK is derived from the Service Authentication Seed (SAS) using the F-auth function, The PAS and the SAS are delivered as part of the BCRO/RO. The PAK and/or SAK are used to authenticate and validate the integrity of the STKM.

6.4 Traffic Encryption Layer-Layer 4 Keys

This layer only has one TAK (Traffic Authentication key). The TAK is used for the integrity protection of the broadcast stream. When IPsec with authentication is used, the Traffic Authentication Seed (TAS) is derived from the decrypted keying material at Layer 3 using SEK/PEK. From the TAS, the TAK is derived using the F-auth function and then the TAK is used to verify the ESP integrity code. When SRTP/ISMACryp with SRTP authentication is used, the decrypted keying material, at Layer 3 using SEK/PEK, is used as the SRTP Master Key. The TAK is derived from the Master Key and used to verify the integrity of the SRTP integrity code.

7. Broadcast Device and Domain Management

In this chapter, binary messages for communication between a Rights Issuer and Broadcast Devices and Mixed-Mode Devices are defined. When these binary messages are communicated over a broadcast channel, they SHALL be carried in an RI Service, see Chapter 12.

In Section 7.1 the common message fields used in the rest of chapter 7 are specified.

In the Sections 7.2 and 7.3 the process of Device registration is described, which enables the reception of BCROs, token handling and domain management over the broadcast channel is described. This process corresponds to the delivery of the Layer 1 (Registration Layer) Keys, which are used for authentication and decryption purposes.

Section 7.2 specifies how to register Broadcast Devices which do not have a return channel to the RI. This process consists of the offline notification of the Device data to the RI and of the notification of the registration data from the RI to the Device. Mixed-Mode Devices may use the offline notification of the Device data as well.

A Mixed-Mode Device or a Broadcast Device connecting via a connected Device may register using the ROAP protocol, as is specified in Section 7.3. This on-line registration, which is based on the OMA DRM v2.0 ROAP protocol, contains some extensions needed for the transmission of registration information, enabling the reception of BCROs, token handling and domain management over the broadcast channel.

Section 7.4 specifies an off-line protocol for requesting certain actions from the RI. Examples of such actions are re-registration, join or leave domain and token requests.

The RI has the possibility to send the registered Device a 1-pass message updating important data as RI certificate, DRM Time, contact number or domain information over the broadcast channel. There are also messages defined for use over the broadcast channel for the delivery of tokens or for forcing a device to join or leave a domain. These 1-pass messages are described in Section 7.5.

Section 7.6 is about the token handling. It describes how a Device can request the RI offline to purchase tokens and how these tokens are delivered to the Device over the broadcast channel. It also describes how a Device reports his token consumption to the RI when requested.

Furthermore, Section 7.7 handles the Domain Management. OMA DRM v2.0 Domains and Broadcast Domains and the protocols needed for their management over the broadcast channel are described.

7.1 General Issues

7.1.1 Message Description Tables

In this chapter, most messages are specified using at least two components: the message description and the message syntax. Each message description contains a table with three columns.

- The first column contains the names of the parameters in the message.
- The second column describes whether a parameter is optional "O" or mandatory "M". In this column, "O" means that the parameter MAY be included in the message, but the device MUST support the interpretation of the parameter. "M" means that the field MUST be included in the message.
- The third column contains remarks.

7.1.2 Common fields

The various messages described in the following sections have some fields in common. These include the following fields:

message_tag: this parameter identifies the type of the message. Refer to Section C.13 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 doesn't support. Refer to Section C.13 for the value of this parameter.

longform_udn(): the long form of the Unique Device Number (UDN). Refer to Section 7.2.1.2 for details.

status: indicates the current status. In the description of the messages that contain this field, a table with possible status values is included. The status parameter SHALL indicate one of these values. The Device SHALL ignore messages with other status values.

certificate_version: 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). The certificate_version can range from 0x00 to 0xff. The value is created by the RI. The RI can start at any value. As soon as something changes in the certificate chain, the RI increases the certificate_version by 1. This saves the Devices the time to go through the complete certificate chain every time they see a message with a certificate chain, which is the same as the one in the previous message(s).

ri_certificate_counter: this parameter indicates the depth of the RI certificate chain. The certificate chain can contain at most 7 certificates. If the ri_certificate_counter contains a value 0, no certificate chain is included. To save bandwidth, the size of error status messages can be reduced by omitting the certificate chain.

c_length: this parameter indicates the length in bytes of the ri_certificate.

ri_certificate: when present, the value of the *ri_certificate* parameter SHALL be a certificate chain including the RI 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.

signature_type_flag: a flag to signal type of signature algorithm used:

signature_type_flag	Value ($_h$)	Remark
RSA 1024	0x0	
RSA 2048	0x1	
RSA 4096	0x2	
reserved for future use	0x3	not used in this version of the specification

Refer to Appendix C.9 for further details.

signature_block: the signature SHALL enable a single source authenticity check on the message. The algorithm used for the signature is RSA-1024 or RSA-2048 or RSA-4096. The signature SHALL apply to the implementation guidelines of PKCS#1, as specified in C.9.

ocsp_response_counter: this parameter indicates the depth of the OCSP response chain. The OCSP response chain can contain at most 7 OCSP responses. If the ocsp_response_counter contains a value 0, no OCSP response chain is included. To save bandwidth, the size of error status messages can be reduced by omitting the OCSP response chain.

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 elements. A Device SHALL check that an OCSP response is present in the received message.

local_time_offset_flag: binary flag to signal presence of the local_time_offset parameter. If the local_time_offset_flag contains a value 0x1, the local_time_offset field is present. If the local_time_offset_flag contains a value 0x0 the local_time_offset field is absent.

local_time_offset: this parameter indicates the local time offset from the (UTC) drm_time as explained in Annex A.4.

message_seq_number: the message_seq_number is the message_seq_number which was present in the request (using the offline NSD protocol) to which this message is a response. This message_sequence_number is encoded in BCD.

time_stamp_flag: binary flag to signal presence of both the parameters registration_timestamp_start and registration_timestamp_end or the parameters domain_timestamp_start and domain_timestamp_end. A value of 0x1 indicates the presence of the fields, a value of 0x0 the absence.

drm_time: this parameter defines the time in Universal Time Coordinated (UTC). This 40-bit field contains the current UTC time and Modified Julian Date (MJD). The 16 most significant bits in the field contain the MJD. The 24 least significant bits contain the UTC time encoded in BCD. Refer to C.8 for more details on the calculation of Modified MJD.

EXAMPLE: 93/10/13 12:45:00 is coded as "0xC079124500".

7.2 Broadcast Device Registration

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 interaction channel or the interaction channel is not able to make a connection to the RI, but the device is able to create an other connection to a connected OMA device. This device is called an unconnected device, and is covered in [DRM-v2] Section 14.

The device has no interaction 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.

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 this document.

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

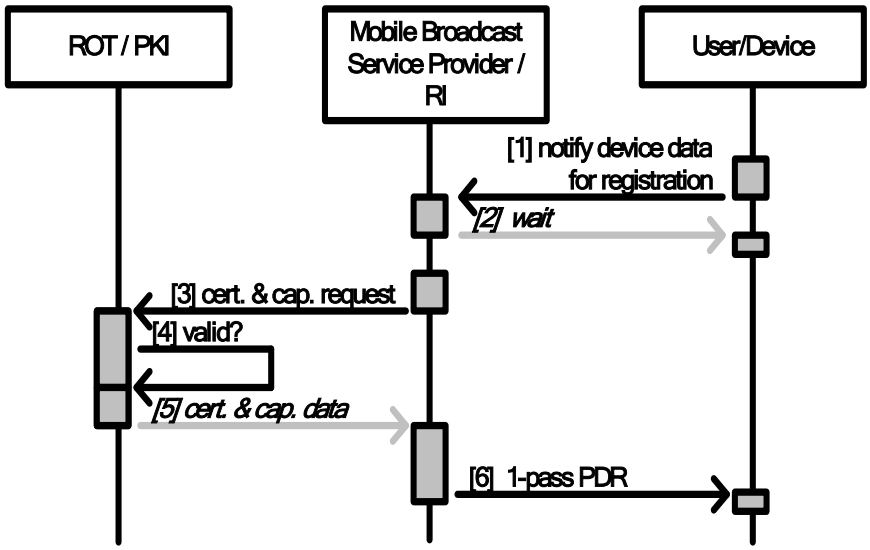


Figure 4: Registration for broadcast mode of operation with one ROT

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 Rights Issuer has the device data from the device [1] via the protocol described in Section 7.2.1, the RI contacts the Root of Trust (ROT) requesting the certificate and capabilities of the Device [3], while the device is entered into registration mode and awaits the registration data [2].

The Root of Trust decides 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.

If the RI received the requested certificate and capabilities from the ROT [5], the RI SHALL send back a registration data message to the device [6].

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 Section 7.2.2, together with the registration data (in the format of the device_registration_response() message). 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 longform_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.

7.2.1 Offline Notification of Detailed Device Data

7.2.1.1 Theory of Operation

The offline Notification of Detailed Device Data protocol is also known as the "offline NDD protocol". The notification of the device data is performed off-line, by means of the device_data_inform() message as defined in Section 7.2.1.3.1.

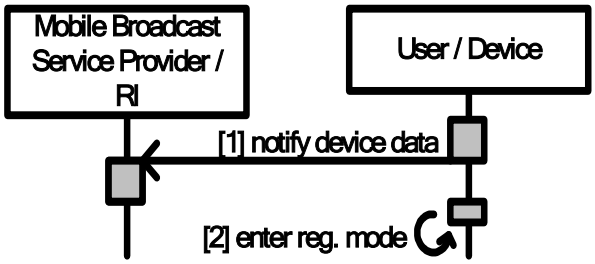


Figure 5: Offline NDD protocol

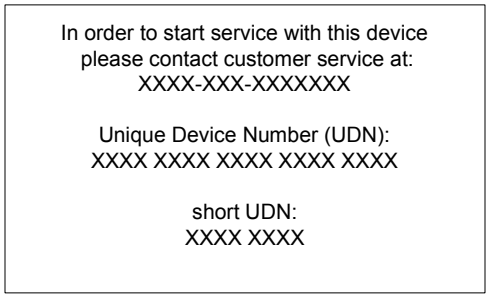
Explanation of the protocol:

The purpose of this protocol is to transfer device data somehow to the RI, in case the device does not support a return channel to the RI. After the user has let the device know that he/she wants to register at an RI, the device produces the device_data_inform() message (refer to Section 7.2.1.3.1 for details) and make this data available to the user.

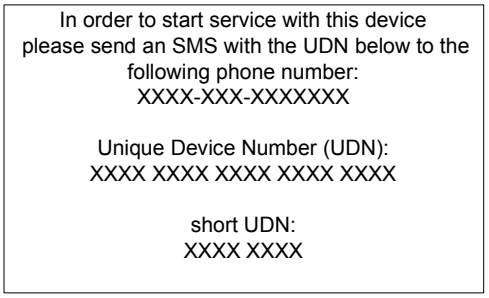
The data of the device_data_inform() message consists of a several series of decimal digits and possibly an alphanumeric character. The user needs to transfer these series somehow to the RI. In order to aid the user in this, the device MAY display a dialogue with instructions. Notifying the device data 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.

An example of a displayed message follows, where the following information is reported back to the RI. Please note that when using displays like in the examples, it is useful to present the numeric fields in the order shown[1]:

[1] 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. Please note: the short UDN will only be displayed after the first registration, when that data MAY be available for display.



An example dialogue showing instructions for vocal notification of UDN to callcenter



An example dialogue showing instructions for notification of UDN per SMS to callcenter

Figure 6: Examples 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 Device Data protocol.

After the notification of the device data, user needs to put the device into registration mode [2]. When put into registration mode, device SHOULD start to listen for the device registration data for a limited time.

7.2.1.2 Unique Device Number (UDN)

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 data format SHALL be used to construct a Unique Device Number (a.k.a. UDN):

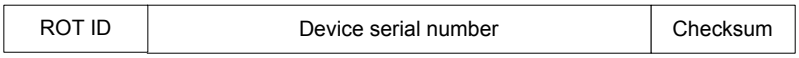


Figure 7: Unique Device Number

Table 1: UDN explanation

Field	Length (digits)	supporting up to
rot_id	3	1000 ROT
device_serial_number	14	10,000 Billion devices
checksum	3	

This totals to 20 digits. The fields are explained below:

rot_id: The first 3 digits in the UDN identify the ROT. Every ROT has an own unique ID.

device_serial_number: There are 10,000 billion (1014) possible device_serial_numbers. This range MAY be subdivided in subranges from which separate entities may issue device serial numbers independently.

checksum: The final digits of the device ID number are check digits, akin to a checksum. The 3 digits allow 1 out of 103 possible errors to remain undetected. The algorithm to construct the checksum SHALL be as specified in Appendix C.6.2.

7.2.1.2.1 Syntax

The 20 digits of the UDN are encoded in BCD (Binary-Coded Decimal) format into the longform_udn(). The message syntax is specified below:

Table 2: longorm_udn

fields	length (bits)	type
longform_udn(){		
rot_id	12	bslbf
device_serial_number	56	bslbf
checksum	12	bslbf
}		

Note: The UDN SHALL be constructed according to the above mentioned message syntax. When the UDN is displayed or in other ways presented to the end user, a(ny) checksum digit with value "10" SHALL be represented by an alphanumeric character different from {0..9}, for example X or Z. This ensures the RI will always receive 20 "characters" from the end user notification, providing an easy way to count if the information is complete.

Notice that there is a field named shortform_udn too. See Section 7.4.1 for more details.

7.2.1.3 device_data_inform() Message

The device_data_inform() message is used to send Detailed Device Data to the RI for Registration.

7.2.1.3.1 Description

The Device data SHALL be 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 3: Notify device data message parameters

Parameter	(M)andatory / (O)ptional	Remark
version	M	
contact_nr	O	
longform_udn()	M	

version: 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_number: 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 Section 7.2.1.1 for an example). This number could also be provided in human readable form in other ways.

longform_udn():identifies the unique_device_number to the RI. The UDN SHALL be part of the credentials entered into the device, like the private key and the certificate. Refer to Section 7.2.1.2 for details.

7.2.1.3.2 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 4: Device data

Parameter	Format and length	Description
version	1 digit	
contact_number	15 digits	dependent on target telco network
longform_udn	20 digits	constructed as described in Section 7.2.1.2.1.

7.2.2 Push Device Registration Protocol

7.2.2.1 Theory of Operation

Note: This protocol is also known as the "1-pass PDR protocol", short for Push Device Registration protocol.

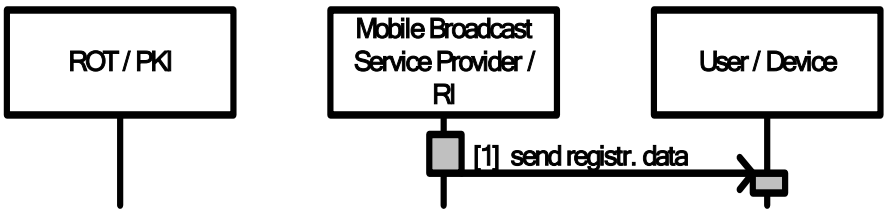


Figure 8: 1-pass PDR protocol - (first) device registration

Note: Transmission of registration data is performed on-line via the broadcast channel. The registration data (device_registration_response() message) is specified in Section 7.2.2.2

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 Section 7.2.2.2) or the domain_registration_response() message (refer to Section 7.7.4). The RI SHALL use the RI mechanisms described in Section 12 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.

7.2.2.2 device_registration_response() Message

7.2.2.2.1 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 5: device_registration_response message description

device_registration_response()		
Parameter name	(M)andatory /	remark

	(O)ptional	
message_tag	M	global, not encrypted
protocol_version	M	global, not encrypted
sign_bcros_flag	O	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_type	M	device specific, not encrypted
signature_type_flag	M	global, not encrypted
shortform_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
flexible_device_data	O	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

message_tag: this parameter identifies the type of the message. Refer to Section C.13 for the value of the message_tag.

protocol_version: this parameter indicates the protocol_version of this message. See Section 7.1 for more details.

sign_bcros_flag: this (OPTIONAL) flag is turned ON if the BCROs will be signed. If this flag is present, the **reserved_for_use** flag is reduced to 3 bits.

longform_udn(): the long form of the UDN. Refer to Section 7.2.1.2.1 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 6: 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.

Note: refer to Section C.7 for the value of the error codes.

certificate_version: a numerical representation of the version of the RI certificate. See Section 7.1.2 for more details.

ri_certificate_counter: this parameter indicates the depth of the RI certificate chain. See Section 7.1.2 for more details.

c_length: this parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): this parameter SHALL be present. See Section 7.1.2 for more details.

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 does not need to 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. See Section 7.1.2 for more details.**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. See Section 7.1.2 for more details. 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 local_time_offset parameter. See Section 7.1.2 for more details.

time_stamp_flag: binary flag to signal presence of both parameter registration_timestamp_start and registration_timestamp_end. See Section 7.1.2 for more details.

subscriber_group_type: This field indicates whether the Device is assigned to a Fixed Subscriber Group of size 256 or 512 Devices, or to a Flexible Subscriber Group. See Table 7 for more details.

Table 7: The meaning of subscriber_group_type

subscriber_group_type	Value ($_h$)	remark
data absent	0x0	will signal absence of keyset_block e.g. on error status to save bandwidth.
reserved for future use	0x1-0x7	not used in this version of the specification
set of 8 SGKs	0x8	indicates a Fixed Subscriber Group size of 256 Devices
set of 9 SGKs	0x9	indicates a Fixed Subscriber Group size of 512 Devices
reserved for future use	0xA-0xE	not used in this version of the specification
flexible group size, set of FSGKs	0xF	indicates a Flexible Subscriber Group size

signature_type_flag: a flag to signal type of signature algorithm used. See Section 7.1.2 for more details.

short_udn_flag: binary flag to signal presence of the shortform_udn field.

short_udn_flag	Value ($_h$)	remark
data absent	0x0	
data present	0x1	

surplus_block_flag: Binary flag to signal the presence of the surplus_block field.

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: an symmetric AES encryption key to address a unique group. This key is also known as UGK. The key length SHALL be 128 bit.

Note: This key is wrapped into the keyset_block. (Refer to 7.2.2.2.3).

subscriber_group_key: a set of AES symmetric encryption keys used for the deduction of the zero message Subscriber Group key (DEK), which is needed to decrypt the SEK and/or PEK. These keys are also known as Subscriber Group Keys (SGKs). The key length SHALL be 128 bit.

Note: this field is only present in the case of assignment of the Device to a fixed Subscriber Group of size 256 or 512 Devices. It is then wrapped into the keyset_block. (Refer to 7.2.2.2.3).

flexible_subscriber_group_key: a set of AES symmetric encryption keys used for the deduction of the zero message Subscriber Group key (DEK), which is needed to decrypt the SEK and/or PEK. These keys are also known as Flexibe Subscriber Group Keys (FSGKs). The key length SHALL be 128 bit.

Note: this field is only used in the case that a device is assigned to a Flexible Subscriber Group. When the field is present, it is wrapped into the keyset_block.(Refer to 7.2.2.2.3).

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: This key is wrapped into the keyset_block. (Refer to 7.2.2.2.3).

unique_device_filter: This 40-bit address is used as a unique identifier of the device for a specific RI (each RI has its own address space). The Unique Device Filter is also known as UDF. This address is wrapped into the keyset_block. (Refer to 6.1.3.2.2).

In case of Fixed Suscriber Group addressing, the following applies. In the case of a group size of 256 devices, the first 32 bits contain the **fixed_group_address** field, whilst the last 8 bits contain the **fixed_position_in_group** field. In the case of 512 devices, the first 31 bits contain the **fixed_group_address** field whilst the last 9 bits contain the **fixed_position_in_group** field.

In the case of Flexible Subscriber Group addressing, this field contains a 40-bit unique address.

Note: An RI can decide to use both Flexible Subscriber Groups and Fixed Subscriber Groups. In this case the RI has to take care that the Group Address of a Fixed Subscriber Group does not equal the first 31 or 32 bits of a UDF of a device in a Flexible Subscriber Group. To ensure this it is recommended that if the RI supports both Subcriber Group types, the MSB of the UDF indicates whether the Device is assigned to a Flexible Subscriber Group or to a Fixed Subscriber Group.

flexible_group_address: the address of the Subscriber Group in the case that the Device was assigned to a Flexible Subscriber Group.

Note: this field is only present in the case that the device is assigned to a Flexible Subscriber Group. It is then wrapped in the flexible_device_data structure in the keyset_block. (Refer to 7.2.2.2.3 and C.11).

flexible_position_in_group: the position of the Device in its Flexible Subscriber Group.

Note: this field is only present in the case that the device is assigned to a Flexible Subscriber Group. It is then wrapped in the flexible_device_data structure in the keyset_block. (Refer to 7.2.2.2.3 and C.11).

flexible_group_size_indicator: this 5-bit field indicates the size of the Flexible Subscriber Group. When flexible_group_size_indicator contains a value k, the Subscriber Group has a size of 2^k devices.

Note: this field is only present in the case that the device is assigned to a Flexible Subscriber Group. It is then wrapped in the flexible_device_data structure in the keyset_block (Refer to 7.2.2.2.3 and C.11).

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: This key is wrapped into the keyset_block. (Refer to 7.2.2.2.3).

token_delivery_key: this is the Token Delivery Key (TDK), which is used in Section 7.6.4.

Note: This key is wrapped into the keyset_block (Refer to 7.2.2.2.3).

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: This key is wrapped into the keyset_block. (Refer to 7.2.2.2.3).

longform_domain_id(): this parameter is also known as the Longform Broadcast Domain Filter (LBDF). Please refer to Section C.11.2 for the definition. The longform_domain_id() is used for mixed-mode operation. Note: This address is wrapped into the keyset_block. (Refer to 7.2.2.2.3).

shortform_domain_id: this parameter is also known as the Shortform Broadcast Domain Filter (SBDF). Please refer to 7.2.2.2.3. An addressing scheme used to filter messages like BCROs. The shortform_domain_id is used for broadcast mode of operation.

Note: This address is wrapped into the keyset_block. (Refer to 7.2.2.2.3).

drm_time: this parameter defines the time in Universal Time Coordinated (UTC). See Section 7.1.2 for more details.

local_time_offset: this parameter indicates the local time offset from the (UTC) drm_time as explained in Annex A.4.

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: please note that 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: please note that 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. See Section 7.1.2 for more details.

7.2.2.2.2 Syntax

Table 8: device_registration_response message syntax

fields	length	type
device_registration_response(){		
/* signature protected part starts here */		
/* message header starts here */		
message_tag	8	bslbf
protocol_version	4	bslbf
sign_bcros_flag	1	bslbf
reserved_for_future_use	3	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_type	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) {		
shortform_udn	32	bslbf
}		
/* message header ends here */		
if (signature_type_flag == 0x0){		
sessionkey_block()	1024	bslbf
} else if (signature_type_flag == 0x1)		
sessionkey_block()	2048	bslbf
} else if (signature_type_flag == 0x2)		
sessionkey_block()	4096	bslbf
}		
if (surplus_block_flag == 0x1){		
surplus_block()	(*1)	bslbf
padding_bits	(*2)	bslbf
}		
/* signature protected part ends here */		
if (signature_type_flag == 0x0){		
signature_block	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

key:

(*1) for details please refer to Section C.12.

(*2) (surplus_block() length) mod 8

7.2.2.2.3 Protection of the (Device Registration) Keyset

The device_registration_response() message is split in two parts: device global data (not time bound) and device specific (time bound).

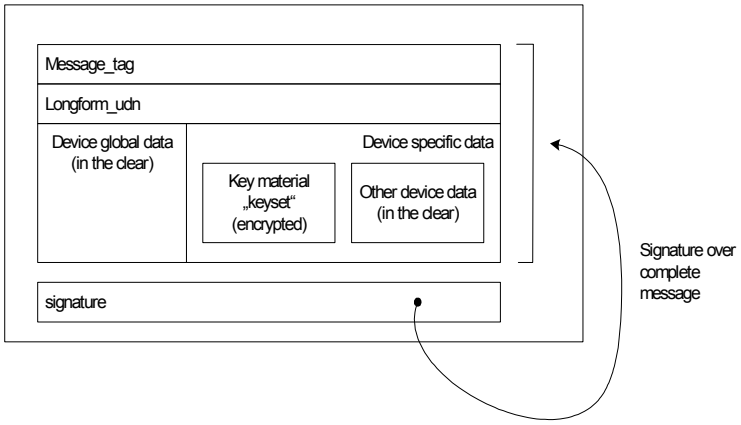


Figure 9: device_registration_response() message

The device global data SHALL be in the clear. The device specific data contains the keyset for the device. The key material SHALL be protected by encryption.

The RI SHALL use its 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. For Fixed Subscriber Group addressing, concatenate the following fields to form the keyset: UGK, SGK1..n, UDK, UDF, BDK, SBDF, LBDF (if applicable), RIAK, TDK under rules of [FIPS 197] and the Tag Length Format described in Section C.11.

   For Flexible Subscriber Group addressing, concatenate the following fields to form the keyset: UGK, UDK, UDF, BDK, SBDF, LBDF (if applicable), RIAK, TDK, flexible_device_data, FSGK1..m under rules of [FIPS 197] and the Tag Length Format described in Section C.11.

   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 Appendix A of [NIST 800-38A]. 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 [AES_WRAP] 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 1024, 2048 or 4096), 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 (1024 or 2048 or 4096) under implementation guidelines of [PKCS#1]. 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 7.2.2.2.2 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. The result SHALL be hashed under implementation guidelines of PKCS#1, as specified in Section C.9. This will produce the signature_input_data.

9. Sign the signature_input_data with RSA (1024 or 2048 or 4096) using the private key of the RI. The signature SHALL apply to the implementation guidelines of PKCS#1, as specified in C.9. 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.

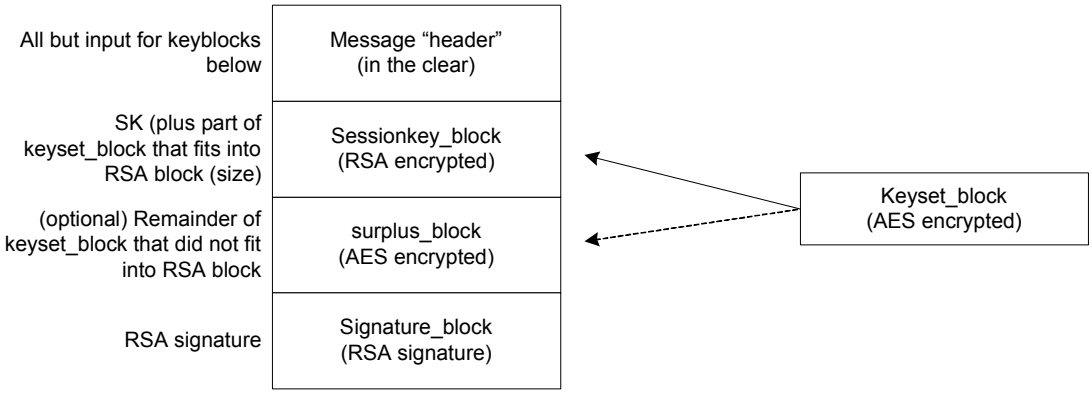


Figure 10: Structure of device_registration_response() message

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 appendix 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). See Appendix C.12 for the determination of the session key length.

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 [AES_WRAP] and the Tag Length Format described in Section C.11.

Note: The SK SHALL be stored into protected storage of the Device. The AES encrypted keyset_block MAY be stored as is into unprotected storage and decrypted by the Device upon use. If the encrypted keyset_block is not stored but the decrypted keys from that block are stored instead, the Device SHALL store all key data safely. In either case, the Device SHOULD use integrity protection of what is stored in unprotected storage to prevent tampering of the keys. The keys SHALL NOT leak outside the Device.

7.2.2.2.4 RI context stored in the Device

After the registration process, the Device SHALL store the RI Context. This RI Context SHALL contain:

RI ID, Unique device filter (UDF).

In the case the Device is assigned to a Flexible Subscriber Group: the size of the Subscriber Group, flexible_group_address and flexible_position_in_group.

The following keys:

- UDK and/or UGK.
- RIAK key. A single RIAK key is bound to a single Subscriber Group or to a single Device if no SGKs, nor FSGKs, nor UGK are issued to the Device.
- Unique device filter (UDF).
- SGK1..n (if the Device is assigned to a Fixed Subscriber Group of size 256 or 512 Devices).
- FSGK1..m and flexible_device_data (if the Device is assigned to a Flexible Subscriber Group).

For Mixed-mode Devices domain context SHALL additionally contain:

- Longform Broadcast Domain Filter (LBDF). A.k.a. "longform_domain_id()". Refer to C.11.2.

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 OMA BCAST Service Guide 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 GROs, but SHALL NOT use these BCROs for new content received after the re-registration request until the device is re-registered with the RI.

The Device SHALL have at most one RI Context per RI. An The Device SHALL support at least 6 RI Contexts for broadcast mode of operation. An existing RI Context SHALL be replaced with a newly established RI Context after successful re-registration with the same RI.

For standard addressing the keyset SHALL include a valid set of :

- UDK and/or UGK.
- RIAK key. A single RIAK key is bound to a single Subscriber Group or to a single Device if no SGKs, nor FSGKs, nor UGK are issued to the Device.

- Unique device filter (UDF).
- SGK1..n (if the Device is assigned to a Fixed Subscriber Group of size 256 or 512 Devices).
- FSGK1..m and flexible_device_data (if the Device is assigned to a Flexible Subscriber Group).

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 C.11.1.

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 C.11.2.

7.3 On-line Registration

A Broadcast Device, an Interactive Device and a Mixed-mode device can register using the ROAP protocol, either directly in case it is a connected device, or via a connected device that acts as a proxy.

An interactive device SHALL register using the ROAP protocol as defined in [DRM-v2].

For Mixed-mode Devices, or Broadcast Devices using a connected device as a proxy,extensions to the ROAP are required to allow transfer of all subscriber group key material and the authentication key for BCROs.

7.3.1 Registration Request

Rights issuers can derive from the device capabilities in the device certificate the modes of operation supported by the registering device. From this information it should be possible to determine whether to include the extensions (defined in the next section) in the registration response or not. To avoid possible confusion, an extension is defined for the **ROAP-RegistrationRequest** to allow a rights issuer to determine directly whether or not to include the broadcast extensions in **ROAP-RegistrationResponse**.

Extensions: The following extensions are defined for the ROAP-RegistrationRequest message in addition to the extensions already defined.

Broadcast Registration Request: This extension allows a device to indicate to a broadcast enabled Rights Issuer to use the broadcast extensions in the registration response.

The following schema fragment defines the *Broadcast Registration Request* extension to the ROAP schema:

```
<complexType name="roap:BroadcastRegistrationRequest">
  <complexContent>
    <extension base="roap:Extension">
    </extension>
  </complexContent>
</complexType>

<element name=”broadcastRegistrationRequest” type=”roap:BroadcastRegistrationRequest”/>
```

When included in a **ROAP-RegistrationRequest**, this extension MUST be marked as critical. It SHALL be sent as an element **<roap:broadcastRegistrationRequest>** in an **ExtensionContainer** (see Appendix C.3.2).

7.3.2 Registration Response

A Rights Issuer that receives a **ROAP-RegistrationRequest** including the *Broadcast Registration Request* extension and that does not support the broadcast extensions MUST abort the registration procedure and respond accordingly. A Rights Issuer that does support broadcast extensions MUST respond with a **ROAP-RegistrationResponse** including the following defined **Broadcast-Registration** extension.

Extensions: The following extensions are defined for the ROAP-RegistrationResponse message in addition to the extensions already defined.

Broadcast Registration: This extension allows an RI to securely transfer broadcast group key material and addressing information as well as the authentication key to use to verify authenticity of BCROs. It SHALL be sent as an element **<roap:broadcastRegistration>** in an **ExtensionContainer** (see Appendix C.3.2).

The following schema fragment defines the *Broadcast Registration* extension to the ROAP schema:

```
<complexType name="roap:SubscriberGroupKey">
 <complexContent>
  <extension base="ds:KeyInfoType">
   <attribute name="node" type="hexBinary"/>
  </extension>
 </complexContent>
</complexType>

<simpleType name="roap:ShortUniqueDeviceNumber">
 <restriction base="string">
  <pattern value="\d{8}"/>
 </restriction>
</simpleType>

<complexType name="roap:SubscriberGroupRegistration">
 <complexContent>
    <sequence>
     <element name="subscriberGroupAddress" type="roap:SubscriberGroupIdentifier"/>
     <element name="uniqueGroupKey" type="roap:UniqueGroupKey"/>
     <element name="uniqueDeviceKey" type="roap:UniqueDeviceKey" minOccurs="0"/>
     <element name="subscriberGroupKey" type="roap:SubscriberGroupKey" minOccurs="0"
maxOccurs="unbounded"/>
     <element name="shortUniqueDeviceNumber" type="roap:ShortUniqueDeviceNumber"/>
     <element name="extensions" minOccurs="0">
       <complexType>
         <sequence>
           <any  minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
         </sequence>
       </complexType>
    </element>
  </sequence>
 </complexContent>
</complexType>

<complexType name="UniqueGroupKey">
 <sequence>
  <element ref="xenc:EncryptedKey"/>
 </sequence>
</complexType>

<complexType name="UniqueDeviceKey">
```

```
  <sequence>
    <element ref="xenc:EncryptedKey"/>
  </sequence>
</complexType>

<complexType name="roap:BroadcastRegistration">
  <complexContent>
    <extension base="roap:Extension">
      <sequence>
        <element name="subscriberGroupRegistration" type="roap:SubscriberGroupRegistration"
minOccurs="0"/>
        <element name="rightsIssuerAuthenticationKey" type="roap:RightsIssuerAuthenticationKey"
minOccurs="0"/>
        <element name="encKey" type="xenc:EncryptedKeyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="broadcastRegistration" type=”roap:BroadcastRegistration”/>

<complexType name="RightsIssuerAuthenticationKey">
  <sequence>
    <element ref="xenc:EncryptedKey"/>
  </sequence>
</complexType>
```

7.3.2.1 Subscriber Group Registration

The optional **<subscriberGroupRegistration>** element holds all information regarding the subscriber group feature: subscriber group address, device position and key material.

In the case of a Fixed Subscriber Group, the **<subscriberGroupAddress>** element MUST contain the subscriber group base address and the device position. It SHALL NOT contain an access mask.

In the case of a Flexible Subscriber Group, the **<subscriberGroupAddress>** element MUST contain the flexible group address, the flexible position in group and the Unique Device Filter. It SHALL NOT contain an access mask.

The **<uniqueGroupKey**> element holds an **<xenc:EncryptedKey>** element. This MUST hold a **<ds:KeyInfo>** element, an empty **<xenc:EncryptionMethod>** element and an **<xenc:CipherData>** element. The **<ds:KeyInfo>** element MUST contains a **<ds:RetrievalMethod>** element of which the **URI** attribute references the key used to encrypt the subscriber group’s unique group key (UGK). **The <xenc:EncryptedKey>** element MUST also hold an empty **<xenc:EncryptionMethod>** element of which the **Algorithm** attribute identified the algorithm used to protect the UGK. This algorithm MUST be AES-128 Key Wrap, and the value of the **Algorithm** attribute MUST be "http://www.w3.org/2001/04/xmlenc#kw-aes128". The **<xenc:CipherData>** element contains the **<xenc:CipherValue>** element that holds the base64 encoded value of the encrypted UGK.

The optional **<uniqueDeviceKey**> element holds an **<xenc:EncryptedKey>** element. This MUST hold a **<ds:KeyInfo>** element, an empty **<xenc:EncryptionMethod>** element and an **<xenc:CipherData>** element. The **<ds:KeyInfo>** element MUST contains a **<ds:RetrievalMethod>** element of which the **URI** attribute references the key used to encrypt the subscriber group’s unique device key (UDK). **The <xenc:EncryptedKey>** element MUST also hold an empty **<xenc:EncryptionMethod>** element of which the **Algorithm** attribute identified the algorithm used to protect the UDK. This algorithm MUST be AES-128 Key Wrap, and the value of the **Algorithm** attribute MUST be "http://www.w3.org/2001/04/xmlenc#kw-aes128". The **<xenc:CipherData>** element contains the **<xenc:CipherValue>** element that holds the base64 encoded value of the encrypted UDK.

The optional **<subscriberGroupKey**> elements each hold one key associated with the binary tree of key nodes from the subscriber group. Each **<subscriberGroupKey>** is of type **<roap:SubscriberGroupKey>** which extends the **<ds:KeyInfo>** type with a single **node** attribute. The value of the **node** attribute is the hexBinary encoded node number of the node

associated with the derivation key contained by the **<subscriberGroupKey>** element. Each **<subscriberGroupKey>** element MUST hold a **<ds:KeyInfo>** element, an empty **<xenc:EncryptionMethod>** element and an **<xenc:CipherData>** element. The **<ds:KeyInfo>** element MUST contains a **<ds:RetrievalMethod>** element of which the **URI** attribute references the key used to encrypt the subscriber group's node key of node i (NKi). **The <xenc:EncryptedKey>** element MUST also hold an empty **<xenc:EncryptionMethod>** element of which the **Algorithm** attribute identified the algorithm used to protect the NKi. This algorithm MUST be AES-128 Key Wrap, and the value of the **Algorithm** attribute MUST be "http://www.w3.org/2001/04/xmlenc#kw-aes128". The **<xenc:CipherData>** element contains the **<xenc:CipherValue>** element that holds the base64 encoded value of the encrypted NKi.

The device MUST check the consistency relations between the node keys and its subscriber position as defined by the broadcast extension.

The **<shortDeviceUniqueNumber>** MUST be included in the RI Context, and MAY be used at a later moment to receive binary push (re)registration messages over the broadcast interface.

7.3.2.2 Authentication Key

The **<rightsIssuerAuthenticationKey>** holds an **<xenc:EncryptedKey>** element. This MUST hold a **<ds:KeyInfo>** element, an empty **<xenc:EncryptionMethod>** element and an **<xenc:CipherData>** element. The **<ds:KeyInfo>** element MUST contains a **<ds:RetrievalMethod>** element of which the **URI** attribute references the key used to encrypt the rights issuer's authentication key (RIAK). **The <xenc:EncryptedKey>** element MUST also hold an empty **<xenc:EncryptionMethod>** element of which the **Algorithm** attribute identified the algorithm used to protect the RIAK. This algorithm MUST be AES-128 Key Wrap, and the value of the **Algorithm** attribute MUST be "http://www.w3.org/2001/04/xmlenc#kw-aes128". The **<xenc:CipherData>** element contains the **<xenc:CipherValue>** element that holds the base64 encoded value of the encrypted RIAK.

7.3.2.3 Broadcast Registration Encryption Key

The **<encKey>** element is of type **xenc:EncryptedKeyType** from [XMLEnc]. It consists of a wrapped broadcast registration encryption key, KBRK. The **id** attribute of this element SHALL be present and SHALL have the same value as the value of the **URI** attribute of the **<ds:RetrievalMethod>** element in any **<ds:KeyInfo>** elements inside the subscriber group registration extension. The **<ds:KeyInfo>** child element of the **<encKey>** element SHALL identify the wrapping key. The child of the **<ds:KeyInfo>** element SHALL be of type **roap:X509SPKIHash**, identifying a particular DRM Agent's public key through the (SHA-1) hash of the DER-encoded subjectPublicKeyInfo value in its certificate.

7.4 Offline Notification of Short Device Data for Requests

The end user of a device might wish to formulate a particular request to the RI. This is done by the protocol as shown in Figure 11.

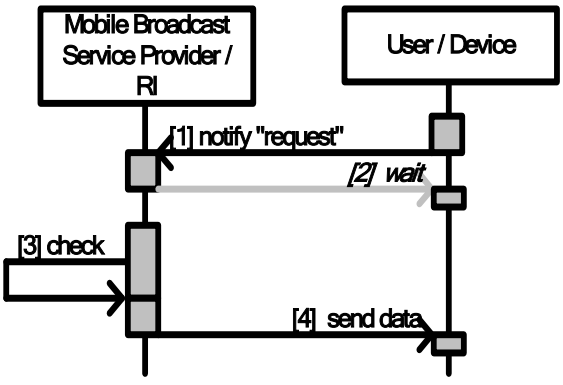


Figure 11: Action request round trip

Explanation of the protocol:

- The end user of the device formulates a request and notifies this request to the RI [1] as specified in the following sections.
- The end user waits after the request has been notified to the Customer Operations Centre in a successful way [2].
- The RI might execute additional checks and composes the data [3].
- The RI MAY send a data message to the device to update data in the device, start the execution of a particular action to produce a desired result or to inform an error status. [4].

7.4.1 Offline-Notification of Short Device Data

Notification of Short Device Data is performed offline by using the "offline NSD protocol", short for offline Notification of Short Data protocol.

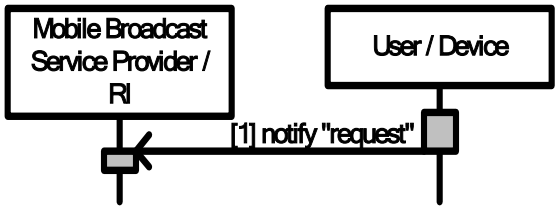


Figure 12: Offline NSD protocol

Refer to Table 10 for an overview of the possible "requests".

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:

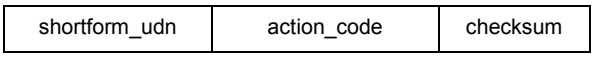


Figure 13: Action Request Code (ARC)

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 9: NSD action request code fields

ARC fields	Length (digits)	Supporting up to
shortform_udn	8	100 Million devices
action_code	2	99 action codes
checksum	2	

The length of the ARC totals to 12 digits. The fields are explained below:

shortform_udn: short form of the UDN. After first time notification of the device data to the RI, the RI MAY issue a short version of the full UDN (called shortform_udn) that is carried in the device_registration_response() message. The shortform_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 shortform_udn number concatenated by a standardised 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 shortform_udn cannot be used because the (new / different) RI does not have the shortform_udn in its database. In this case the only possibility for the hosting RI to identify the device uniquely would be via the longform_udn. It

is the responsibility of the device to decide when it is appropriate to use the longform_udn instead, for example by comparing the BSD/A ID received with the BSD/A ID remembered from registration.

action_code: following the shortform_udn the user of the device can notify an action code to the RI. The NSD protocol defined in this specification SHALL use following action_codes to construct the ARC:

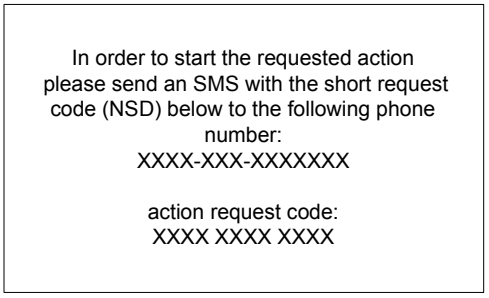
Table 10: NSD action types

Action type	action_code	Described in section
re-registration (only at same RI)	01	7.4.1.1
resend BCRO	02	12.9
reserved for future use	03 - 09	
join domain	10 - 19	7.4.1.3
leave domain	20 - 29	7.4.1.4
token_consumption_report	31 - 39	7.4.1.5
reserved for future use	40 - 49	
token_request	50 - 59	7.4.1.5
reserved for future use	60 - 69	
notify DRM time drift	70 - 89	7.4.1.7
reserved for future use	90 - 99	

checksum: the constructed shortform_udn and action_code is appended by checksum digits. The algorithm to construct the checksum SHALL be as specified in C.6.1.

Example: In order to request re-registration, the 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[2]:

An example dialogue showing instructions for vocal notification of ARC to callcenter



An example dialogue showing instructions for notification of ARC per SMS to callcenter

Figure 14: Samples of notification displays showing an ARC message

7.4.1.1 Request Re-Registration (Only at Same RI)

After sending this ARC the user will wait until he/she receives the confirmation of the RI in the form of a device_registration_response() message. Refer to 7.2.2.2.

[2] 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

7.4.1.2 Request Resend BCRO

A user can request the Rights Issuer to resend a BCRO, which was not received yet, over the broadcast channel. For that, the Device will display to the user an identification of the program or service (i.e. name of content or service) for which it did not receive the BCRO yet, together with customer service contact information. Refer to Section 12.9.2 for more details.

7.4.1.3 Request Join Domain

The Action Request Code (ARC) for the NSD protocol is formed according to the following rules:

- the first digit is used to notify the join domain action
- the second digit is used as a message sequence number 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 message sequence number from the request in the response message.

7.4.1.4 Request Leave Domain

The Action Request Code (ARC) for the NSD protocol is formed according to the following rules:

- the first digit is used to notify the leave domain action
- the second digit is used as a message sequence number 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 message sequence number from the request in the response message.

7.4.1.5 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 message sequence number 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 message sequence number from the request in the response message.

An example of a displayed message follows, where the following information is reported back to the RI[3]:

[3] 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 be available for display).

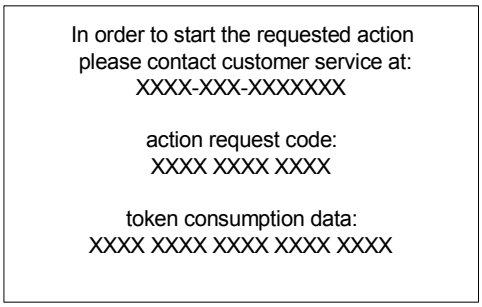


Figure 15: Samples of notification displays showing an ARC message

The token consumption data are defined as follows:

Table 11: Token consumption data

Field	Length (digits)	Supporting up to
tokens_consumed	4	9999 tokens to be reported
report_authentication_code	13	
checksum	3	

This totals 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 Section C.16 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 message_seq_number (second digit of the action code of the ARC of this message). See C.15 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 103 possible errors to remain undetected. The checksum algorithm used is the UDN checksum, see Section C.6.2.

7.4.1.6 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 message_seq_number 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 and the RI MAY request additional data (such as e.g. a bookable account). The RI SHALL incorporate the message_seq_number from the request in the token_delivery_response message.

7.4.1.7 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 89 will decode as timedrift >= 100. Some examples of valid ARC codes are given below:

E.g. 1: Device notifies 4 minute timedrift from newly received DRM time message: action code is 71.

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.

The time drift SHALL be measured by a Device when an Update DRM time message is received by the Device with status 'Success' or 'DeviceTimeError'. The 'Notify DRM time drift display' SHALL be available in the Device for the user and SHOULD be shown when an Update DRM time message is received by the Device with status 'DeviceTimeError'. The latter

message may be useful e.g. when checking a customer complaint, or when collecting statistics on time drift. This option should be used with great care since it involves user interaction. See also Section 7.5.4.1.

7.5 Inform Registered Device Protocol

7.5.1 Theory of Operation

Note: This protocol is also known as the "1-pass IRD protocol", short for Inform Registered Device protocol.

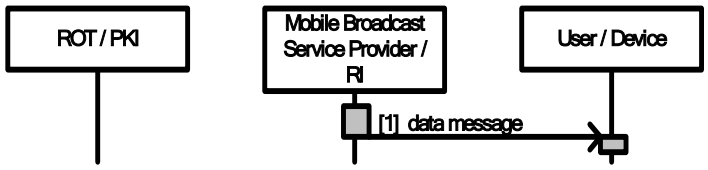


Figure 16: 1-pass IRD protocol – RI initiated message to device.

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 12: Messages of the 1-pass IRD protocol

Message name	For message syntax refer to section	Message	Remark
force to join domain	7.7.6	join_domain_msg()	
force to leave domain	7.7.7	leave_domain_msg()	
force to re-register	7.5.2	re_register_msg()	
token delivery	7.6.4.2	token_delivery_response()	
update contact number	7.5.5.1	update_contact_number_msg()	In Data Carousel, see Section 12.1.
update domain	7.7.5	domain_update_response()	
update DRM Time	7.5.4	update_drmtime_msg()	In Data Carousel, see Section 12.1.
update RI certificate	7.5.3	update_ri_certificate_msg()	

See C.13 for the coding of message_tag. The processing of each message will be discussed in following sections.

7.5.2 Force to Re-Register

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 Section 12.6 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 OMA BCAST SERVICE GUIDE 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 GROs, 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.

7.5.2.1 re_register_msg() Message

7.5.2.1.1 Description

Using the 1-pass IRD protocol (refer to 7.5.1) the RI sends a register_msg message, indirectly triggering a (re)registration . The message is specified as follows:

Table 13: Re-register message description

re_register_msg( )		
Parameter name	(M)andatory / (O)ptional	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	

message_tag: this parameter identifies the type of the message. Refer to C.13 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 doesn't support. Refer to Section C.13 for the value of this parameter.

longform_udn(): the long form of the UDN. Refer to Section 7.2.1.2.1 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 14: 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 interaction 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 interaction channel only. When the device receives this status code it will also exclusively use the interaction channel for all other messages. When the interaction 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 interaction 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 interaction channel capabilities after receiving this status code, but will not use the interaction channel.

Note: Refer to C.7 for the value of the error codes.

signature_type_flag: a flag to signal type of signature algorithm used. Section 7.1.2 for more details.

certificate_version: a numerical representation of the version of the RI certificate. See Section 7.1.2 for more details.

ri_certificate_counter: This parameter indicates the depth of the RI certificate chain. See Section 7.1.2 for more details.

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.

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.

ocsp_response_counter: This parameter indicates the depth of the OCSP response chain. See Section 7.1.2 for more details.

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-1024 or RSA-2048 or RSA-4096. See Section 7.1.2 for more details.

7.5.2.1.2 Syntax

Table 15: Re-register 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
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	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

7.5.3 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 Section 12.6 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.

7.5.3.1 update_ri_certificate_msg() Message

Using the 1-pass IRD protocol (refer to 7.5) the RI sends a update_ri_certificate_msg() message, forcing the device to update its RI certificate chain.

This update_ri_certificate_msg() trigger is almost identical to the re_register_msg() message described in Section 7.5.2.1, with the following adaptations:
being that the message_tag is different. Refer to C.13 for the value of the message_tag.

Status/Error code is Succes or NotSupported. Refer to C.7 for the value of the error codes.

7.5.4 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 timeoffset SHALL be valid.

The RI SHALL use the mechanisms described in Section 12.6 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.

7.5.4.1 update_drmtime_msg() Message

7.5.4.1.1 Description

Using the 1-pass IRD protocol (refer to 7.5) the RI sends a update_drmtime trigger message with the drmtime to the device as specified below:

Table 16: Update DRM time message description

update_drmtime_msg( )		
Parameter name	(M)andatory / (O)ptional	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	

message_tag: This parameter identifies the type of the message. Refer to C.13 for the value of the message_tag.

protocol_version: This parameter indicates the protocol_version of this message. See Section 7.1.2 for more 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 17: 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 to7.4).Please note: this capability should be used with great care.)

Note: Refer to C.7 for the value of the error codes.

local_time_offset_flag: Binary flag to signal presence of the local_time_offset parameter. See Section 7.1.2 for more details.

signature_type_flag: A flag to signal type of signature algorithm used. See Section 7.1.2 for more details.

drm_time: This parameter defines the time in Universal Time Coordinated (UTC). See Section 7.1.2 for more details.

local_time_offset: This parameter indicates the local time offset from the (UTC) drm_time as explained in Annex A.4.

signature_block: The signature SHALL enable a single source authenticity check on the message. See Section 7.1.2 for more details.

7.5.4.1.2 Syntax

Table 18: Update DRM time 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	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

7.5.5 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 Section 12.6 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).

7.5.5.1 update_contact_number_msg() Message

7.5.5.1.1 Description

Using the 1-pass IRD protocol (refer to 7.5.1) the RI sends a update_contact_number_msg() message with a (set of) contact number(s) to the device as specified below:

Table 19: Update contact number message description

update_contact_number_msg( )		
Parameter name	(M)andatory / (O)ptional	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	

message_tag: this parameter identifies the type of the message. Refer to Section C.13 for the value of the message_tag.

protocol_version: this parameter indicates the protocol_version of this message. See Section 7.1.2 for more 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 20: 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 OMA BCAST Service Guide).

Note: refer to C.7 for the value of the error codes.

signature_type_flag: a flag to signal type of signature algorithm used. See Section 7.1.2 for more details.

certificate_version: is a numerical representation of the version of the RI certificate. See Section 7.1.2 for more details.

ri_certificate_counter: this parameter indicates the depth of the RI certificate chain. See Section 7.1.2 for more details.

c_length: this parameter indicates the length in bytes of the ri_certificate.

ri_certificate(): this parameter SHALL be present. See Section 7.1.2 for more details.

ocsp_response_counter: this parameter indicates the depth of the OCSP response chain. See Section 7.1.2 for more details.

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. See Section 7.1.2 for more details. 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 7.5.5.1.3.

signature_block: the signature SHALL enable a single source authenticity check on the message. See Section 7.1.2 for more details.

7.5.5.1.2 Syntax

Table 21: Update contact number 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	
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	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

7.5.5.1.3 Format of the Contact Object

Table 22: 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 23.

Table 23: 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/she 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_bsm_phone_number	The number the user of the device needs to contact to start service provision.	20
0x05	int_home_bsm_phone_number	The number the user of the device needs to contact to start service provision when he/she would call from abroad.	20
0x06	home_bsm_sms_number	The SMS number the user of the device needs to contact to start service provision.	20
0x07	home_bsm_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/she 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		

NOTE: the purpose of the contactdata of contact_types 0x00 and 0x04, 0x01 and 0x05, 0x02 and 0x06, 0x03 and 0x07, is the same. The difference is the source of the information (RI or BSM). The source of information MAY differ based on the Mobile Broadcast Service Provider. However, for each Mobile Broadcast Service Provider, Devices SHOULD NOT receive contactdata with the same purpose from more than one source.

contact_length - This parameter indicates the length in bytes of the contact field. Maximum length of the contacts is specified in Table 23.

UTF-8 [RFC 3629] 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.

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-shop". 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: "shop+subscribe". The string that forms the SMS number is encoded using UTF-8.
URLs	The URL is encoded as hexadecimal, according to [RFC 1738], supporting URLs like: www.shop.com/start. The string that forms the URL is encoded using UTF-8.

7.6 Token Handling

7.6.1 Protocol Overview

The theory of operation (refer to Section C.16) results in the specification of several protocols:

- offline protocols (from device to RI)

Protocol	section	purpose
token request protocol	7.6.2	request to purchase tokens
token reporting protocol	7.6.3	protocol to report the consumption of tokens

- 1-pass protocols (from RI to device)

Protocol	section	purpose
1-pass binary Push Device Registration protocol	0	transmit registration data to device
1-pass binary Inform Registered Device protocol	7.5	inform device via messages.

The protocols interrelate in following way (roundtrip):

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)

7.6.2 Token Request Protocol

When the user of a device wants to obtain tokens, he/she uses the NSD protocol with the token_request action type. (refer to Section 7.4.1.6).

7.6.3 Token Reporting Protocol

When the user of a device is instructed by his/her device to report token consumption, he/she uses the NSD protocol with the token_consumption_message action type in order to send a token consumption report. (refer to Section 7.4.1.5).

7.6.4 token_delivery_response() Message

7.6.4.1 Description

Using the 1-pass IDR protocol (refer to Section 7.5.1) the RI sends a token_delivery_response() message, informing the device of the delivery of new tokens. The message is specified below:

Table 24: Token delivery response message description

token_delivery_response()

Parameter name	(M)andatory / (O)ptional	remark
message_tag	M	not encrypted
protocol_version	M	not encrypted
message_length	M	not encrypted
group_size_flag	M	not encrypted
sign_token_delivery_flag	M	not encrypted
address_mode	M	not encrypted
One	M	not encrypted
rights_issuer_id	M	not encrypted
Status	M	not encrypted
message_seq_number	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
signature_type_flag	O	not encrypted
signature_block	O	not encrypted
MAC	M	not encrypted

message_tag: this parameter identifies the type of the message. Refer to C.13 for the value of the message_tag.

protocol_version: this parameter indicates the protocol_version of this message. See Section 7.1.2 for more details.

message_length: 12-bit field indicating the length in bytes of the message starting immediately after this field.

group_size_flag: in the case of Fixed Subscriber Group sizes, this 1-bit field indicates the group size used. If set to 0 a Subscriber Group size of 256 Devices is used. If set to 1 a Subscriber Group size of 512 Devices is used. In the case of a Flexible Subscriber Group, this flag has no meaning and MUST be ignored.

address_mode: 3-bit field indicating the addressing mode used by this message. The meaning of address_mode is the same as in the BCRO. However for the token_delivery_response message only the addressing of a unique device is allowed. Therefore address_mode MUST contain either the value 0x2 or the value 0x3.

one: 1-bit flag which SHALL have the value 0x1 in this version of the specification. This field MAY have value 0x0 in future versions of the specification

udf: this 40-bit field contains a Unique Device Filter and is used to address a unique device.

In case of Fixed Suscriber Group addressing, the following applies. In the case of a group size of 256 devices, the first 32 bits of the udf contain the **fixed_group_address** field, whilst the last 8 bits contain the **fixed_position_in_group** field. In the case of 512 devices, the first 31 bits contain the **fixed_group_address** field whilst the last 9 bits contain the **fixed_position_in_group** field.

In the case of Flexible Subscriber Group addressing, the udf contains a 40 bit unique address.

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 [DRM-v2].

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 25: 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 it was erroneous and 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 message_seq_number 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 message_seq_number as the message_seq_number 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 message_seq_number 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 message_seq_number as the message_seq_number in this token delivery response message, or from first start-up in case the response_flag was set to 0x0.

Note: refer to C.7 for the value of the error codes.

message_seq_number: if the response_flag equals 0x1, the message_seq_number is the message_seq_number 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 message_seq_number MAY be ignored. See Section 7.1.2 for more details.

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 message_seq_number 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 message_seq_number 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:

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:

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:

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 (minimum size of the id tracking list to be defined by Root of Trust in the compliance rules contract).

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, two'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 key used for the encryption of the token_quantity is the Token Delivery Key.

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 key used for the encryption of the Report Authentication Key is the Token Delivery Key.

signature_type_flag: a flag to signal type of signature algorithm used. See Section 7.1.2 for more details.

signature_block: the signature SHALL enable a single source authenticity check on the message. See Section 7.1.2 for more details.

MAC: this is the authentication code calculated over all bytes before this field in this message using HMAC-SHA1-96 (see [RFC 2104]). 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 C.14. 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 Section C.16.2.

7.6.4.2 Syntax

Table 26: Token delivery response message syntax

fields	length	type
token_delivery_response(){		
/* MAC protected part starts here */ /* signature protected part starts here */		
message_tag	8	bslbf
protocol_version	4	bslbf
message_length	12	uimsbf
group_size_flag	1	bslbf
sign_token_delivery_flag	1	bslbf
reserved for future use	2	bslbf
address_mode	3	uimsbf
one	1	bslbf
udf	40	uimsbf
rights_issuer_id()	160	bslbf
status	8	bslbf
message_seq_number	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
signature_type_flag	1	bslbf
reserved for future use	6	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){		
encrypted_token_quantity	32	bslbf
}		
encrypted_report_authentication_key	128	bslbf
/* encrypted part ends here */		

/* signature protected part ends here */		
/* MAC protected part ends here */		
if(sign_token_delivery_flag == 1) {		
if(signature_type_flag == 0x0) {		
signature_block	1024	bslbf
} else if(signature_type_flag == 0x1) {		
signature_block	2048	bslbf
} else if(signature_type_flag == 0x2) {		
signature_block	4096	bslbf
}		
}		
/* MAC protected part ends here */		
MAC	96	bslbf
}		

Note that all reserved for future use fields SHALL have the value 0 for token delivery response messages created according to this version of the specification.

7.7 Domain Management

7.7.1 Concept of Domains

A domain is a group of Devices that share a common secret, which allows these Devices to share content bound to the domain.

In this specification there are two ways of deploying a domain: as specified in [DRM-v2] (the so called OMA DRM 2.0 Domain) and the equivalent in case there is no interactivity channel (the so called Broadcast Domain).

OMA DRM v2.0 Domains and Broadcast Domains were originally intended to address multiple Devices belonging to the same user, which are registered to the same Domain. However, these Domains can also be used for another object: addressing a very large group of Devices subscribed to the same service or a service bundle for accessing low value content. These large Domains are sometimes referred to as Service Domains.

Using a Broadcast Domain in this mode can provide high bandwidth savings but needs a complete rekeying if only one Device is excluded from the Domain. Using an OMA DRM v2.0 Domains in this mode reduces the number of Rights Objects to be generated to one per Domain. The trade-off for using this mode is that a security incident can affect more devices.

7.7.1.1 OMA DRM 2.0 Domain

OMA DRM 2.0 Domains are the Domains as specified in [DRM-v2]. Only Interactive Devices (or Unconnected Devices that can use a connected Device as a proxy) can belong to an OMA DRM 2.0 Domain, which is defined, limited and managed by the Rights Issuer.

The common secret shared in an OMA DRM 2.0 Domain is called Domain Key. The Domain Key is used to protect the content that is bound to this domain. The Content Encryption Key stored in the Rights Object related to this content is encrypted using the Domain Key. Content and services bound to an OMA DRM 2.0 Domain can only be shared with other Devices in the same Domain, subject to permissions specified by content or service providers.

[DRM-v2] defines ROAP protocols for joining and leaving a Domain. Devices belonging to an OMA DRM 2.0 Domain will adhere to these protocols.

7.7.1.2 Broadcast Domain

Broadcast Domains are the equivalent to the OMA DRM 2.0 Domains in case there is no interaction channel. Devices in a Broadcast Domain share a common group key, which is called Broadcast Domain Key (BDK). The BDK, which was delivered during the registration process or in the domain registration response message, is used to encrypt one or more Service Encryption Keys (SEK) or Program Encryption Keys (PEK). Devices in a Broadcast Domain can share content and

services with any other Device in the same Broadcast Domain, subject to permissions specified by content or service providers.

For Broadcast Domain join and leave operations, offline protocols from Device to Rights Issuer and 1-pass protocols (binary messages) from Rights Issuer to Device are defined.

7.7.2 Domain Joining and Leaving

Interactive devices will adhere to [DRM-v2].

- Interactive devices will therefore use OMA DRM 2.0 domain ID.

Broadcast devices will adhere to the mechanisms as described in this section.

- Broadcast devices will use "shortform_domain_id" a.k.a. SBDF.

Mixed-mode Devices SHALL have the "interoperability" requirement to support both domain 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 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.

7.7.3 Protocol Overview

The theory of operation results in the specification of several protocols:

- offline protocols (from device to RI)

protocol	section	purpose
offline Domain Join Request protocol	7.7.3.1	request to join a domain
offline Domain Leave Request protocol	7.7.3.2	request to leave a domain

- 1-pass protocols (from RI to device)

protocol	section	purpose
1-pass binary Push Device Registration protocol	0	transmit registration data to device
1-pass binary Inform Registered Device protocol	7.5	inform device via messages

The protocols interrelate in following way (roundtrip):

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	domain_update_response() message

(request to leave a domain)	(inform device via messages)
join_domain_msg() (inform device via messages)	offline domain join request, which on it's turn may result in domain_registration_response() as listed above
leave_domain_msg() (inform device via messages)	offline domain leave request, which on it's turn may result in domain_update_response() as listed above

7.7.3.1 Offline Domain Join Request

When the user of a device might want to join a particular domain, he/she uses the NSD protocol with the destined action code range (refer to 7.4.1.3).

7.7.3.2 Offline Domain Leave Request

When the user of a device might want to leave a particular domain, he/she uses the NSD protocol with the destined action code range. (refer to 7.4.1.4).

7.7.4 domain_registration_response() Message

7.7.4.1 Description

Using the 1-pass PDR protocol (see 7.2.2.1) the RI sends a domain_registration_response() message, informing the device of a new domain keyset. The message is specified below:

Table 27: Message description

domain_registration_response()		
Parameter name	(M)andatory / (O)ptional	remark
message_tag	M	global, not encrypted
protocol_version	M	global, not encrypted
longform_udn	M	global, not encrypted
message_seq_number	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

message_tag: this parameter identifies the type of the message. Refer to C.13 for the value of the message_tag.

protocol_version: this parameter indicates the protocol_version of this message. See Section 7.1.2 for more details.

longform_udn(): the long form of the UDN. Refer to Section 7.2.1.2 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 28: 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.

Note: refer to C.7 for the value of the error codes.

message_seq_number: the message_seq_number is the message_seq_number which was present in the request (using the offline NSD protocol) to which this message is a response. See Section 7.1.2 for more details.

time_stamp_flag: binary flag to signal presence or absence of the domain_timestamp_start and domain_timestamp_end parameters. See Section 7.1.2 for more details.

certificate_version: a numerical representation of the version of the RI certificate. See Section 7.1.2 for more details.

ri_certificate_counter: this parameter indicates the depth of the RI certificate chain. See Section 7.1.2 for more details.

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. See Section 7.1.2 for more details.

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. See Section 7.1.2 for more details. 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: 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: please note that this parameter can also be used against replay attacks.)

signature_type_flag: a flag to signal type of signature algorithm used: See Section 7.1.2 for more details.

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: This key is wrapped into the keyset_block. (Refer to 7.7.4.3).

longform_domain_id(): this parameter is also known as the Longform Broadcast Domain Filter (LBDF). Please refer to C.11.2 for the definition. The longform_domain_id() is used for mixed-mode operation. Note: This address is wrapped into the keyset_block. (Refer to 7.7.4.3).

shortform_domain_id: this parameter is also known as the Shortform Broadcast Domain Filter (SBDF). Please refer to C.11.1. An addressing scheme used to filter for messages like BCROs. The shortform_domain_id is used for broadcast mode of operation.

Note: This address is wrapped into the keyset_block. (Refer to 7.7.4.3).

signature_block: the signature SHALL enable a single source authenticity check on the message. See Section 7.1.2 for more details.

7.7.4.1.1 Syntax

Table 29: Domain registration response 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
message_seq_number	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()	1024	bslbf
} else if (signature_type_flag == 0x1)		
sessionkey_block()	2048	bslbf
} else if (signature_type_flag == 0x2)		
sessionkey_block()	4096	bslbf
}		
/* signature protected part ends here */		
if (signature_type_flag == 0x0){		
signature_block	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

7.7.4.2 Stored Domain Context in Device

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 C.11.1.

For mixed-mode operation, devices' domain context SHALL additionally contain:

- Longform Broadcast Domain Filter (LBDF). A.k.a. "longform_domain_id()". Refer to C.11.2.

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 OMA BCAST SERVICE GUIDE 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 GROs, 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 include a valid set of :

- BDK key.
- Shortform Broadcast Domain Filter (SBDF). A.k.a. "shortform_domain_id". Refer to C.11.1.

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 C.11.2.

7.7.4.3 Protection of the (Domain Registration) Keyset

The domain_registration_response() message is split in two parts: device specific (time bound) data and global (not time bound) data.

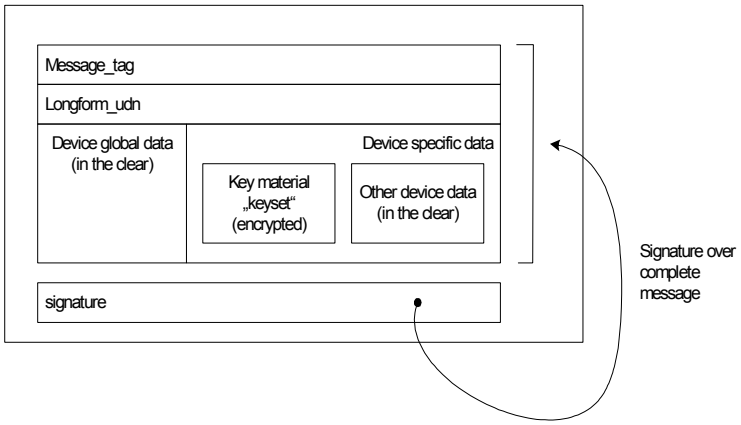


Figure 17: domain_registration_response() message

The device global data SHALL be in the clear. The device specific data contains the keyset for the device.

The RI SHALL use its 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 197] and the Tag Length Format described in Section C.11. 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 Appendix A of [NIST 800-38A]. Note that if the non-padded keyset was already a multiple of 64 bits in length, it is padded with 64 bits. More than one context is allowed up to the RSA blocksize.
3. Encrypt the keyset using [AES_WRAP] using the generated SK as (AES-WRAP style) KEK. This will produce the *keyset_block*.
   - Calculate the part of the keyset_block that would fit into the RSA block (depending on the size of RSA used, be that 1024, 2048 or 4096), including the SK and under implementation rules of the PKCS#1.

4. Encrypt SK plus the keyset_block with the public key of the target device using RSA (1024 or 2048 or 4096) under implementation guidelines of [PKCS#1]. This will produce the *sessionkey_block()*.

5. Concatenate the (non encrypted) parameters that were not used in the key_block and create the message "header" from this. Refer to 7.7.4.1.1 for details. (for reason of completeness: of course the sessionkey_block() and the signature_block are not part of the message header)
6. Concatenate the message "header" and the sessionkey_block(). The result SHALL be hashed under implementation guidelines of [PKCS#1] as specified in Section C.9. This will produce the *signature_input_data*.

7. Sign the signature_input_data with RSA (1024 or 2048 or 4096) using the private key of the RI. The signature SHALL apply to the implementation guidelines of PKCS#1, as specified in C.9. This will produce the *signature_block*.

8. The domain_registration_response() message comprises of the message "header" plus sessionkey_block() and the signature_block.

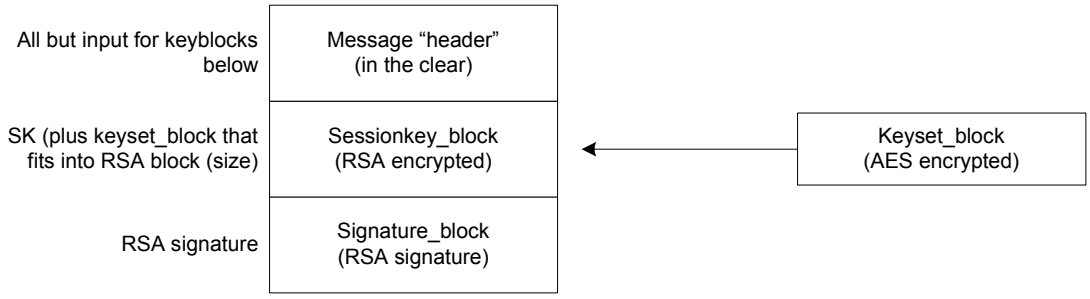


Figure 18: Structure of domain_registration_response() message.

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). Then locate the keyset_block part from the header and (eventual) padding (according to PKCS#1). See Appendix C.12 for the determination of the session key length.

5. Use the SK to decrypt the keyset_block.
    - Allocate the individual keyset_items from the keyset_block according to [AES_WRAP] and the Tag Length Format described in Section C.11.

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.

7.7.5 domain_update_response() Message

7.7.5.1 Description

Using the 1-pass IRD protocol (see 7.5), the RI sends a domain_update_response() message, informing the device that it left a particular domain. The message is specified below:

Table 30: Domain update response message description

domain_update_response()		
Parameter name	(M)andatory / (O)ptional	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
message_seq_number	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

message_tag: this parameter identifies the type of the message. Refer to C.13 for the value of the message_tag.

protocol_version: this parameter indicates the protocol_version of this message. See Section 7.1.2 for more details.

longform_udn(): the long form of the UDN. Refer to Section 7.2.1.2.1 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. In all cases except when the status is NotSupported, the Device SHALL remove the Domain keyset that was associated to the particular Domain.

Table 31: 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.
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

Note: refer to C.7 for the value of the error codes.

message_seq_number: the message_seq_number is the message_seq_number which was present in the request (using the offline NSD protocol) to which this message is a response. See Section 7.1.2 for more details.

certificate_version: is a numerical representation of the version of the RI certificate. See Section 7.1.2 for more details.

ri_certificate_counter: this parameter indicates the depth of the RI certificate chain. See Section 7.1.2 for more details.

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. See Section 7.1.2 for more details.

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. See Section 7.1.2 for more details. 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. See Section 7.1.2 for more details.

signature_block: the signature SHALL enable a single source authenticity check on the message. See Section 7.1.2 for more details.

7.7.5.2 Syntax

Table 32: Domain update response 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
message_seq_number	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	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

7.7.6 join_domain_msg() Message

Using the 1-pass IRD protocol (see 7.5) 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 Section 7.5.2.1, with the only adaptation being that the message_tag is different. Refer to C.13 for the value of the message_tag.

7.7.7 leave_domain_msg() Message

Using the 1-pass IRD protocol (see 7.5), 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 Section 7.5.2.1, with the only adaptations being that:

- the message_tag is different. Refer to C.13 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:

7.7.7.1 Syntax

Table 33: Leave domain 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*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	1024	bslbf
} else if (signature_type_flag == 0x1)		
signature_block	2048	bslbf
} else if (signature_type_flag == 0x2)		
signature_block	4096	bslbf
}		
}		

8. Broadcast Rights

8.1 Broadcast Rights Objects

8.1.1 Goals and Constraints

The delivery of rights objects over a Broadcast network without return channel necessitates some changes to the current ROAP because of the following reasons:

- the XML encoding according to the ROAP schema is not optimised for size
- the current ROAP does not support a subscription group addressing mechanism
- the current ROAP uses signatures based on the RSA PKI scheme that yield large signatures.

This chapter defines a new format for the delivery of authenticated and integrity protected rights objects called Broadcast Rights Objects (BCROs), in which content encryption keys are cryptographically protected with either:

- Broadcast Domain Key (BDK): to address a domain.
- Unique Group Key (UGK): to address the whole Subscriber Group.
- Deduced Encryption Key (DEK): to address a subset of the Subscriber Group.
- Unique Device Key (UDK): to address a unique Device.

The primary design goal is to offer the same or equivalent cryptographic protection on BCROs as is available for ROs obtained via the standard ROAP protocol. This includes authentication, integrity checking and confidentiality of encryption keys.

The secondary design goal is optimisation of message size. This is motivated by the fact that these rights objects may have to be Broadcast repeatedly, as no return path is available to confirm reception. It is assumed that an out-of-band mechanism is available to perform an equivalent of a RORequest, i.e. the initiation of rights object acquisition.

There are two options to integrity protect BCROs. If bandwidth savings are the primary requirement, integrity protection is provided via symmetric key MACs, resulting in savings of approximately 100 octets. However, if origin authentication is a requirement, as is the case with OMA DRM v2.0, BCROs can be digitally signed.

8.1.2 Design Considerations and Decisions

The BCROs are intended to be broadcast to receivers in a well-defined repetitive manner. The particular means of delivery is to be defined in the context of the Broadcast system. It is the intention to support devices without a return channel (next to more capable devices), which implies that BCRO will be transmitted repeatedly to increase the chance of a receiver to capture BCROs addressed to that device.

The key-wrapping technique used in standard ROAP to cryptographically bind a MAC and REK to a device or domain will not be used. Instead the Inferred Encryption Key (IEK), which is derived from the Broadcast Domain Key, the Unique Device Key, the Unique Group Key or the concatenation of the Device Keys depending on the addressing mode, is directly used to protect the Content Encryption Keys in the BCRO. The motivation for this is that an additional REK adds little or no extra security, but adds significant size to a BCRO (as the size of the BCRO would increase by inserting a new field to include the encrypted REK).

Addressing of a unique Device is done using Unique Device Filter (UDF). Addressing of a Device using its Device ID is not supported when using a BCRO.

The broadcast content is protected with a varying encryption key. The encryption keys associated with assets in the BCRO will be applied to decrypt the key stream messages on the key stream layer. Besides decryption, such messages should also be

authenticated. To avoid using the rights issuer authentication key for these frequent messages, the BCRO also carries an authentication key to be used for authenticating key stream messages, see Section 5.5.4 of [BCAST10-ServContProt].

8.1.3 Broadcasting Broadcast Rights Objects

When BCROs are communicated over a broadcast channel, they SHALL be carried in an RI Service, see Chapter 12.

8.2 Format of the Broadcast Rights Object

8.2.1 Format of the OMADRMBroadcastRightsObject() Class

The *OMADRMAsset()*, *OMADRMPermission()* and *OMADRMConstraint()* object correspond in their meaning to their counterparts in OMA-DRM-REL-V2_0. The *OMADRMAction()* object corresponds to the allowed elements in the permissions element from the same specification. The MAC protected BCRO (OMADRMBroadcastRightsObject() class) is mandatory for devices supporting BCROs. The Signature protected BCRO (OMADRMBroadcastRightsObjectSigned() object) is optional for devices supporting BCROs.

Field	Length	Type
OMADRMBroadcastRightsObjectBase() {		
message_tag	8	uimsbf
protocol_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
if (address_mode == 0x0) {		
fixed_group_address	32	uimsbf
} else if (address_mode == 0x1) {		
fixed_group_address	32	uimsbf
if(group_size_flag == 0) {		
fixed_bit_access_mask	256	bslbf
} else {		
fixed_bit_access_mask	512	bslbf
}		
} else if (address_mode & 0x6 == 0x2) {		
udf	40	uimsbf
} else if (address_mode == 0x4) {		
domain_id	38	uimsbf
domain_generation	10	uimsbf
} else if (address_mode == 0x5) {		
flexible_group_address()	variable	OMADRMGroupAddress()
broadcast_encryption_scheme	2	uimsbf
if(flexible_bitmask_present) {		
flexible_bit_access_mask()	variable	OMADRMBitAccessMask()

Field	Length	Type
}		
if(node_number_present) {		
node_number()	variable	OMADRMNodeNumber()
}		
zero_padding_bits	variable	
} else if (address_mode == 0x6) {		
flexible_group_address()	variable	OMADRMGroupAddress()
}		
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()[i]	variable	OMADRMAsset()
}		
if (permissions_flag == 1) {		
number_of_permissions	8	uimsbf
for (i=0; i<number_of_permissions; i++) {		
permission()[i]	variable	OMADRMPermission()
}		
}		
}		

Field	Length	Type
OMADRMBroadcastRightsObject() {		
OMADRMBroadcastRightsObjectBase()	variable	
/* MAC is computed over OMADRMBroadcastRightsObjectBase() */		
MAC	96	bslbf
}		

Field	Length	Type
OMADRMBroadcastRightsObjectSigned() {		
OMADRMBroadcastRightsObjectBase()	variable	
signature_type_flag	2	uimsbf
reserved_for_future_use	6	bslbf
/* signature is computed over all preceding fields. */		
if(signature_type_flag == 0x0) {		
signature	1024	bslbf
} else if(signature_type_flag == 0x1) {		
signature	2048	bslbf
} else if(signature_type_flag == 0x2) {		
signature	4096	bslbf
}		
}		

message_tag: Tag identifying this message as a BCRO. The value for this field is defined in C.13.

protocol_version: 4-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: this field indicates the length of the remainder of the BCRO in bytes starting immediately after this field (excluding locally added information).

group_size_flag: in the case of Fixed Subscriber Group sizes, this 1-bit field indicates the group size used. If set to 0 a Subscriber Group size of 256 Devices is used. If set to 1 a Subscriber Group size of 512 Devices is used.

NOTE: this flag has no meaning in the case of Flexible Subscriber Groups.

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.

Field: address_mode	Description
0x0	addressing of a whole Fixed Subscriber Group
0x1	addressing of a subgroup of devices in a Fixed Subscriber Group using a bitmask size of 256 or 512 bit depending on group_size_flag. This address mode is not used for Flexible Subscriber Groups.
0x2-0x3	addressing of a unique device
0x4	addressing of an OMA domain.
0x5	addressing of a subgroup of devices in a Flexible Subscriber Group. The size of the Subscriber Group is determined at registration. This addressing mode is not used for Fixed Subscriber Groups.
0x6	addressing of a whole Flexible Subscriber Group
0x7	reserved for future use

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.

fixed_group_address: indicates the Fixed Subscriber Group address. Each RI has its own address space.

rights_issuer_id: the ID of the rights issuer. This is the 160-bit SHA1 hash of the DER encoded public key of the RI. See X509PKISHash in OMA.

fixed_bit_access_mask: if the BCRO addresses a subset of a Fixed Subscriber Group with size 256 or 512 (address_mode 0x1) then the fixed_bit_access_mask can be used to define to which receivers in the group this BCRO is addressed to. Receivers not listed in the fixed_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 fixed_bit_access_mask is given by the group_size_flag.

udf: this 40-bit field contains a Unique Device Filter and is used to address a unique device.

In case of Fixed Suscriber Group addressing, the following applies. In the case of a group size of 256 devices, the first 32 bits of the udf contain the **fixed_group_address** field, whilst the last 8 bits contain the **fixed_position_in_group** field. In the case of 512 devices, the first 31 bits contain the **fixed_group_address** field whilst the last 9 bits contain the **fixed_position_in_group** field.

In the case of Flexible Subscriber Group addressing, the udf contains a 40 bit unique address.

flexible_group_address(): indicates the Flexible Subscriber Group address. Each RI has its own address space. See Section 8.2.3.1 for its coding.

broadcast_encryption_scheme: indicates which broadcast encryption scheme is used. See Table 54 in Appendix C.11.1 for more details.

flexible_bitmask_present: this is no dedicated bit in the BCRO, but a boolean value depending on the broadcast_encryption_scheme. See Table 54 in Appendix C.11.1 for details. When TRUE, a flexible_bit_access_mask field follows.

node_number_present: this field is no dedicated bit in the BCRO, but a boolean value depending on the broadcast_encryption_scheme. See Table 54 in Appendix C.11.1 for details. When TRUE a node_number field follows.

flexible_bit_access_mask(): if the BCRO addresses a subset of a Flexible Subscriber Group, then the flexible_bit_access_mask is used to define to which receivers in the group this BCRO is addressed. Receivers not listed in the flexible_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. See Section 8.2.2 for the coding of flexible_bit_access_mask.

node_number(): indicates the position of the node that contains the DEK in the OFT. See Section C.17.1 for details on the numbering of the nodes and Section 8.2.3.3 for the coding of the field.

zero_padding_bits: these (less than 8) bits ensure that the next field is byte aligned.

domain_id: this 38-bit field indicates the domain ID.

domain_generation: this 10 bit field specifies the generation of the domain.

bcro_timestamp: field containing a timestamp at the point of issuing of the BCRO. This 40-bit field contains the time and date of the moment of issuing 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), see also Appendix C.8.

EXAMPLE 1: 93/10/13 12:45:00 is coded as "0xC079124500".

refresh_time: the refresh_time specifies the time when the Device should acquire a new BCRO. It does not specifies when the keys in the BCRO expire. This field is a hint to a receiver to acquire a new BCRO for the content listed in the BCRO before the keys in the BCRO expires. The encoding is similar to that of the bcro_timestamp field.

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 GROs with the same purchase_item_id.

purchase_item_id: 32-bit field specifying the purchase ID this GRO is associated with. The purchase_item_id is used to associate the BCRO with the corresponding Purchase Item in the Service Guide, to enable the Device to display to the user the information that a purchase was completed (see [BCAST10-SG], Section 5.1.2.6). The purchase_item_id field in the BCRO carries the value of the binaryPurchaseItemID field in the corresponding PurchaseItem in the Service Guide (see [BCAST10-SG], Section 5.1.2.6).

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-SHA1-96 (see [RFC 2104]). The MAC is only present in the OMADRMBroadcastRightsObject() object.

The MAC is used to authenticate and check the integrity of the BCRO. The key used to create the MAC is the BCRO authentication key BAK as described in C.14.3.

signature_type_flag: the signature_type_flag is as defined in Section 6.1.3.2.1, reproduced below:

signature_type_flag	Value ($_h$)	remark
RSA 1024	0x0	
RSA 2048	0x1	
RSA 4096	0x2	
reserved for future use	0x3	not used in this version of the specification

signature: the signature is calculated over all bytes before this field with the exception of the first two bytes in the BCRO using RSA-1024, RSA-2048 or RSA-4096. This is only present in the optional OMADRMBroadcastRightsObjectSigned object.

8.2.2 Format of flexible_bit_access_mask()

An addressing bitmask is a string of bits, where each bit corresponds to one particular device. When a device is addressed, its bit in the addressing bit mask is set to 1, otherwise to 0.

The field flexible_bit_access_mask() contains the coded addressing bitmask. The addressing bitmask is split up into subblocks, each of which is coded separately. Depending on the characteristics of the subblock the coding method is chosen. The format of flexible_bit_access_mask() is as follows:

Field	Length	Type
OMADRMBitAccessMask() {		
do {		
subblock_coding_type	**2**	**uimsbf**
if(subblock_coding_type == 0x1) {		
bitmapped_bitmask()	**variable**	**OMADRMBitmappedBitmask()**
} else if(subblock_coding_type == 0x2) {		
block_compressed_bit_access_mask()	**variable**	**OMADRMBlockCompressedBitmask()**

Field	Length	Type
} else if(subblock_coding_type == 0x3) {		
outlier_compressed_bit_access_mask()	**variable**	**OMADRMOutlierCompressedBitmask()**
}		
} while(subblock_coding_type != 0x0)		
}		

subblock_coding_type: 2-bit value indicating how the subblock is coded.

Field: subblock_coding_type	Description
0x0	indicates the end of the bitmask
0x1	the subblock is not compressed, but coded by the method as described in Section 8.2.2.1.
0x2	the subblock is coded using the Block Compression Method as described in Section 8.2.2.2.
0x3	the subblock is coded using the Outlier Compression Method as described in Section 8.2.2.3.

zero_padding_bits: these (less than 8) bits are appended at the end of the flexible_bit_access_mask field to ensure that the subsequent field is byte aligned.

8.2.2.1 Bitmapped Bitmask

The bitmapped_bitmask() field contains a non-compressed subblock. It consists of an indicator for the length of the subblock followed by the subblock. The bitmapped_bitmask() field has the following format:

Field	Length	Type
OMADRMBitmappedBitmask() {		
block_length()	**variable**	**OMADRMBlockLength()**
bit_map	**block_length+1**	**bslbf**
}		

block_length(): indicates the length of the subblock. For a subblock of length *k*, block_length contains the value *k-1*. See Section 8.2.3.5 for more details on the coding of the field block_length.

bit_map: field of block_length()+1 bits, that codes the subblock.

For EXAMPLE, a subblock 0010100101011010 has a length of 16 bits, therefore block_length() contains a value 15 and is coded as **11110** 101 (see Section 8.2.3.5). It is followed by the 16 bits 0010100101011010.

8.2.2.2 Block Compression Method

The Block Compression Method is used when the subblock consists of alternating blocks of ones and zeros. The lengths of these blocks are specified. The block_compressed_bit_access_mask() has the following format:

Field	Length	Type
OMADRMBlockCompressedBitmap() {		
firstbit	**1**	**bslbf**
nole()	**variable**	**OMADRMNole()**
for(i=0; i<nole+1; i++) {		
block_length()[i]	**variable**	**OMADRMBlockLength()**

Field	Length	Type
}		
}		

firstbit: indicates the value of the first bit.

nole() (number of list entries): indicates the number of blocks that follow. If *k* blocks follow, nole contains a value *k-1*. This value is coded as indicated in Section 8.2.3.4.

block_length(): an array that indicates the lengths of the blocks. For a block of length *k*, the corresponding field block_length contains a value *k-1*.

EXAMPLE of coding a subblock using the Block Compression Method:

Let us consider the following 512 bit subblock:

20 x '0', 15 x '1', 2 x '0', 80 x '1', 92 x '0',100 x '1', 203 x '0'.

It starts with a '0', therefore firstbit contains a 0.

There are 7 blocks; therefore nole() contains the value 6 and is coded as **00** 0110 (see Section 8.2.3.4).

Block 1 has a length of 20, therefore its block_length() contains the value 19 and is coded as **111110** 0001, where 0001 is the binary representation of 1=19-18 (see Section 8.2.3.5).

Block 2 has a length of 15; its block_length() is coded as **11110** 100.

Block 3 has a length of 2; its block_length() is coded as **0** 1.

Block 4 has a length of 80; its block_length() is coded as **1111110** 0101101.

Block 5 has a length of 92; its block_length() is coded as **1111110** 0111001.

Block 6 has a length of 100; its block_length() is coded as **1111110** 1000001.

Block 7 has a length of 203, its block_length() is coded as **1111111** 00000000000000000101000.

In this example 98 bits are needed in order to specify the subblock.

8.2.2.3 Outlier Compression Method

The Outlier Compression Method exploits the fact that a subblock can have a sparse amount of '1's or '0's. The outlier_compressed_bit_access_mask() has the following format:

Field	Length	Type
OMADRMOutlierCompressedBitmap() {		
range_flag	**1**	**bslbf**
nole()	**variable**	**OMADRMNole()**
for(i=0; i<nole+2; i++) {		
block_length()[i]	**variable**	**OMADRMBlockLength()**
}		
}		

range_flag: indicates the coding type. When it is equal to 0, we have blocks of '0's separated by single '1's. When it equals 1, we have blocks of '1's separated by single '0's. A bit set to the value that is in a minority is called 'outlier'.

nole() (number of list entries): indicates the number of blocks. The amount of blocks is one more than the amount of outliers (since the coding starts with a block before the first outlier and ends with a block behind the last outlier). If there are *k* blocks, nole contains a value of *k-2*. See Section 8.2.3.4 for the coding of nole.

block_length(): an array that indicates the lengths of the blocks. The first block_length() defines the length of the block in front of the first outlier, whilst the last block_length defines the length of the block behind the last outlier. Notice that a length 0 is coded as 0. See Section 8.2.3.5 for more details on the coding of block_length().

EXAMPLE of coding a subblock using the Outlier Compression Method:

Let us consider the following 512-bit bit_access_mask():

1x'0', 90 x '1', 1 x '0', 80 x '1', 2 x '0', 338 x '1'.

range_flag is equal to 1, since we have blocks of '1's separated by single '0's.

Since there are 5 blocks of '1's separated by 4 single '0's, nole() contains **00** 0011. Notice that 0011 is the binary representation of 3 = 5-2 (see Section 8.2.3.4).

For each of the five blocks of '1's (of which two have length 0), a block_length() field follows:

The first '0' occurs at the first position, so it is considered to be preceded by a block of length 0. Therefore the first block_length() contains 0 and is coded as **0** 0.

The second '0' occurs after 90 '1's, therefore the second block_length() contains the value 90 and is coded as **1111110** 0111000.

The third block_length() contains the value 80 and is coded as **1111110** 0101110.

The third block is followed by two adjacent zeros. For this reason, the fourth block_length() contains the value 0 and is coded as **0** 00.

The fifth block_length() contains the value 338, and is coded as **1111111** 00000000000000010110000.

In this example 68 bits are needed in order to specify the bit_access_mask().

8.2.3 Efficient Coding Tables

Efficient Coding Tables (ECTs) are used to code values in such a way that low values require a small number of bits, whilst extra bits are included for the higher values. In general they have the following form:

Field	Length	Type
OMADRMEfficientCodingTable() {		
indicator	variable	bslbf
translated_value	variable	uimsbf
}		

indicator: bit string of variable length indicating the amount of bits that are used to code the translated_value field.

translated_value: contains the binary representation of the relative position of the value in the value range as can be found in the corresponding Efficient Coding Table. This means that a value *X* is coded as *X-L*, where *L* is the lower bound of the value range that contains *X*.

8.2.3.1 OMADRMGroupAddress()

indicator	amount of bits for value	value range
0	6	0 – 63
10	11	64 – 2 111

110	16	2 112 – 67 647
1110	20	67 648 – 1 116 223
1111	32	1 116 224 - 4 296 083 519

For EXAMPLE, the value 1200 is coded as **10** 10001110000, where 10001110000 is the binary representation of 1136=1200-64.

8.2.3.2 OMADRMPositionInGroup()

indicator	bit length of translated_value	value range
0	9	0 – 511
10	13	512 – 8703
110	18	8704 – 270 847
1110	22	270 848 – 4 465 151
1111	27	4 465 152 – 138 682 879

For EXAMPLE, the value 2000 is coded as **10** 0010111010000, where 0010111010000 is the binary representation of 1488=2000-512.

8.2.3.3 OMADRMNodeNumber()

indicator	bit length of translated_value	value range
0	10	0 – 1 023
10	14	1 024 – 17 407
110	18	17 408 – 279 551
1110	22	279 552 – 4 473 855
1111	27	4 473 856 – 138 691 583

For EXAMPLE, the value 2000 is coded as **10** 00001111010000, where 00001111010000 is the binary representation of 976=2000-1024.

8.2.3.4 OMADRMNole()

indicator	bit length of translated_value	value range
00	4	0 – 15
01	8	16 – 271
10	16	272 – 65 807
11	20	65 808 – 1 114 383

For EXAMPLE, the value 18 is coded as **01** 00000010, where 00000010 is the binary representation of 2=18-16.

8.2.3.5 OMADRMBlockLength()

indicator	bit length of translated_value	value range
0	1	0 –1
10	1	2 – 3
110	1	4 – 5
1110	2	6 – 9
11110	3	10 – 17
111110	4	18 – 33
1111110	7	34 – 161
1111111	22	162 – 4 194 465

For EXAMPLE, the value 16 is coded as **11110** 110, where 110 is the binary representation of 6=16-10.

8.2.4 Format of the OMADRMAsset() Object

Field	Length	Type
OMADRMAsset() {		
BCI	96	bslbf
key_flag	1	uimsbf
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 == 0x0) {		
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. The BCI can be a service_BCI or a program_BCI. These are defined in Section 11.1.1.

reserved_for_future_use: all fields reserved_for_future_use SHALL be set to 0 for this version of the specification.

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 GRO.

asset_type: 2-bit flag indicating the asset type as defined in the table below.

Field: asset_type	Description

0x0	Broadcast stream protected by IPSec, SRTP or ISMACryp as defined in this specification. This asset MAY contain either a PEK or a SEK.
0x1	Downloaded file content as defined by OMA. This asset MAY contain a CEK (Content Encryption Key).
0x2-0x3	reserved

permissions_category_flag: 1-bit flag indicating that a permissions_category field is present in this asset object.

purchase_item_id: 32-bit field specifying the purchase ID of the parent GRO this BCRO is associated with. Refer to Section 8.2.1 for the specification of this relation.

rekeying_period_number: 7-bit field specifying the rekeying_period_number of the parent GRO. The purchase_item_id and rekeying_period_number are used together with the socID and deviceID or domainID to uniquely identify the parent GRO.

permissions_category: for program 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 TKM 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 TKM 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 TKM). 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). TKM 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 then 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 IEK used to decrypt this field depends on the addressing mode of the BCRO The IEK is derived from the UGK, the DEK, the UDK or the BDK. Which key is used for the derivation of the IEK depends on the addressing mode of the BCRO and SHALL be determined using Table 34. The IEK SHALL be derived from the respective keys as described in Sections 10.3.4 and 5.1.

Table 34: Keys used for the derivation of the IEK in different addressing modes

Field: address_mode	Keys used
0x0 (Fixed Subscriber Group addressing / whole group)	UGK (Unique Group Key)
0x1 (Fixed Subscriber Group addressing / subset)	DEK (Deduced Encryption Key: based on fixed_bit_access_mask and SGKs, refer to Section 10.3.4.)
0x2 or 0x3 (unique device)	UDK (Unique device key)
0x4 (OMA Domain)	BDK (Broadcast Domain Key)
0x5 (Flexible Subscriber Group addressing / subset)	DEK (Deduced Encryption Key: based on the broadcast_encryption_scheme and FSGKs, see Table 54 in Appendix C.11 and Section 10.3.4.)
0x6 (Flexible Subscriber Group addressing / whole group)	UGK (Unique Group Key)

encrypted_program_encryption_authentication_key: if key_type is set to 1 then this field contains the encrypted PEAK, the program encryption key (PEK) concatenated with the program authentication seed (PAK). 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 IEK used to decrypt this field is depending on the addressing mode of the BCRO. The IEK is derived from the UGK, the DEK, the UDK or the BDK. Which

key is used for the derivation of the IEK depends on the addressing mode of the BCRO and SHALL be determined using Table 34. The IEK SHALL be derived from the respective keys as described in Sections 10.3.4 and 5.1.

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 IEK used to decrypt this field is depending on the addressing mode of the BCRO. The IEK is derived from the UGK, the DEK, the UDK or the BDK. Which key is used for the derivation of the IEK depends on the addressing mode of the BCRO and SHALL be determined using Table 34. The IEK SHALL be derived from the respective keys as described in Sections 10.3.4 and 5.1.

8.2.5 Format of the OMADRMPermission() Object

Field	Length	Type
OMADRMPermission() {		
number_of_assets	6	uimsbf
constraint_flag	1	uimsbf
actions_flag	1	uimsbf
for (i=0; i<number_of_assets; i++) {		
asset_index	8	uimsbf
}		
if (constraint_flag == 1) {		
OMADRMConstraint()		
}		
if (actions_flag == 1) {		
number_of_actions	8	uimsbf
for (i=0; i<number_of_actions; i++) {		
OMADRMAction()[i]		
}		
}		
}		

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.

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.

action_flag: 1-bit flag. When set to 1, 1 or more actions are contained in 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

8.2.6 Format of the OMADRMAction() Object

Field	Length	Type
OMADRMAction() {		
action_type	7	uimsbf
constraint_flag	1	uimsbf
if (constraint_flag == 1) {		
OMADRMConstraint()		
}		
}		

action_type: 7-bit field specifying the type of action as listed in table below:

Field: action_type	Description

0x00	play action
0x01	display action
0x02	execute action
0x03	print action
0x04	export action
0x05	access action
0x06	save action
0x07-0x7F	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.

8.2.7 Format of the OMADRMConstraint() Object

Field	Length	Type
OMADRMConstraint() {		
number_of_constraints	4	uimsbf
constraint_descriptor_length	12	uimsbf
for (i=0; i<number_of_constraints; i++) {		
OMADRMConstraintDescriptor()[i]		
}		
}		

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.

constraint_tag: tag identifying the specific constraint_descriptor as listed below:

Field: 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	token management constraint
0x08-0xFF	reserved for future use

8.2.7.1 Count Constraint Descriptor

Field	Length	Type
OMADRMCountConstraintDescriptor() {		
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.

8.2.7.2 Timed Count Constraint Descriptor

Field	Length	Type
OMADRMTimedCountConstraintDescriptor() {		
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 SHOULD 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.

8.2.7.3 Date-Time Constraint Descriptor

Field	Length	Type
OMADRMDateTimeConstraintDescriptor() {		
constraint_tag	8	uimsbf
length	8	uimsbf
start_flag	1	uimsbf
end_flag	1	uimsbf
reserved	6	bslbf
if(start_flag == 1) {		
start_date	40	mjdutc
}		
if(end_flag == 1) {		
end_date	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 a 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.

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.

8.2.7.4 Interval Constraint Descriptor

Field	Length	Type
OMADRMIntervalConstraintDescriptor() {		
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 during which the permissions can be exercised over the content. The time_interval period MUST begin when the associated permission is first exercised. The permission can then be exercised any number of times within the time_interval period. The length of the field is given by the length field and SHOULD NOT exceed 32 bit.

8.2.7.5 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.

Field	Length	Type
OMADRMAccumulatedConstraintDescriptor() {		
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 excercised. The period is given in seconds. The length of the field is given by the length field and SHOULD NOT exceed 32 bit.

8.2.7.6 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.

Field	Length	Type
OMADRMIndividualConstraintDescriptor() {		
constraint_tag	**8**	**uimsbf**
length	**8**	**uimsbf**
reserved	**4**	**bslbf**
id_type	**4**	**uimsbf**
individual_id	**8*(length - 1)**	**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 below:

Field: 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.
0x2-0xF	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 the table above.

8.2.7.7 System Constraint Descriptor

Constraint used identify systems to which the content and GROs are allowed to be exported to.

Field	Length	Type
OMADRMSystemConstraintDescriptor() {		
constraint_tag	8	uimsbf

length	8	uimsbf
system_id	64	bslbf
proprietaryinformation	8*length - 64	bslbf
}		

length: the number of bytes following this field.

system_id: the system id of the system the content and GRO can be exported to. This is the HMAC-SHA1-64 encoded hash (using 0 as the key) of the system name as registered with OMNA. The values of the system name URIs and corresponding hashes are registered with OMNA.

proprietaryinformation: this is a string of bytes, containing proprietary parameters for the system. It is outside the scope of this specification to define the syntax and semantics of these bytes. Thus, this is a mechanism to transport proprietary information. This may e.g. be required when exporting to a (possibly non-DRM) system and requiring that no more copies are to be made.

8.2.7.8 Token management constraint descriptor

The token_management_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 token_management_constraint_descriptor indicate how "much" consumption of DRM content requires how many tokens need to be consumed from the token store.

The format of the token_management_constraint_descriptor is specified in the table below.

Field	Length	Type
token_management_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
}		
if (token_constraint_type==0x2) {		
timer	16	uimsbf
}		
}		

length – The number of bytes following this field.

token_constraint_type – If the value of this field equals 0x0 (COUNT) or 0x2 (TIMED_COUNT), the consumption of the DRM content shall 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 0x0 (COUNT) or 0x2 (TIMED_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 0x1 (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 shall 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.

timer – Specifies the number of seconds after which the count state is reduced starting from beginning to render the content in the case that the value of the token_constraint_type field has the value 0x2 (TIMED_COUNT).

8.3 Acquisition of Rights Objects over an Interaction Channel

Devices can acquire rights to access broadcast content by retrieving and processing binary BCROs. In addition, Devices that support an interaction channel next to the broadcast interface can also acquire rights to access broadcast content via the ROAP protocol or the exchange of Domain GROs.

The ROAP protocol via the interaction channel ensures an authenticated delivery of one or more **<protectedRO>** elements. The exchange of Domain GRO's also consists of the exchange of one or more **<protectedRO>** elements.

If a **<protectedRO>** is to convey rights to access broadcast content, then the following applies for all assets that encode rights for broadcast content:

- The **<o-dd:uid>** element in the **<o-ex:context>** element in the **<o-ex:asset>** element MUST hold the BCI (binary content identifier) for the broadcast content referred to by this asset.
- The **<o-ex:digest>** element in the **<o-ex:asset>** SHALL NOT be present.
- The **<xenc:CipherValue>** element contained in the **<ds:KeyInfo>** element MUST hold the AES-wrapped encryption key (SEK or PEK), The RO MUST also contain an additional **<ds:KeyInfo>** element holding the wrapped authentication seed (SAS or PAS).

8.4 Save Permission

The normative statements in this Section 8.4 only apply to the concept of creating super-distributable OMA assets containing a recording of broadcast content, that is suitable for standard OMA DRM v2 devices.

For BCAST Devices with the DRM Profile, recording broadcast content protected using ISMACryp the adapted PDCF described in section 13 MAY be used to record the content directly, together with the STKM key stream, without decryption and local re-encryption. Note that this is not suitable for standard OMA DRM v2 devices, only for BCAST Devices. For further details see the recording section in [BCAST10-ServContProt].

A rights issuer can allow a device to make super-distributable recordings of a broadcast asset by including a save permission in a GRO for that asset. The save permission explicitly allows creating new assets containing a rendering of the broadcast content in permanent storage. The device MUST also have access permission for that broadcast asset in order to create this permanent copy.

The super-distributable recorded assets MUST be in a DCF or PDCF format, and are super-distributable to other devices. The recording device MUST create a new CommonHeaders box for use in each new asset file. The ContentID and RightsIssuerURL are generated from information that is retrieved from the service guide, and the secure DRM time of the device.

If the device does not support secure DRM time, it MUST not allow save permissions.

The context of the broadcast asset (service guide, session description protocol or key stream messages) SHOULD provide at least the Content Identifier, RightsIssuerURL and Content Encryption Key to use when creating the CommonHeaders box and the protected content in each created asset file.

8.4.1 Element <save>

Element	<!ELEMENT o-dd:save (o-ex:constraint?)>
Semantics	The **<save>** element grants the permission to create a permanent representation of some broadcast asset. It contains an optional **<constraint>** element. This **<constraint>** element, if present MUST be combined with any top-level constraint, and both constraints should be satisfied in order for the save permission to be enabled. A rights issuer MUST only include a save permission for broadcast assets. A device MUST ignore save permissions for non-broadcast assets. The save permission only allows creation of OMA DRM v2 compatible DCF or PDCF files. The device SHOULD get from context information (o.a. original assets CommonHeaders box, service guide, session description protocol) relevant information about the broadcast asset to create a CommonHeaders box for use in either a DCF or a PDCF file.

8.4.2 Element <access>

Element	<!ELEMENT oma-dd:access (o-ex:constraint)>
Semantics	The **<access>** element grants the permission to create an immediate[4] rendering of audio or video Content directly from a broadcast, multicast, or unicast stream during its reception. It contains an optional **<constraint>** element. It contains an optional **<constraint>** element. If the **<constraint>** element is specified the DRM Agent MUST grant acces rights according to the **<constraint>** child element and the top-level **<constraint>** element if any. If no child **<constraint>** element is specified the DRM Agent MUST grant access rights according to the top-level **<constraint>** element if any. If neither child nor top-level **<constraint>** element is specified, the DRM Agent MUST grant unlimited access rights. A **<system>** element contained in a **<constraint>** child element to **<access>** is used to specify target system that may be used for creating an immediate rendering of the broadcast, multicast, or unicast stream during its reception. The **<access>** element has the semantics of rendering immediately the broadcast, multicast, or unicast stream somehow into user perceivable form. The DRM Agent MUST NOT grant access according to **<access>** to Content that cannot be rendered in this way. Note that the DRM Agent MUST NOT grant access to stored content, not even stored broadcast, multicast, or unicast streams, based on the **<access>** permission. In order to specify rights for stored content, the **<play>** element MUST be utilized instead (Section 5.4.2 in [DRMREL-v2]). 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.

8.4.3 Construction of the Asset, CommonHeaders and Recording Key

All broadcast content accessed via a service/program GRO, and thus identified with a service_BCI/program_BCI, can be viewed as a continuum of content that belongs to the same OMA group. All content recorded by the device using a combined access+save permission for an asset identified by service_BCI/program_BCI must be accessible to that same device through a play permission associated with the same asset (identified by the service_BCI/program_BCI).

To enable this, and still create uniquely identifiable assets, the OMA group feature is used.

[4] "immediate" here means a time period in the order of one or a few seconds at most.

The way the new asset is created depends on whether the recording device has access to the broadcast content using a service GRO (containing a SEK, associated with a service_BCI) or a program GRO (containing a PEK, associated with a program_BCI).

8.4.3.1 Recording Broadcast Content

The device makes a recording of broadcast content that is accessed through an asset, that identifies the Broadcast Content Identifier (service_BCI or program_BCI), and which is associated with either a Service Encryption Key or a Program Encryption Key. In the following sections, **BCI** refers to the broadcast content identifier of that asset, and **KEY** refers to either the SEK or the PEK, whichever is associated with that asset.

	Asset contains program_BCI and PEK	Asset contains service_BCI and SEK
$\mathbf{BCI}_{\mathbf{service/program}}$	program_BCI	service_BCI
$\mathbf{KEY}_{\mathbf{sek/pek}}$	PEK	SEK

The device MUST include a GroupID box in the new asset that is to hold the recorded content. The GroupID in that box is derived from BCIservice/program, and MUST be as specified in Table 35.

The content of the created asset MUST be encrypted with a key CIEK. The GroupKey stored in the box MUST be the key CIEK that is encrypted with KEYsek/pek.

The EncryptionAlgorithm field in the GroupID box MUST identify the AES-CBC mode algorithm. The recording device MUST generate a suitable CIEK value at random. This allows superdistribution to be achieved without distribution of the SEK/PEK in the RO which gives access to the superdistributed content. The initialisation vector MUST be randomly generated by the device:

CIEK := random 128-bit AES key or KEYsek/pek

IV := random 128 bit number

GroupKey := IV || AES-CBC{ KEYsek/pek }(CIEK)

Table 35: Fields in the GroupID box

Field	Contents
GKEncryptionMethod	MUST be AES-CBC.
GroupID	Shall equal the following string in case of a service: "cid:service_BCI@" || base64Binary(service_BCI) Shall equal the following string in case of a program: "cid:program_BCI@" || base64Binary(program_BCI) NOTE: The double quote characters above are string identifiers and are not put in the GroupID.
GroupKey	Contains the result of applying the encryption algorithm defined by GKEncryptionMethod to the CIEK key as plaintext, using KEYsek/pek as encryption key and a randomly selected initialization vector. This initialization vector MUST be prefixed to the resulting ciphertext.

The CommonHeaders box MUST contain a unique ContentID, as well as a proper RightsIssuerURL.

Table 36: CommonHeaders box fields

Field	Contents
`EncryptionMethod`	Determined by the recording device.
`PaddingScheme`	Determined by the recording device.
`PlaintextLength`	Determined by the length of the recorded asset, calculated by the recording device.

ContentIDLength **ContentID[]**	MUST equal: GroupID + base64(recording timestamp) Note that GroupID is defined in Table 35.
RightsIssuerURLLength **RightsIssuerURL[]**	MUST equal: RightsIssuerURL + "?rib=" + base64(recording information block) Where the RightsIssuerURL is retrieved from the service guide, using its association with the service_CID (in case the asset holds a service_BCI) or the program_CID (in case the asset holds a program_BCI). The recording information block holds the BCIservice/program, the recording timestamp, the CIEK (but salted and encrypted with the KEYsek/pek) and an integrity protection.
TextualHeadersLength **TextualHeaders[]**	Determined by context information (original asset, service guide, session description protocol).
ExtendedHeaders[]	Contains the GroupID box.

In the definition of these fields, the base64() operation is defined by [RFC2045], the '+' denotes concatenation, the recording timestamp is defined by Section 8.4.3.2 and the recording information block is defined in Section 8.4.3.3.

Based on the values of the 'rib', the rights issuer can determine and verify the integrity of the recording information, including the CIEK. This then allows the rights issuer to issue GROs to the saved asset or to the whole group of recorded content (that share the same GroupId).

8.4.3.2 Recording Timestamp

The representation with which the device should represent the date and time of the start or the end of the recording is defined by two timestamps that are NTP timestamps as specified by [RFC1305], but with the fractional seconds part truncated to leave only the 4 most significant bits.

The first timestamp indicates the date and time of the start of the recording, whereas the second timestamp indicates the end of the recording.

Field	Length	Type
OMADRMRecordingTimestamp() {		
startDateAndTime	36	NTP timestamp, see below
endDateAndTime	36	NTP timestamp, see below
}		

Example:

The recording timestamp:

(msb) 11000110100110011101010001010110 0001

11000110100110100000000101011100 0111 (lsb)

corresponds to the recording start time and date NTP timestamp:

11000110100110011101010001010110 00010000000000000000000000000000

which equals 3331970134.0625 seconds after January 1st, 1900, 00:00 UTC, or Jan 8th, 2005, 11:15:34.0625 UTC

and the recording end time and date NTP timestamp:

11000110100110100000000101011100 01110000000000000000000000000000

which equals 3331981660.4375 after January 1st, 1900, 00:00 UTC, or Jan 8th, 2005, 14:27:40.4375 UTC

Note that the whole seconds part of the NTP timestamp format is 32 bits, and will roll-over on February 6, 2036 06:28:16 UTC. For that reason, devices and rights issuers SHALL interpret NTP timestamps of which the whole seconds part has a most significant bit of 0, as signalling a date and time in the epoch 2036-2172.

8.4.3.3 Recording Information Block

The RightsIssuerURL holds a 'rib' parameter, which equals the base64 encoded recording information block defined in this section.

Field	Length	Type
OMADRMRecordingInformationBlockBase() {		
BCI	96	bslbf
OMADRMRecordingTimestamp()timestamp()	72	OMADRMRecordingTimestamp()
salt	128	bslbf
salted_key	128	bslbf
}		

Field	Length	Type
OMADRMRecordingInformationBlock() {		
OMADRMRecordingInformationBlockBase()	**424**	
MAC	**96**	**bslbf**
}		

Field	Length	Type
OMADRMRecordingInformationBlockSigned() {		
OMADRMRecordingInformationBlockBase()	**424**	
signature_type_flag	**2**	**uimsbf**
reserved_for_future_use	**6**	**bslbf**
/* signature is computed over all preceding fields.		
***/**		
if(signature_type_flag == 0x0) {		
signature	**1024**	**bslbf**
} else if (signature_type_flag == 0x01) {		
signature	**2048**	**bslbf**
} else if (signature_type_flag == 0x02) {		
signature	**4096**	**bslbf**
}		
}		

BCI**:** contains the BCIservice/program (service_BCI or program_BCI, depending on the asset to which the save permission is applied).

timestamp(): this contains the recording start date and time and the recording end date and time.

salt: this is a random 128 bit number, generated by the recording device which is used to salt the CIEK before it is encrypted.

salted_key: this field contains the result of encrypting the salted CIEK with KEYsek/pek:

salted_key := AES-ECB{ KEYsek/pek } (CIEK **xor** salt)

Note: AES-ECB is used in this case to avoid the padding overhead of AES-CBC as used in Section 8.4.3.1.

MAC: this is the authentication code calculated over all bytes before this field in the OMADRMRecordingInformationBlock using HMAC-SHA1-96 (see [RFC 2104]). The MAC is used check the integrity of the recording information. The key used to create the MAC is KEYsek/pek, depending on the asset to which the save permission is applied.

OMADRMRecordingInformationBlockSigned is only applicable when the sign_bcro_flag is turned on in the device_registration_response message. As such this class is OPTIONAL.

8.4.3.4 Access to Recorded Assets

Recorded assets have a GroupID box that defines them as being part of a group of assets that are protected with the same key, and that share a common GroupId. By making sure that the recording device uses its access permission content id as the GroupId of all the recorded assets recorded using that access permission, play permissions can be issued with the same content id as the access permission; and it will apply to all recorded material that was recorded using that access permission.

On the other hand, the ContentIDs of the generated assets are unique (by qualifying the base content id with the recording timestamp) as required by the OMA DCF specification, and other devices can use the RightsIssuerURL to contact the original rights issuer to acquire play rights for that content. The rights issuer is free to provide group rights or individual asset rights. A group right would contain the GroupId, whereas an individual right would refer to the exact ContentID (as can be retrieved from the RightsIssuerURL).

8.4.4 Recording Concept

The concept of controlled recording is illustrated in the following figure. A rights issuer has issued a GRO to device A. This gives device A the right to access a certain broadcast asset, as well as the right to create a super-distributable copy of (part) of that broadcast asset in a new asset. Another device B may receive a copy of this new content file and contacts the rights issuer to acquire (play) rights for this content.

Note: a similar mechanism applies to BCAST Devices using the DRM Profile when using the adapted PDCF for recording content protected using ISMACryp (see Section 13).

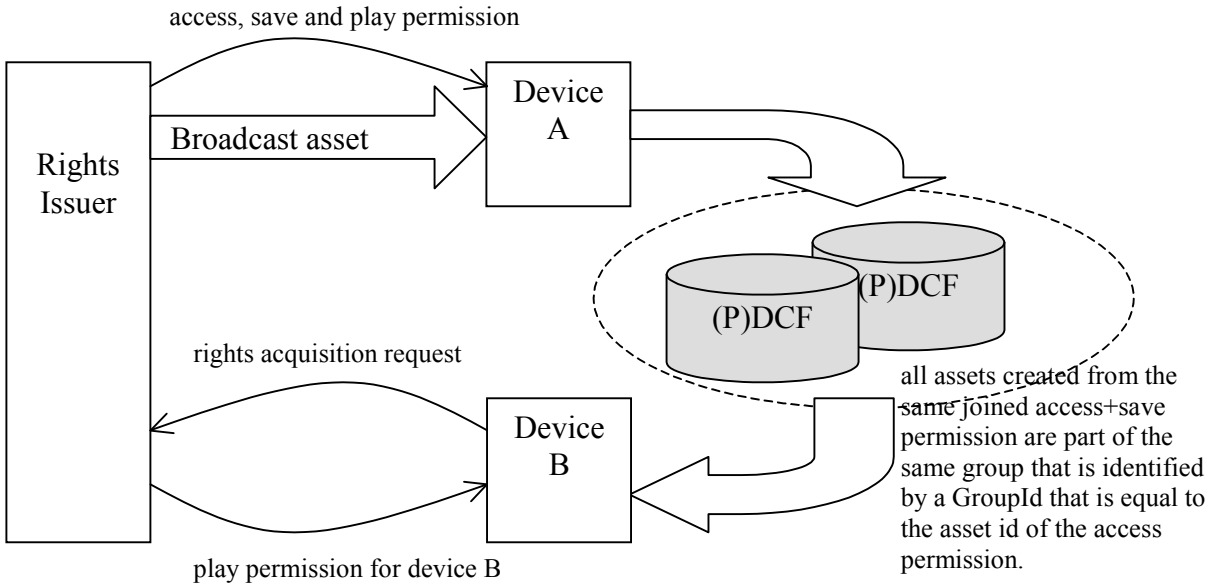


Figure 19: Recording and super-distributing the recorded asset

In Figure 19, the content of each (P)DCF file recorded from a service or program was decrypted from the IPDC streaming as plaintext, then re-encrypted with CIEK into the file. As (P)DCF fields, GroupID equals to BCI, both GroupKey and

RightsIssuerURL contain CIEK encrypted by SEK or PEK, and both RightsIssuerURL and ContentID contain recording start and end times. The server and path components of RightsIssuerURL, to be used by http protocol for locating the ROAP server in the Internet, equal to the URL originally found in the service guide.

Device A will still use its original GRO for accessing all the recordings made from the service or program.

Device B will receive from the RI a new RO, bound either to the GroupID or ContentID of the recording, with the PLAY permission for the (P)DCF file of the recorded content, encrypted by the CIEK. The start and end times of the recording enable the RI to calculate the price of the RO for playing it. In case the new RO is bound to the ContentID, the RI will get the CIEK key for the new RO from the RightsIssuerURL of the acquisition request, while in case of a GroupID bound RO the original SEK or PEK (for decrypting CIEK from the GroupKey) is included in the new RO.

The RI is a standard OMA DRM 2.0 Rights Issuer, which is able to decrypt the rib parameter included in the RightsIssuerURL, extract the CIEK and put it into the Rights Object.

9. Token Management

This section defines extensions to the OMA DRM 2.0 REL DTD and ROAP schemas added to accommodate the management of tokens.

9.1 Additions to the OMA DRM 2.0 REL

This section defines additions to the OMA DRM 2.0 REL DTD to indicate when and how tokens are consumed. The new *token-based* constraint defines that usage of the corresponding DRM content involves consumption of tokens. A device can receive tokens from multiple RIs and use them to consume DRM content whose usage is defined as token-based in the RO associated with the DRM content.

The **<token-based>** element can contain one of the following three elements which define what kind of stateful consumption will be governed by token availability:

- **<token-constraint-count>**: Defines that tokens will be consumed every time the content is rendered.
- **<token-constraint-timed-count>**: Defines that tokens will be consumed every time the content is rendered for more than the number of seconds defined in the timer attribute.
- **<token-constraint-accumulated>**: Defines that tokens will be consumed every time the content is rendered for more than a particular amount of accumulated time, e.g. every 30 minutes of usage (which may not be contiguous).

Each of the above elements will themselves contain the following elements which define how many tokens will be consumed in a particular usage scenario.

- *token-unit*: The unit of the specified constraint which corresponds to tokens being decremented, e.g. a single count or 30 minutes of time.
- *tokens-consumed*: Tokens consumed per *token-unit*, e.g. 3 tokens consumed for every count.

The example of the usage of this constraint in Figure 20 instructs the DRM agent to consume two tokens every time that the corresponding content item is played once.

```
<o-dd:play>
  <o-ex:constraint>
    <oma-dd:token-based>
      <oma-dd:token-constraint-count>
        <oma-dd:token-unit>1</oma-dd:token-unit>
        <oma-dd:tokens-consumed>2</oma-dd:tokens-consumed>
      </oma-dd:token-constraint-count>
    </oma-dd:token-based>
  </o-ex:constraint>
</o-dd:play>
```

Figure 20: Example usage of token-based constraint

9.1.1 Element <token-based>

Element	<!ELEMENT oma-dd:token-based (oma-dd:token-constraint-count | oma-dd:token-constraint-timed-count | oma-dd:token-constraint-accumulated)>

Semantics	The **<token-based>** element specifies that tokens are consumed when the DRM content to which the constraint applies is used in a certain way. It contains one of the **<oma-dd:token-constraint-count>**, **<oma-dd:token-constraint-timed-count>** or **<oma-dd:token-constraint-accumulated>** elements depending on what kind of stateful consumption will be governed by token availability.

9.1.2 Element <token-constraint-count>

Element	<!ELEMENT oma-dd:token-constraint-count (oma-dd:token-unit,oma-dd:tokens-consumed)>
Semantics	The **<token-constraint-count>** element indicates that every time the number of counts specified in the enclosed **<token-unit>**-element is consumed, the token store is decremented by the number of tokens in the enclosed **<tokens-consumed>** element. For example, if the **<token-unit>** element contains "1" and the **<tokens-consumed>** element contains "2", then each time the permission is exercised the token store is decremented by 2 tokens.

9.1.3 Element <token-constraint-timed-count>

Element	<!ELEMENT oma-dd:token-constraint-timed-count (oma-dd:token-unit,oma-dd:tokens-consumed)>
Semantics	The **<token-constraint-timed-count>** element indicates that every time the number of timed counts specified in the enclosed **<token-unit>** element is consumed, the token store is decremented by the number of tokens in the enclosed **<tokens-consumed>** element. For example, if the **<token-unit>** element contains "1" and the **<tokens-consumed>** element contains "2", and the **<timer>** attribute contains 10 seconds, then each time the permission is exercised for more than 10 seconds, the token store is decremented by 2 tokens.

9.1.4 Element <token-constraint-accumulated>

Element	<!ELEMENT oma-dd:token-constraint-accumulated (oma-dd:token-unit,oma-dd:tokens-consumed)>
Semantics	The **<token-constraint-accumulated>** element specifies the maximum period of time during which the permission can be exercised over the DRM Content before the token store is decremented by the number of tokens in the **<tokens-consumed>** element. For example, if the **<token-unit>** element specifies 900 seconds and the **<tokens-consumed>** element contains "1", then each time the permission is exercised for 900 seconds (since the last token decrement) the token store is decremented by 1 token.

9.1.5 Element <token-unit>

Attribute	<!ELEMENT oma-dd:token-unit (#PCDATA)>

Semantics	The format of the **<token-unit>** element depends on the enclosing element. If the enclosing element is **<token-constraint-count>** or **<token-constraint-timed-count>**, the corresponding **<token-unit>** value specifies the number of times permission may be granted over an asset in order for the corresponding number of tokens in the **<tokens-consumed>** element to be consumed. This must be a positive integer value. If the enclosing element is **<token-constraint-accumulated>**, then the **<token-unit>** value specifies the number of seconds a permission may be granted over an asset in order for the corresponding number of tokens in the **<tokens-consumed>** element to be consumed. The lexical representation of this value MUST use the restricted accumulated format PnDTnHnMnS or any reduced precision and truncated representation version thereof as specified in [XMLSchema]. For example, P15DT10H30M20S represents an accumulated of 15 days, 10 hours, 30 minutes and 20 seconds. The specified period SHOULD be greater than zero. If the specified period is equal to zero, then the permission MUST NOT be granted. [XMLSchema] allows the number of seconds in the period to include decimal digits to arbitrary precision. However, to ensure interoperability, ROs MUST NOT contain fractional seconds in the period.

9.1.6 Element <tokens-consumed>

Attribute	<!ELEMENT oma-dd:tokens-consumed (#PCDATA) >
Semantics	The **<tokens-consumed>** element contains a positive integer value. It specifies the number of tokens by which the token store should be decremented when a single token unit (as specified in the **<token-unit>** element) is consumed when exercising the permission to which the **<oma-dd:tokenbased>** constraint is attached. For example, if the **<token-unit>** indicates 900 seconds and **<tokens-consumed>** contains 1, then the token store will be decremented each time the DRM content is played for a total of 900 accumulated seconds. If the DRM agent detects that no tokens exist for this RI, or a number of tokens is less than that contained in the **<tokens-consumed>** element, the DRM Agent MUST NOT grant the corresponding permission to the DRM Content.

9.1.7 Element <permission>

Element	<!ELEMENT o-ex:permission (o-ex:constraint?, o-ex:asset*, o-dd:play?, o-dd:display?, o-dd:execute?, o-dd:print?, o-dd:save?, oma-dd:export?, oma-dd:access?)>

Semantics	The **<permission>** element contains an optional **<constraint>** element, zero or more **<asset>** elements and a set of optional permissions specifying the rights over a piece of Content, such as **<play>**, **<display>**, **<execute>**, **<print>**, **<save>**, **<export>**, and **<access>** permission elements. The **<constraint>** element is the top-level constraint. As a sibling element to other permission elements such as **<play>**, **<display>** it applies to all sibling permission elements inside the same **<permission>** element. The DRM Agent MUST honor the top level constraint in addition to honoring possible constraints specified as a child element to a permission element, e.g., **<play>**, when granting access to content according to such a permission. The **<asset>** elements specified within the **<permission>** element enable expression linking allowing its sibling permission elements in the same **<permission>** element to apply to DRM Content referenced by **<asset>** elements contained in an **<agreement>** element (i.e., outside a **<permission>** element). The link is established through the use of the "id" and "idref" attributes specified in Sections 5.2.2.1 and 5.2.2.2 in [DRMREL-v2]. Note that the DRM Agent MUST respect both, constraints specified as child elements to a permission element and those specified as top-level constraints in the same Rights Object. I.e., the stricter of two constraints of the same type prevails for a given permission element. Of course, Rights Objects with contradictory constraints should not be issued in the first place. When there is a top-level constraint that is otherwise not allowed as a child constraint to a permission, e.g., **<count>** and **<export mode="move">**, the child constraint takes precedence over the top-level constraint as applied to this permission. For example, in the move scenario, Content and Rights Object would be moved, and the **<count>** constraint would accordingly be removed, too. A DRM Agent MUST grant access to DRM Content referenced by an **<asset>** element in the agreement model according to permissions specified inside a **<permission>** element that is as sibling elements to an **<asset>** element in the permission model, where the **<asset>** element referencing the DRM Content and the **<asset>** element inside the **<permission>** element are linked by matching "id" and "idref" attributes. If no **<asset>** element is present in a permission element such as **<play>**, then the permission applies to all **<asset>** sibling elements in the same Rights Object. The **<export>** permission is associated with all of the DRM Content referenced by **<asset>** elements within the same Rights Object. A single Rights Object has at most one **<export>** element within a given **<permission>** element.

9.1.8 Attribute "timer"

Attribute	<!ATTLIST oma-dd:token-timed-count timer CDATA #REQUIRED >
Semantics	The attribute contains a positive integer value. It specifies the number of seconds after which the count state is reduced starting from beginning to render the Content. For example, if the timer value is set to "30" (without the quotes) the count state is decremented after the content has been rendered for 30 seconds. When the number of counts specified in the corresponding **<token-unit>** element is consumed, the token store is decremented by the number of tokens in the **<tokens-consumed>** element. For example, if the **<token-unit>** element contains "1", the timer attribute specifies 30 seconds, and the **<tokens-consumed>** element contains 2, then each time the permission is exercised for at least 30 seconds, the token store is decremented by 2 tokens.

9.2 Extensions to ROAP to Issue Tokens

The OMA DRM 2.0 ROAP-protocol is extended in this section to allow tokens to be delivered to a device. Either a 1-pass or 2-pass token delivery protocol can be used.

The 2-pass token delivery protocol is illustrated in Figure 21. The first element of the 2-pass ROAP extension is a ROAP trigger which prompts the device to send a ROAP-TokenRequest to the RI. The RI then responds with a ROAP-TokenDeliveryResponse.

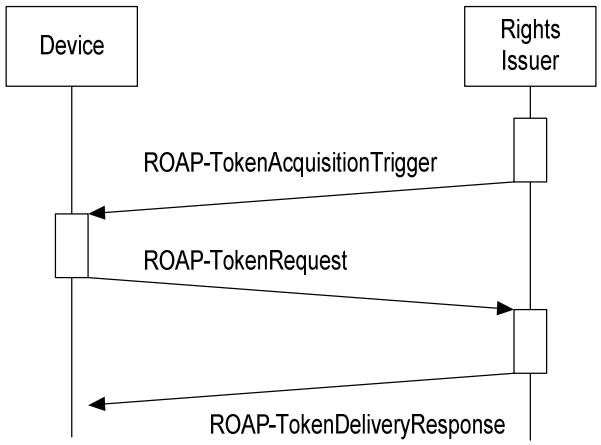


Figure 21: the 2-pass token delivery protocol

In the 1-pass token delivery protocol, which is illustrated in Figure 22, only a ROAP-TokenDeliveryResponse is delivered by the RI to the device.

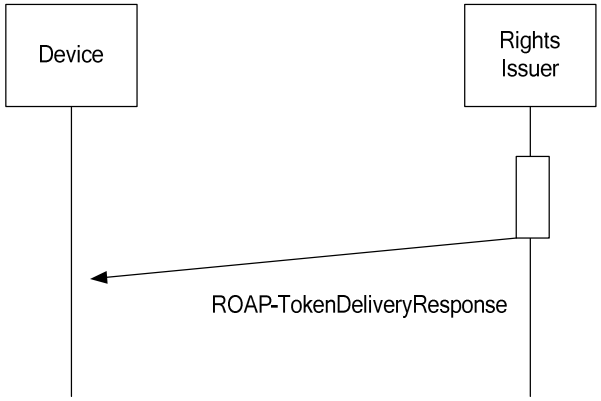


Figure 22: the 1-pass token delivery protocol

9.2.1 ROAP-TokenAcquisitionTrigger

The full extensions to the ROAP schema for triggers required to add this trigger are shown in Figure 23. See also Appendix C.3.4 for a definition of the extensibility mechanism upon which the new trigger is built.

The XML representation of the token acquisition trigger is defined by the TokenAcquisitionTrigger type below and also validates against the ExtendedRoapTrigger type defined in Appendix C.3.4. It SHALL be signalled as an **<extendedTrigger>** element with the **type** attribute set to "**tokenAcquisition**". The elements in the token acquisition trigger have the following meaning:

- The RI ID MUST uniquely identify the Rights Issuer.

- If present, the **<riAlias>** element SHALL be processed according to [DRM-v2].
- The **<nonce>** element provides a way to couple ROAP triggers with ROAP requests.
- The DRM Agent MUST use the URL specified by the **<roapURL>** element when initiating the ROAP transaction. When the **<roapTrigger>** element carries an **<extendedTrigger>** element with the **type** attribute set to "**tokenAcquisition**", the PDU MUST be a ROAP-TokenAcquisitionRequest PDU.
- The Token Delivery ID identifies the token request in a similar way to the way the RO ID identifies an RO.
- If the trigger is signed, the **<ds:Reference>** element of the **<ds:SignedInfo>** child element of the trigger **<signature>** shall reference a ROAPTrigger element by using the same value for the URI attribute as the value for the ROAP trigger element's id attribute.

The token acquisition trigger is defined using the complex type **roap:TokenAcquisitionTrigger** which is shown below and added to the ROAP schema.

```
<complexType name="BasicRoapTrigger">
  <sequence>
    <element name="riID" type="roap:Identifier"/>
    <element name="riAlias" type="string" minOccurs="0"/>
    <element name="nonce" type="roap:Nonce" minOccurs="0"/>
    <element name="roapURL" type="anyURI"/>
  </sequence>
  <attribute name="id" type="ID"/>
</complexType>

<complexType name="DomainTrigger">
  <complexContent>
   <extension base="roap:BasicRoapTrigger">
    <sequence>
      <element name="domainID" type="roap:DomainIdentifier" minOccurs="0"/>
      <element name="domainAlias" type="string" minOccurs="0"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>

<complexType name="ROAcquisitionTrigger">
  <complexContent>
   <extension base="roap:DomainTrigger">
    <sequence>
      <sequence maxOccurs="unbounded">
        <element name="roID" type="ID"/>
        <element name="roAlias" type="string" minOccurs="0"/>
        <element name="contentID" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </sequence>
   </extension>
  </complexContent>
</complexType>

<complexType name="ExtendedRoapTrigger">
   <complexContent>
    <extension base="roap:BasicRoapTrigger">
      <sequence>
        <any  minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
      </sequence>
```

```
      <attribute name="type" type="string" use=”required”/>
    </extension>
  </complexContent>
</complexType>

<complexType name="TokenAcquisitionTrigger">
  <complexContent>
    <extension base="roap:BasicRoapTrigger">
      <sequence>
        <element name="tokenDeliveryID" type="ID"/>
        <any  minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
      </sequence>
      <attribute name="type" type="string" use=”required” fixed=”tokenAcquisition”/>
    </extension>
  </complexContent>
</complexType>

<!-- ROAP trigger -->
<element name="roapTrigger" type="roap:RoapTrigger"/>
<complexType name="RoapTrigger">
 <annotation>
  <documentation xml:lang="en">
   Message used to trigger the device to initiate a Rights Object Acquisition Protocol.
  </documentation>
 </annotation>
 <sequence>
  <choice>
   <element name="registrationRequest" type="roap:RegistrationRequestTrigger"/>
   <element name="roAcquisition" type="roap:ROAcquisitionTrigger"/>
   <element name="joinDomain" type="roap:DomainTrigger"/>
   <element name="leaveDomain" type="roap:DomainTrigger"/>
   <element name="extendedTrigger" type="roap:ExtendedTrigger"/>
  </choice>
  <element name="signature" type="ds:SignatureType" minOccurs="0"/>
  <element name="encKey" type="xenc:EncryptedKeyType" minOccurs="0"/>
 </sequence>
 <attribute name="version" type="roap:Version"/>
 <attribute name="proxy" type="boolean"/>
</complexType>
```

Figure 23: Token acquisition trigger complex type

9.2.2 ROAP-TokenRequest

A device can create a token request from the device to a rights issuer. This is an extension of the existing ROAP request type.

ROAP-TokenRequest	
Parameter	**Mandatory/Optional**
Device ID	M
RI ID	M
Device Nonce	M
Token Delivery ID	M

Certificate Chain	M
Extensions	O
Signature	O

Figure 24: Token request message description

Device ID: identifies the requesting Device.

RI ID: identifies the authorizing RI.

Device Nonce: a nonce chosen by the Device.

Token Delivery ID: identifies the tokens to be issued to this device in a similar fashion to the way an RO ID identifies a RO. The Token Delivery ID MUST uniquely identify the tokens to be issued in the TokenDeliveryResponse.

Certificate Chain: this parameter is sent unless it is indicated in the RI Context that this RI has stored necessary Device certificate information. When present, the parameter value SHALL be as described for the Certificate Chain parameter in the ROAP-RegistrationRequest message.

Extensions: the following extensions are defined for the ROAP-TokenRequest message:

- **Peer Key Identifier:** an identifier for an RI public key stored in the Device. If the identifier matches the RI's current public key, or if the extension is empty, it means the Device has already stored the RI ID and the corresponding RI certificate chain, and the RI need not send down its certificate chain in its response message.
- **No OCSP Response:** presence of this extension indicates to the RI that there is no need to send any OCSP responses since the Device has cached a complete set of valid OCSP responses for this RI.
- **OCSP Responder Key Identifier:** this extension identifies an OCSP responder key stored in the Device. If the identifier matches the key in the certificate used by the RI's OCSP responder, the RI MAY remove the OCSP Responder certificate chain from the OCSP response before providing the OCSP response to the Device.

The Device MUST send the Peer Key Identifier extension if, and only if, it has stored the RI public key. The Device MUST send the No OCSP Response extension if, and only if, it has a complete set of valid OCSP responses for the RI certificate chain. The Device MUST send the OCSP Responder Key Identifier extension if, and only if, it has stored an OCSP Responder key for this RI.

Signature: a signature on this message (besides the *Signature* element itself). The signature method is as follows:

- The message except the *Signature* element is canonicalized using the exclusive canonicalization method defined in [XC14N].
- The result of the canonicalization, *d*, is considered as input to the signature operation.
- The signature is calculated on *d* in accordance with the rules of the negotiated signature scheme

The RI MUST verify the signature on the ROAP-TokenRequest message.

9.2.2.1 Message Syntax

The **<tokenRequest>** element specifies the ROAP-TokenRequest message. It has complex type roap:TokenRequest, which extends the basic roap:Request type.

```
<element name="tokenRequest" type="roap:TokenRequest"/>

<complexType name="TokenRequest">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to request tokens
```

```
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Request">
      <sequence>
        <element name="deviceID" type="roap:Identifier"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="nonce" type="roap:Nonce"/>
        <element name ="tokenDeliveryID" type="ID"/>
        <element name="certificateChain" type="roap:CertificateChain" minOccurs="0"/>
        <element name="extensions" type="roap:Extensions" minOccurs="0"/>
        <element name="signature" type="base64Binary"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

9.2.3 ROAP-TokenDeliveryResponse

The ROAP-TokenDeliveryResponse is returned to the device by the RI in response to a ROAP-TokenRequest, or can also be used in the 1-pass version without any preceding messages.

Parameter	ROAP-TokenDeliveryResponse		
	2-pass Status = Success	*2-pass Status ≠ Success*	*1-pass*
Status	M	M	M
Device ID	M	-	M
RI ID	M	-	M
Token Delivery ID	M	-	M
Device Nonce	M	-	-
Token Quantity	M	-	M
Token Reporting URL	O	-	O
Latest Token Consumption Time	O	-	O
Earliest Reporting Time	O	-	O
Latest Reporting Time	O	-	O
Certificate Chain	O	-	O
OCSP Response	O	-	M
Extensions	O	-	O
Signature	M	-	M

Figure 25: Token delivery response

Status: indicates if the request was successfully handled or not. In the latter case, an error code as specified in OMA DRM 2.0 status codes are sent. Some additional status values have been defined to support token management.

Device ID: identifies the requesting Device. The value returned here MUST equal the Device ID sent by the Device in the ROAP-TokenRequest message that triggered this response in the 2-pass ROAP. In the 1-pass ROAP, the RI selects the

Device ID of the recipient Device. If the Device ID is incorrect, the *ROAP-TokenDeliveryResponse* processing will fail and the Device MUST discard the received *ROAP-TokenDeliveryResponse* PDU.

RI ID: identifies the RI. In the 2-pass protocol, the value MUST equal the RI ID sent by the Device in the preceding ROAP-TokenRequest message.

Token Delivery ID: identifies the tokens to be issued to this device in a similar fashion to the way an RO ID identifies a RO. This ID should match the Token Delivery ID in the preceding *ROAP-TokenRequest* message. Devices must discard any *ROAP-TokenDeliveryResponse* message with a token delivery ID which is identical to the one in any previously processed *ROAP-TokenDeliveryResponse* messages.

Device Nonce: if present (2-pass), the nonce MUST have the same value as the corresponding parameter value in the preceding ROAP-TokenRequest or ROAP-TokenConsumptionReport. If the Device Nonce is incorrect, the ROAP-TokenDeliveryResponse processing will fail and the Device MUST discard the received Token Delivery Response PDU.

Token Quantity: contains the number of tokens being issued. If this is a positive number, the device should increment its token store by the given quantity. If it is a negative number the device should decrement the token store by the given quantity. If the value is zero, then this TokenDeliveryResponse is only being used to acknowledge receipt of a TokenConsumptionReport and not to install new tokens on the device.

Token Reporting URL: the presence of this parameter indicates that token consumption from this token delivery must be reported. The parameter defines the URL to which the *ROAPTokenConsumptionReport* message should later be sent.

Earliest Reporting Time: the device should report consumption after this time and before the latest reporting time. If the device reports consumption of tokens before the date/time defined in this parameter, in the subsequent token delivery response the RI may not change the latest token consumption time. In other words the next delivery of tokens is within the same reporting period. The field should only be defined when a token reporting URL is specified.

Latest Reporting Time: the device should report consumption before this time and after the earliest reporting time. If the RI receives the report before this time, it should send a new *ROAP-TokenDeliveryResponse* message before the latest token consumption time so the device can continue consumption. This field should only be defined when a token reporting URL is specified.

Latest token consumption time: after the date/time indicated in this parameter, the device SHALL NOT use any tokens which have been received after the last *ROAP-TokenDeliveryResponse* message which includes a token reporting URL. If reports are being made on time by the device, this date is constantly being updated and therefore consumption should never be blocked. This field should only be defined when a token reporting URL is defined.

OCSP Response: this parameter, when present, SHALL be a complete set of valid OCSP responses for the RI's certificate chain. The Device MUST NOT fail due to the presence of more than one OCSP response element. This parameter will not be sent if the Device sent the Extension No OCSP Response in the preceding ROAP-RegistrationRequest (and the RI did not ignore that extension).

Certificate Chain: this parameter MUST be present unless a preceding ROAP-TokenRequest message contained the Peer Key Identifier extension, the extension was not ignored by the RI, and its value identified the RI's current key. When present, the value of a Certificate Chain parameter shall be as described for the Certificate Chain parameter of the ROAP-RegistrationResponse message.

Extensions: this parameter allows to add future extensions to ROAP-TokenDeliveryResponse message.

Signature: a signature on data sent in the protocol. The signature is computed using the RI's private key and the current message (besides the Signature element itself). The signature method is as follows:

- All elements except the Signature element are canonicalized using the exclusive canonicalization method defined in [XC14N].
- The resulting data d is considered as input to the signature operation.
- The signature is calculated on d in accordance with the rules of the negotiated signature scheme

The Device MUST verify this signature. A Device MUST NOT accept the token acquisition as successful unless the signature verifies, the RI certificate chain has been successfully verified, and the OCSP response indicates that the RI certificate status is good. If the acquisition protocol failed, the Device MUST NOT install the received tokens.

9.2.3.1 Message Syntax

The **<TokenDeliveryResponse>** element specifies the ROAP-TokenDeliveryResponse message. It has complex type **roap:TokenDeliveryResponse**, which extends the basic **roap:Response** type.

```
<element name="TokenDeliveryResponse" type="roap:Response"/>

<complexType name="TokenDeliveryResponse">
 <annotation>
  <documentation xml:lang="en">
   Message sent from RI to Device to deliver tokens
  </documentation>
 </annotation>
 <complexContent>
  <extension base="roap:Response">
   <sequence minOccurs="0">
    <element name="deviceID" type="roap:Identifier"/>
    <element name="riID" type="roap:Identifier"/>
    <element name ="tokenDeliveryID" type="ID"/>
    <element name="nonce" type="roap:Nonce" minOccurs="0"/>
    <element name="tokenQuantity"  type="Integer"/>
    <element name="tokenReportingURL"  type="anyURI" minOccurs="0"/>
    <element name="earliestReportingTime"  type=" dateTime "  minOccurs="0"/>
    <element name="latestReportingTime"  type=" dateTime"  minOccurs="0"/>
    <element name="latestTokenConsumptionTime"  type="dateTime" minOccurs="0"/>
    <element name="certificateChain" type="roap:CertificateChain"  minOccurs="0"/>
    <element name="ocspResponse" type="base64Binary" minOccurs="0" maxOccurs="unbounded"/>
    <element name="extensions" type="roap:Extensions" minOccurs="0"/>
    <element name="signature" type="base64Binary"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

Figure 26: Message syntax of token delivery response

The **Status** simple type enumerates all possible error messages. The following additional status values (shown in **<simpleType name="Status">**, Figure 27, are defined to support token management. These values are only valid in a TokenDeliveryResponse message.

- TokenConsumptionReportError: The RI did receive a token consumption report but it was erroneous and the device should resend.
- NoTokenConsumptionReport: The RI did not receive a token consumption report yet, but was expecting one as the present date and time is later than the last token consumption time in a previous token delivery message.

```
<simpleType name="Status">
  <restriction base="string">
    <enumeration value="Success"/>
    <enumeration value="Abort"/>
    <enumeration value="NotSupported"/>
    <enumeration value="AccessDenied"/>
    <enumeration value="NotFound"/>
    <enumeration value="MalformedRequest"/>
    <enumeration value="UnknownCriticalExtension"/>
    <enumeration value="UnsupportedVersion"/>
    <enumeration value="UnsupportedAlgorithm"/>
    <enumeration value="NoCertificateChain"/>
    <enumeration value="InvalidCertificateChain"/>
    <enumeration value="TrustedRootCertificateNotPresent"/>
    <enumeration value="SignatureError"/>
    <enumeration value="DeviceTimeError"/>
    <enumeration value="NotRegistered"/>
    <enumeration value="InvalidDCFHash"/>
    <enumeration value="InvalidDomain"/>
    <enumeration value="DomainFull"/>
    <enumeration value="DomainAccessDenied"/>
    <enumeration value="RightsExpired"/>
    <enumerationvalue="TokenConsumptionReportError"/>
    <enumerationvalue="NoTokenConsumptionReport"/>
  </restriction>
</simpleType>
```

Figure 27: Updates to status type

Upon transmission or 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 ROAP protocol.

9.3 Extensions for ROAP for Reporting

Reporting can be done via ROAP by devices with a backchannel. The report from the device is based on the ROAPRequest type. The response expected to such a request is a ROAP-TokenDeliveryResponse or ROAP-TokenAcquisitionTrigger.

ROAP-TokenConsumptionReport	
Parameter	**Mandatory/Optional**
Device ID	M
RI ID	M
Token Delivery ID	M
Nonce	M
Report Time	M
Tokens Consumed	M
Certificate Chain	O
Extensions	O
Signature	M

Table 37: ROAP TokenConsumptionReport

Device ID: identifies the requesting Device.

RI ID: identifies the RI.

Token Delivery ID: should be is identical to the TokenDeliveryID value in the last ROAP-TokenDeliveryResponse message received by the device from this RI. The Token Delivery ID can be used by the RI to link this consumption report to the previous ROAP-TokenDeliveryMessage which defined the reporting period, reporting time, etc. for this report.

Nonce: this nonce is chosen by the Device. Nonces are generated and used in this message as specified in Section 5.3.10 of the OMA DRM 2.0 specification.

Report Time: the current DRM Time, as seen by the Device.

Tokens Consumed: contains information on how many tokens were consumed since the last report.

Certificate Chain: this parameter is sent unless it is indicated in the RI Context that this RI has stored necessary Device certificate information. When used the parameter value SHALL be as desribed for the Certificate Chain parameter in the ROAP-RegistrationRequest message.

Extensions: this parameter allows to add future extensions to the ROAP-TokenConsumptionReport message.

Signature: a signature on this message (besides the Signature element itself). The signature method is as follows:

- The message except the Signature element is canonicalized using the exclusive canonicalization method defined in Section 5.3.3 of [DRM-v2].
- The result of the canonicalization, *d*, is considered as input to the signature operation.
- The signature is calculated on *d* in accordance with the rules of the negotiated signature scheme

The RI MUST verify the signature on the ROAP-TokenConsumptionReport message.

Finally, the device must receive and process a ROAP-TokenDeliveryResponse in response to a ROAP-TokenConsumptionReport. The ROAP TokenDeliveryResponse message is used to acknowledge receipt of the report and optionally deliver new tokens. On receipt of a ROAP TokenDeliveryResponse, the device SHALL clear all token consumption information for the preceding report period.

9.3.1 Message Syntax

The **<tokenConsumptionReport>** element specifies the ROAP-TokenConsumptionReport message. It has complex type **roap:TokenConsumptionReport**, which extends the basic **roap:Request** type.

```
<element name="tokenConsumptionReport" type="roap:TokenConsumptionReport"/>

<complexType name="TokenConsumptionReport">
 <annotation>
  <documentation xml:lang="en">
   Message sent from Device to RI to report token consumption report
  </documentation>
 </annotation>
 <complexContent>
  <extension base="roap:Request">
   <sequence>
    <element name="deviceID" type="roap:Identifier"/>
    <element name="riID" type="roap:Identifier"/>
    <element name="nonce" type="roap:Nonce"/>
    <element name="tokenDeliveryID" type="ID"/>
    <element name="time" type="dateTime"/>
    <element name="tokensConsumed"  type="nonNegativeInteger"/>
    <element name="certificateChain" type="roap:CertificateChain" minOccurs="0"/>
    <element name="extensions" type="roap:Extensions" minOccurs="0"/>
```

```
      <element name="signature" type="base64Binary"/>
    </sequence>
   </extension>
  </complexContent>
</complexType>
```

Figure 28: Message syntax of token consumption report

10.Subscriber Groups

10.1 Introduction

Subscriber groups enable the efficient addressing of receiver device groups in BCROs.

A subscriber group is a set of devices that share a group address along with cryptographic key material and algorithms that allow any subset of this group to be associated with a cryptographic key. A subscriber group can be cryptographically secure, which means that it has the additional property that any device from the group cannot deduce the distinct cryptographic keys for subsets that exclude the device.

The capability to address multiple devices using a single message provides for improved efficiency of the communication protocols. In particular it is very beneficial in the distribution of BCROs.

10.2 Addressing

10.2.1 Addressing Modes

Subscriber group addressing allows for three addressing modes, as is explained in Figure 29 below.

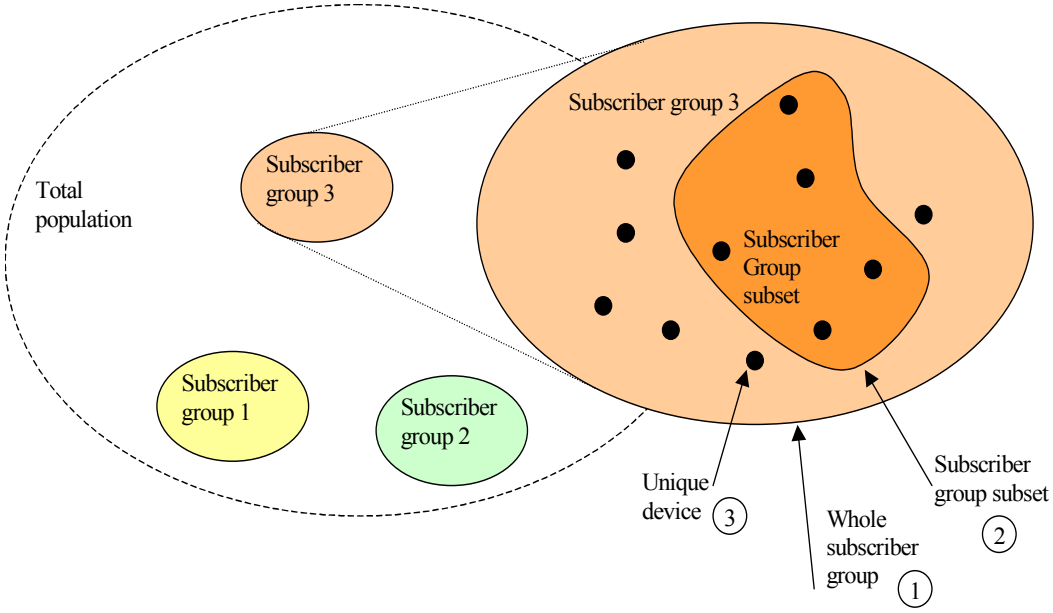


Figure 29: Subscriber group concept

A whole subscriber group contains all devices in a group. A subscriber group subset can be smaller than or as large as the whole group. One or more subscriber groups form the population of devices.

The following sections describe the relation between the registration data and the BCRO. The registration data is sent to the device after successful registration of the device. At a later stage the device may receive a BCRO as a means to obtain the content (encryption) key, which in turn is used to decrypt the encrypted content. When using subscriber group addressing, the content key is encrypted with an Inferred Encryption Key (IEK) by the RI.

There are three types of addressing possible.

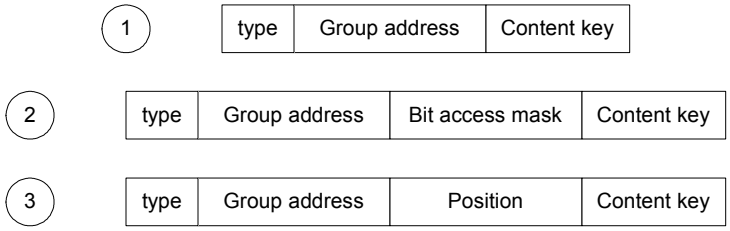


Figure 30: Addressing modes

The first addressing mode addresses the whole subscriber group, each of which has a unique address. The second addressing mode allows the rights issuer to specify exactly which devices in a subscriber group may access the BCRO. This is done by adding a Eurocrypt style bit access mask to the group address. Each device in the subscriber group has a unique position in that group (determined at registration time). The bit in the bit access mask at this position determines whether the BCRO may be processed by a device.

The third addressing mode addresses a single unique device. This is achieved by appending the device's position in the subscriber group to the subscriber group address.

10.2.2 Subscriber Group Identifier

To identify a subscriber group, a subscriber group subset or a subscriber group unique device, a new identifier type is required. The following schema defines the **roap:SubscriberGroupIdentifier** identifier:

```
<complexType name="SubscriberGroupIdentifier">
  <sequence>
    <choice>
      <element name="subscriberGroupBase" type="base64Binary"/>
      <sequence>
        <element name="flexibleGroupAddress" type="base64Binary"/>
        <element name="uniqueDeviceFilter" type="base64Binary" minOccurs="0"/>
      </sequence>
    </choice>
    <choice minOccurs="0">
      <element name="subscriberAccessMask" type="base64Binary"/>
      <element name="subscriberPosition" type="base64Binary"/>
    </choice>
  </sequence>
</complexType>
```

The SubscriberGroupIdentifier MUST contain *either* a **<subscriberGroupBase>** element *or* a **<flexibleGroupAddress>** element. If a Device in a Fixed Subscriber Group is addressed, the **<subscriberGroupBase>** element MUST be present. If a Device in a Flexible Subscriber Group is addressed, the **<flexibleGroupAddress>** element MUST be present and the <uniqueDeviceFilter> element MAY be present.

10.2.2.1 Fixed Subscriber Groups

In the case of a Fixed Subscriber Group, the field unique_device_filter from Section 6.2.2.2.1 consists of a fixed_group_address and a fixed_position_in_group. The fixed_group_address is represented by 31 or 32 bits depending on the subscriber group size. The <subscriberGroupBase> element contains the base64 representation of this fixed_group_address.the **<subscriberGroupBase>** element contains the base64 representation of the (depending on the subscriber group size) 31 or 32 bit field fixed_group_address, as is described in Section 7.2.2.2.1. If the subscriber group has a size of 512 devices, the 31 bit field fixed_group_address is padded to a 32 bit field by adding a least significant bit (the *padding bit*). Unless the **<subscriberPosition>** element is included, the padding bit can have an arbitrary value.

In Fixed Subscriber Groups, the **<uniqueDeviceFilter>** element MUST NOT be included.

When the whole Fixed Subscriber Group is addressed, neither a <subscriberAccessMask> nor the <subscriberPosition> element is included.

When a unique Device in a Fixed Subscriber Group is addressed, the **<subscriberPosition>** element is included. The element contains the base64 coding of the 8 least significant bits of the fixed_position_in_group field (see also Section 8.2.1. In the case the subscriber group size is 512 Devices, the most significant bit is stored in the padding bit in the **<subscriberGroupBase>** element.

When a subset of a Fixed Subscriber Group is addressed, the **<subscriberAccessMask>** element is included. The **<subscriberAccessMask>** element contains the base64-coded 256 or 512 bit field bit_access_mask as described in Section 8.2.1.

10.2.2.2 Flexible Subscriber Groups

In the case of a Flexible Subscriber Group, the **<flexibleGroupAddress>** element contains the base64 representation of the flexible_group_address() field as is described in Section 7.2.2.2.1. Before base64 coding, the flexible_group_address() field is zero-padded (with less than 8 bits) to ensure byte alignment.

In Flexible Subscriber Groups, the **<subscriberPosition>** element is only used for registration in the ROAP-RegistrationResponse message to signal the position of the Device in the Flexible Subscriber Group. This position is first coded in an OMADRMPositionInGroup() structure (see 8.2.3.2), then is zero-padded (with less than 8 bits) and finally stored in base64-coded form in the **<subscriberPosition>** element.

When the whole Flexible Subscriber Group is addressed neither of the elements **<subscriberAccessMask>, <subscriberPosition>** and **<uniqueDeviceFilter>** is included.

When a unique Device in a Flexible Subscriber Group is addressed, the **<uniqueDeviceFilter>** element is included. This element contains the base64 representation of the 40-bit unique_device_filter (see Section 7.2.2.2.1.

When a subset of a Flexible Subscriber Group is addressed, the **<subscriberAccessMask>** element contains the base64 coded flexible_bit_access_mask() as described in Sections 8.2.1 and 8.2.2.

10.3 Confidentiality of Message Content

10.3.1 Introduction

If the subscriber group addressing is cryptographically secure, then it can be used very effectively to distribute a BCRO to such a subset, where the content encryption keys in the BCRO are protected with the distinct key associated with that particular subset. All devices in the subset can determine this key, and hence can decrypt the content encryption keys in the BCRO. All other devices in the group cannot, and therefore cannot access the protected content.

Refer to C.17 for a more detailed introduction to confidentiality in the subscriber group addressing concept.

10.3.2 Subscriber Group Key Material

Each subscriber group has a single unique group key that is used to protect the confidentiality of sensitive broadcast information when the subscriber group is addressed as a whole. This unique group key (*UGK*) is transferred to each device in the subscriber group upon registration with the rights issuer. The *UGK* is shared between all devices in the same subscriber group.

Each device in a subscriber group also receives a unique device key that is used to protect the confidentiality of sensitive broadcast information when device addressing is used (subscriber group address and subscriber position). This unique device key (*UDK*) is transferred to the device upon registration with the rights issuer.

Each device in a subscriber group also has a subset of the node keys NK_i for the case that two or more, but not all devices in a subscriber group are addressed by a BCRO. The keys in the subset are called Subscriber Group Keys (SGKs) or, in the case of a Flexible Subscriber Group, Flexible Subscriber Gorup Keys (FSGKs). The device can use these keys to compute all device keys DK_j, except its own device key.

10.3.3 Fixed Subscriber Groups and Flexible Subscriber Groups

In this specification, the Subscriber Groups come in two flavours. There are the **Fixed Subscriber Groups**, which have a fixed size of 256 or 512 devices, and the **Flexible Subscriber Groups**.

The two flavours appear in the device_registration_response message and the BCRO. During registration the Device is informed whether it is assigned to a Flexible Subscriber Group or a Fixed Subscriber Group. The subsequent messages to a specific Subscriber Group will always be of the same flavour as in the registration.

Broadcast Services and Devices MAY support Flexible Subscriber Groups and/or Fixed Subscriber Groups or no Subscriber Groups at all. The use of Subscriber Groups is bearer specific and is specified in the various adaptation specifications.

10.3.3.1 Fixed Subscriber Groups

Subscriber Groups of this type have a size of 256 or 512 devices. Devices in a Subscriber Group of 256 have a 32 bit group address (indicating the Subscriber Group) and the position of the device in the Subscriber Group is specified by 8 bits. For devices in a Subscriber Group of 512 devices the group address has 31 bits and the position in group is expressed by 9 bits. In a group of 256 devices, each device gets 8 SGKs, whereas in a group of 512 devices, each device gets 9 SGKs.

The following fields are typical Fixed Subscriber Group fields:

- fixed_group_address
- fixed_position_in_group
- group_size_flag
- SGK (Subscriber Group Key)

These fields are only used when the Device is assigned to a Fixed Subscriber Group.

10.3.3.2 Flexible Subscriber Groups

The size of a Flexible Subscriber Group can be selected from a set of 31 possible sizes ranging from 2^1 to 2^{31} (always powers of 2). The device is informed about the size of the Flexible Subscriber Group at registration.

Flexible Subscriber Groups also allow the choice of another broadcast encryption scheme. See Table 54 in Appendix C.11 for more details.

The following fields are typical Flexible Subscriber Group fields:

- flexible_device_data
- flexible_group_address
- flexible_position_in_group
- flexible_group_size_indicator
- broadcast_encryption_scheme
- FSGK (Flexible Subscriber Group Key)

These fields are only used when the Device is assigned to a Flexible Subscriber Group.

Note that when the zero-message broadcast encryption scheme is used, the FSGK has the same meaning as the SGK. However, there are 8 or 9 SGKs whilst there can be up to 31 FSGKs, supporting group sizes up to $2^{31} \approx 2\ 000\ 000\ 000$ devices.

Devices and RIs that support Flexible Subscriber Groups MUST support group sizes of up to $2^{14} = 16\ 384$ devices. They MAY support bigger group sizes.

10.3.4 Addressing Subscriber Groups

To protect the confidentiality of the key material included in an asset in a given BCRO, that key material is encrypted using a key called *Inferred Encryption Key (IEK)*, and its computation depends on the addressing mode used by the BCRO and the content identifier *BCI* of the first asset encoded in the BCRO.

10.3.4.1 Domain Addressing

In case domain addressing is used by the BCRO, then the *IEK* depends on the Broadcast Domain Key (BDK) of the addressed domain:

IEK = HMAC-SHA1-128{ *BDK* } (*BCI*)

10.3.4.2 Unique Device Addressing

In case unique device addressing is used by the BCRO, then the *IEK* is computed using the Unique Device Key (UDK):

IEK = HMAC-SHA1-128*{ UDK } (BCI)*

10.3.4.3 Group Addressing

In case the whole group is addressed by the BCRO, then the *IEK* is computed using the Unique Group Key (UGK):

IEK = HMAC-SHA1-128*{ UGK } (BCI)*

10.3.4.4 Subset Addressing

If the BCRO is addressed to two or more, but not all devices in a subscription group and the zero-message broadcast encryption scheme is used, the *IEK* is equal to the Deduced Encryption Key (DEK), which is computed using the concatenated device keys associated with the revoked devices as follows:

IEK = *DEK* = HMAC-SHA1-128{ DK_a || DK_b || DK_c ||… || } (*BCI*)

Where DK_a, DK_b, DK_c, … are the device keys of the devices that have a '0' in the addressing bitmask and therefore must not be allowed to access the asset.

Each node key NK_i is associated with a node number. The nodes from the subscriber group key derivation tree are sequentially numbered in a breadth-first manner, starting from the root node with number 0. See Appendix C.17.1 for more details on the node numbering.

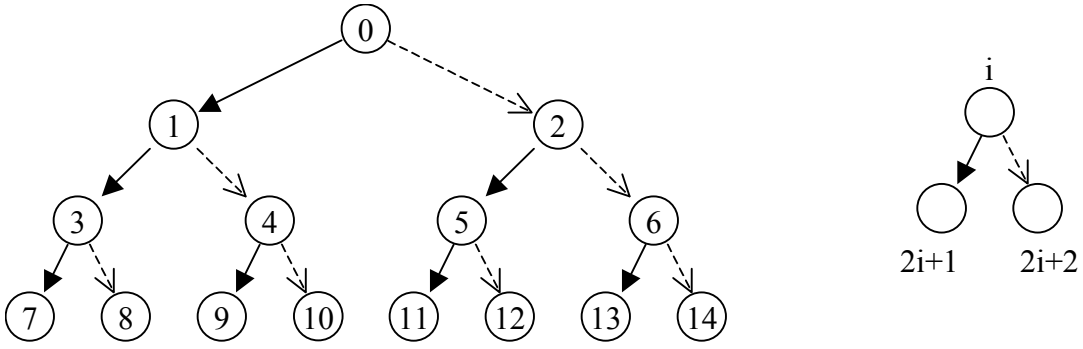


Figure 31 Subscriber group node (and node key) numbering

Each device gets a set of node keys ((F)SGKs) such that it can apply the key derivation functions ‘left’ and ‘right’ to compute the node keys of all leaf nodes except for the leaf node that is associated with its own position. The Device can also derive all keys that are in the path from an SGK to a leaf node. The relation between subscriber position and associated leaf node number is:

leaf node number = subscriber position + subscriber group size – 1

Each node in the subscriber group key tree can be associated also with a depth in the tree. The root node has depth 0, its child nodes 1 and 2 have depth 1. In general, the child nodes of a node with depth *d* have depth *d+1*. With this defined, the set of node keys has the following property: all nodes associated with the node keys given to a device have different depth, and the root node is not part of this set.

If NK_i denotes the key associated with node *i*, then the key derivation functions 'left' and 'right' are defined as:

$$NK_{2i+1} := \text{left}(i) := \text{AES-128-ENCRYPT}\{ NK_i \}((2i+\text{LEFT\_CONSTANT}) \bmod 2^{128})$$

$$NK_{2i+2} := \text{right}(i) := \text{AES-128-ENCRYPT}\{ NK_i \}((2i+\text{RIGHT\_CONSTANT}) \bmod 2^{128})$$

Where LEFT_CONSTANT = 0x01010101010101010101010101010101 and RIGHT_CONSTANT = 0x02020202020202020202020202020202.

Example:

The very small subscriber group from Figure 31 consists of 8 devices (numbered 0 to 7, associated with nodes 7 to 14). A Rights Issuer randomly generates a key for the root of the key tree. From that root key, all other keys in the tree are computed using the key derivation functions.

Then:

i	NK_i	Derivation
0	0123456789abcdef0123456789abcdef	(not derived, randomly determined by the rights issuer , never send to devices)
1	e50ae5f0c279c65ec332d9bcc1117e92	=AES{0123456789abcdef0123456789abcdef } (01010101010101010101010101010101)
2	1c55d4149103150fc10da6800dd5884a	=AES{0123456789abcdef0123456789abcdef } (02020202020202020202020202020202)
3	4d8249b05af00c67ee7b600927a75eb6	=AES{e50ae5f0c279c65ec332d9bcc1117e92} (01010101010101010101010101010103)
4	a9f5aa423ca8d1efbbcf50014be61b82	=AES{e50ae5f0c279c65ec332d9bcc1117e92} (02020202020202020202020202020204)
5	bad128a946f85174d66ffc326fe5f9e8	=AES{1c55d4149103150fc10da6800dd5884a} (01010101010101010101010101010105)
6	811adb84ab42947df9028444448aa7e4	=AES{1c55d4149103150fc10da6800dd5884a} (02020202020202020202020202020206)
7	b4bdbf499b8c43e184d270fe198f08df	=AES{4d8249b05af00c67ee7b600927a75eb6} (01010101010101010101010101010107)
8	660967ab0c5d5960652b484af71ecba8	=AES{4d8249b05af00c67ee7b600927a75eb6} (02020202020202020202020202020208)
9	8e465d379f5cfc324a9c0f3eacb92ee1	=AES{a9f5aa423ca8d1efbbcf50014be61b82} (01010101010101010101010101010109)
10	3527bdd7eaccb5c0e6d89a7004d603d8	=AES{a9f5aa423ca8d1efbbcf50014be61b82} (0202020202020202020202020202020a)
11	c0d7b5c58b9732b5480dc4c54093c738	=AES{bad128a946f85174d66ffc326fe5f9e8} (0101010101010101010101010101010b)
12	49916d5d931a68ce2e99bf6726098f2e	=AES{bad128a946f85174d66ffc326fe5f9e8} (0202020202020202020202020202020c)
13	3dd317bc38087c3f310c238861958706	=AES{811adb84ab42947df9028444448aa7e4} (0101010101010101010101010101010d)
14	0433bc21b1d5ca5b2b0778475c2ca5ba	=AES{811adb84ab42947df9028444448aa7e4} (0202020202020202020202020202020e)

The key NK_0 is the root key from which all other keys are derived. It is randomly selected by the rights issuer and is never distributed to any device. A device that knows NK_0 can compute all device exclusion keys, also its own, and hence circumvent being excluded.

The keys 7 to 14 in bold are the keys associated with the devices 0 to 7 respectively.

To effectively disallow devices 1,6 and 7 to access a certain asset, the rights issuer derives a *DEK* by concatenating the device revocation keys (NK_8, NK_{13} and NK_{14}) , and using this concatenation as key for computing a MAC over the broadcast content identifier *BCI* as retrieved from the BCRO:

$$DEK = \text{HMAC-SHA1-128}\{ DK_1 \,||\, DK_6 \,||\, DK_7\}\,(BCI)$$
$$= \text{HMAC-SHA1-128}\{ NK_8 \,||\, NK_{13} \,||\, NK_{14}\}\,(BCI)$$

Device 2 (that is not excluded) has been given the following node keys $\{ NK_{10}, NK_3, NK_2 \}$

$$DK_1 = NK_8$$
$$= \text{right}(NK_3)$$

$$DK_6 = NK_{13}$$
$$= \text{left}(NK_6)$$
$$= \text{left}(\text{right} (NK_2))$$

$$DK_7 = NK_{14}$$
$$= \text{right}(NK_6)$$
$$= \text{right}(\text{right}(NK_2))$$

And note that in computing DK_6 the device already computes NK_6, that is also applied in the computation of DK_7.

An attempt by e.g. device 7 to compute the *DEK* will fail because it will be given the key set { NK_{13}, NK_5, NK_1 }, and although that is sufficient to calculate DK_1 and DK_6, it cannot compute its own key.

$$DK_1 = NK_8$$
$$= \text{right}(NK_3)$$
$$= \text{right}(\text{left}(NK_1))$$

$$DK_6 = \underline{NK_{13}}$$

$$DK_7 = NK_{14}$$
$$= \text{right}(NK_6)$$
$$= \text{right}(\text{right}(NK_2))$$
$$= \text{right}(\text{right}(\text{right}(NK_0)))$$

At that point, there is no more key derivation function available to compute the unknown key NK_0 (which is the root key, and is never distributed to devices!). Because device 7 cannot compute DK_7 it cannot construct the key $DK_1 || DK_6 || DK_7$ that is needed to compute the *DEK* for this BCRO.

10.3.5 Consistency

For any device position, it is easy to derive the node numbers of the key nodes for which the keys must be included in the set of node keys for that device. If $sibling_i$ yields the unique node that has the same parent as node i, $parent_i$ yields the parent node of node i, and NK_i yields the key associated with node i, then the following algorithm yields all the keys to be included in the device's set of derivation keys:

KeySet = ∅

while node ≠ root

node := $\text{sibling}_{\text{node}}$

KeySet := KeySet ∪ NK_{node}

node := $\text{parent}_{\text{node}}$

end

With this algorithm it is easy to check the consistency of the key set and the subscriber position given to a device.

11.Broadcast Service Support

11.1 Key Stream Handling

Key stream handling is an ordered sequence of steps that allows refreshing the cryptographic context of the broadcast content transport layer.

The following steps are required to complete this process:

- reception of a key stream message (out of scope of this document)
- linking this key stream message to an appropriate GRO
- using the authentication key from the GRO to authenticate the key stream message
- using the encryption key from the GRO to decrypt the key material in the key stream message
- refresh the cryptographic context of the transport layer using the decrypted key material.

11.1.1 Linking Key Stream Message to Generalised Rights Object

To successfully process a key stream message, the Device MUST find an appropriate GRO that refers to the correct content and holds the appropriate key material. Both normal RO (e.g. as delivered via ROAP) as well as BCROs are equally usable in this respect.

A key stream message is linked to a GRO by comparing content identifiers. In a normal RO, this is the value encoded in the **<o-ex:context>** element of the **<o-ex:asset>** elements inside the **<o-ex:rights>** element in the **<ro>** element of the **<protectedRO>** element in the **<ROResponse>** message. In a normal RO, the content identifier is a CID (Content ID). In a BCRO, this is the value of the BCI fields in each asset.

The CID is constructed as follows:

program RO

program_CID = 'cid:' || stringtomakeitunique || '#P' || baseCID ||'@' || hex(program_CID_extension)

service RO

service_CID = 'cid:' || stringtomakeitunique || '#S' || baseCID ||'@' || hex(service_CID_extension)

Note that 'program_CID' and 'service_CID' shall be globally unique. Note further that because of the specification of 'baseCID' in the OMA BCAST SG, the global uniqueness is already guaranteed and therefore, 'stringtomakeitunique' shall be the empty string.

The hex() function is a hexadecimal presentation of the parameter containing hexadecimal characters 0-9 and a-f (in lowercase) with possible preceding zeros.

EXAMPLE For a 16-bit value 2748, hex() returns "0abc". There are always two characters generated for each byte.

The BCI used is a binary value, which is defined by the key stream layer:

program BCRO

program_BCI = SHA1-64('cid:' || stringtomakeitunique || '#P' || baseCID || '@') || program_CID_extension)

service BCRO

service_BCI = SHA1-64('cid:' || stringtomakeitunique || '#S' || baseCID || '@') || service_CID_extension)

Note that 'program_BCI' and 'service_BCI' shall be globally unique. Note further that because of the specification of 'baseCID' in the OMA BCAST SG, the global uniqueness is already guaranteed and therefore, 'stringtomakeitunique' shal be the empty string.

To process a key stream message, the DRM Agent should be given also the *bsdaID* and the *serviceBaseCID*. These values are defined in the service guide [BCAST10-SG].

In case *program_flag*=1 in the key stream message, the agent would first try to find a GRO with matching content identifiers. The DRM agent will determine if any of the GROs it has stored governs an asset that has a content identifier (CID or BCI) based on *bsdaID*, *baseCID* and *program_CID_extention*.

Where *program_CID_extension* is found in the key stream message. Alternatively, the agent could be given the whole content identifier in combination with the key stream message to be processed. This requires the agent's environment to compute this content identifier using information from the service guide [BCAST10-SG] and the key stream message.

If one or more of such GROs are found, the Device MUST select one GRO among those as specified in 5.9 "Order of Rights Object Evaluation" in [DRMREL-v2]. That GRO is now linked to this key stream message.

Otherwise, if service_flag=1 in the key stream message (regardless of P=1 or P=0) then the agent tries to find GROs with a content identifier (CID or BCI) based on *bsdaID*, *baseCID* and *service_CID_extention*.

If one or more of such GROs are found, one is selected among those using the normal OMA procedures. That GRO is now linked to this key stream message.

If no suitable GRO is found, then the DRM Agent MUST stop processing this key stream message.

11.1.2 Authentication

Using the suitable and selected GRO, it MUST verify the proper MAC field.

If the GRO is linked to the key stream message using a program_BCI or a program_CID, then it holds a holds a PEK/PAK combination, and the PAK must be used to verify the program_mac field of the key stream message.

If the GRO is linked to the key stream message using a service_BCI or a service_CID, then it holds a SEK/SAK combination, and the SAK must be used to verify the service_mac field of the key stream message.

If the verification succeeds it may proceed with decryption of the traffic key material.

When the computed MAC differs from the value encoded in the message, verification fails and the DRM Agent MUST stop processing this key stream message.

11.1.3 Confidentiality

After successful verification, the protected_traffic_key_material field may be decrypted.

There are three possibilities:

1. **service_flag=1/program_flag=0**

   This is the case in subscriber only access. The content is not available as a pay-per-view item. The protected_traffic_key_material should be decrypted using the SEK in the GRO. Successful verification proves SAK valid, so SEK can be applied.

2. **service_flag=0/program_flag=1**

   This is the case in pay-per-view only access. The content is not available as a subscription item. The protected_traffic_key_material should be decrypted using the PEK in the GRO. Successful verification proves a valid PAK, so PEK can be applied.

3. **service_flag=1/program_flag=1**

   This is the case in subscriber access combined with pay-per-view access. The protected_traffic_key_material should be decrypted using the PEK in the GRO, and if a GRO holding the PEK is not available, then an intermediate decryption of the protected_program_key_material is required.

   Based on the selected GRO, two scenarios can be followed.

   If the selected GRO holds a PEK/PAK pair, then PEK can be applied to decrypt the protected_traffic_key_material field.

   If the selected GRO holds a SEK/SAK pair, then first the SEK is applied to decrypt the protected_program_key_material field. From the decrypted protected_program_key_material field, the PEK is found. With the PEK now available, the protected_traffic_key_material field is decrypted.

If the DRM Agent encounters any problems during the process of decrypting the traffic key material, it MUST stop processing this key stream message.

11.1.4 Cryptographic Context Update

After successful linking a key stream message to a GRO, verification of the appropriate MAC and decryption of the confidential key material, the cryptographic context of the broadcast content transport layer can be updated.

11.1.5 On the Use and Precedence of Program GROs, Service GROs and permissions_category

An operator can give a user access to a service using a Service GRO. The Service GRO is typically valid for a longer period, e.g. a month. A Service GRO contains permissions and constraints for all programs in the service.

In the simplest case, the Service GRO contains one set of permissions and constraints, which means that all programs in the service have the same permissions and constraints for users that have received such a Service GRO.

An operator might wish to treat certain programs of a service in a special way. Perhaps the operator wants to sell additional permissions to users for specific programs. Perhaps the operator wants to put different restrictions to certain programs for users who just have the subscription (e.g. more restrictions for a new movie, less restrictions for preview type content). There are two mechanisms provided in this specification to do this.

11.1.5.1 Use of Multiple Program GROs in Addition to a Service GRO.

The first mechanism this specification provides for having different permissions and constraints for programs is to have a Program GRO for each program for which the permissions and constraints are different from the ones in the encompassing Service GRO. Therefore, a Program GRO of a program SHALL have precedence over a Service GRO of the service for which the program is a part.

These Program GROs can be broadcast to broadcast only devices and they can be sent over the interaction channel to devices that have an interaction channel. The distribution of these Program GROs can consume much bandwidth in case there are many programs for which the permissions and constraints are different from the Service GRO.

11.1.5.2 Use of permissions_category and Service GROs

The second mechanism this specification provides for having different permissions and constraints for programs is by using the permissions_category. In this second mechanism, only Service GROs are distributed, which can save considerable bandwidth in comparison with the first mechanism, where multiple Program GROs are distributed.

It is possible to have more than one set of permissions and constraint for each asset in the Service GRO. Each of these sets is identified by an 8-bit number which is called the permissions_category. The Traffic Key Message can contain a permissions_category number. This number in the Traffic Key Message indicates which of the sets of permissions and constraints in the Service GRO applies to the program that is broadcast at the time of reception of the permissions_category number in the Traffic Key Message. The permissions_category SHOULD remain constant over one program.

Using the permissions_category number functionality, it is possible to have a service with programs with different permissions and constraints by just distributing a Service GRO containing multiple sets of permissions and constraints, each set indexed by permissions_category.

The RO as defined in OMA DRM V2 does not have the permissions_category functionality. Therefore, this specification contains an enhancement that defines this functionality.

11.1.5.3 Use of Program GROs without a Service GRO

An operator can give access to the user on a program by program basis using Program ROs, without using a Service RO. This may be useful in e.g. Pay-Per-View business models.

11.1.5.4 Precedence of Permissions and Constraints in Program and Service GROs.

The permissions and constraints in a Program GRO for a program in a service SHALL have precedence over any set of permissions and constraints in any Service GRO applicable to that service. This is also independent on whether or not a set of permissions and constraints is indicated for that program in the Traffic Key Message by the permissions_category field.

In the absence of a Program GRO for a program in a service, the set of permissions and constraints that is indicated for that program in the Traffic Key Message by the permissions_category field SHALL have precedence over any other set of permissions and constraints in any Service GRO that is applicable to the service of which that program is a part.

If the above two cases do not apply, all permissions and constraints in a Service GRO that is applicable to the service of which the program 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 GRO that is applicable to the service of which that program is a part.

If the above three cases do not apply, all permissions and constraints in the Service GRO that is applicable to the service of which the program is a part apply for that program i.e. the device can select any permission together with its constraint.

12.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 C.13.

Within this chapter, the objects and messages to be carried are referred to as "objects".

The data carried by broadcast 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 OMA BCAST Service Guide.

All RI Streams forming part of any service SHALL be identified as such in the OMA BCAST Service Guide.

An informative schedule MAY be broadcast for RI Services. Where available, this SHALL be provided as part of the OMA BCAST Service Guide. It is used to indicate times at which data for particular sets of devices or Broadcast Groups will be broadcast. This allows devices to listen to RI Services only when necessary, and will also allow Service Operators 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.

This specification aims to allow enough flexibility for operators to fulfil their own requirements for message and RO delivery, and to trade off latency against bandwidth, while also allowing devices to minimise 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 chapter 12.1.

12.1 Expected Mode of Operation [Informative]

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 this specification, in order to support the needs of Service Operators and Rights Issuers. Any message can be carried in any RI Service, at the discretion of the Service Operator and Rights Issuer.

Using the OMA BCAST Service Guide, 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 RI 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 Broadcast Group. A schedule giving the times at which information for particular sets of devices or Broadcast Groups will be carried is made available in the OMA BCAST Service Guide. Devices need only listen to the service at the times relevant to it.
- It is expected that all BCROs required by any authorised device to receive a protected service will be carried in a Data 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 OMA BCAST Service Guide. The mechanism specified in this chapter 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.

12.2 Scheduled RI Stream

In a Scheduled RI Stream, the timing of message broadcast may be scheduled in some way, according to device or Broadcast 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 Broadcast Groups. Where provided, it SHALL be available in the OMA BCAST Service Guide.

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.

12.3 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.

12.4 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 BCROs for free previews or free-to-view services; or

BCROs for content being carried by the service.

Devices SHALL be able to identify In-Band RI Streams within protected services from the OMA BCAST Service Guide.

12.5 Broadcast Format of RI Streams

All the objects defined in this specification are carried in Rights Issuer Streams. The format of these streams is defined in this chapter.

These streams SHOULD have the following characteristics:

- The bandwidth overhead of the stream format SHOULD be minimised.
- 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.

12.5.1 IP Characteristics

Rights Issuer streams are IP streams advertised in the OMA BCAST Service Guide. The OMA BCAST Service Guide SHALL also carry an identifier for the version of this specification used to generate each RI Stream. The format of the IP packets is UDP [RFC 768]. This specification does not specify any limits to the length of these IP packets – this will instead be determined by the underlying network.

Section 12.5.2 defines the packet format of the RI Stream.

12.5.2 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.

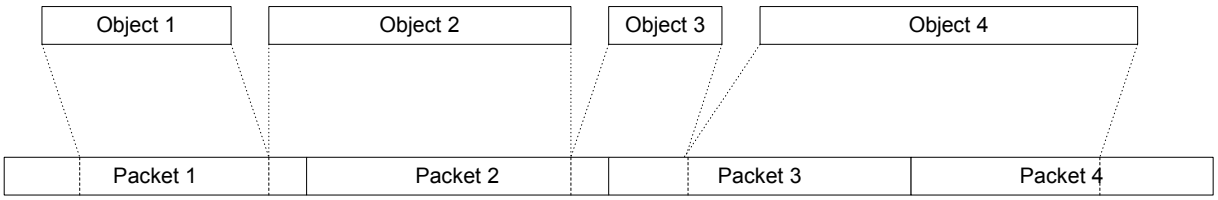


Figure 32: Example mapping of objects to RI Stream packets

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 38: 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(27).

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.

12.5.3 Implementation Notes

12.5.3.1 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 12.5.3.2 as apparently missing packets could later be received due to packet reordering.

12.5.3.2 Changes in Packet Order (Informative)

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 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 this specification.

It is recommended that Service Operators and Rights Issuers minimise changes in packet order within their systems.

12.5.3.3 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.

12.6 Mapping of Messages to RI Services and Streams

Within a broadcast network, devices discover streams using the OMA BCAST Service Guide and various SI/PSI (Service Information/Program Specific Information) tables.

- The OMA BCAST Service Guide 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 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.

Broadcast 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 chapters describe how messages are mapped to services and streams.

12.6.1 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 OMA BCAST Service Guide, 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.

12.6.2 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.

12.7 Discovery of RI Services, Streams and Schedule Information

RI services and their schedule information are announced using the RightsIssuerServiceData document format described in this section. The OMA BCAST Service Guide announces a Rights Issuer service as Service Fragment with ServiceType “RI Service” [BCAST10-SG]. The RightsIssuerServiceData file is delivered in a file delivery session associated with the service.

Once a device has acquired the RightsIssuerServiceData for the RI service, it uses the information contained in it to identify RI Streams addressed to a group that the device or subscriber belongs to. If the RI stream is scheduled, that information includes the time interval during which the device is expected to acquire the RI Service messages it is interested in.

The broadcast delivery of the instances of 'RightsIssuerServiceData' has the following characteristics and constraints. For the delivery the network SHALL

- use FLUTE file delivery session containing at least one FDT Instance,
- list all the delivered files in every instance of FDT and
- use the string "application/vnd.oma.drm.risd+xml" as the value of 'Content-Type' for every instance of 'RightsIssuerServiceData' in every FDT Instance.

12.7.1 Rights Issuer Service Data

Table 39: Definition of Rights Issuer Service Data

Name	Type	Category	Cardinality	Description	Data Type
RightsIssuer Services	E	NO/TO	1	The RightsIssuerServices document describes the existing RI services. Contains the following element: RightsIssuerServiceData	
RightsIssuer ServiceData	E1	NO/TO	1..N	The RightsIssuerServiceData element describes the data associated with a RI service. BSMs that support the broadcast mode of operation and Broadcast Devices of the DRM profile, SHALL support this element and all its subelements and attributes. Contains the following attributes: id version Contains the following sub-elements: RightsIssuerStream	
id	A	NO/TO	1	The identifier of the Rights Issuer from which this RI service is originated.	anyURI
version	A	NO/TO	1	The OMA BCAST version to which this document conforms. For this specification the version SHALL be 1	positiveInteger
RightsIssuer Stream	E2	NO/TO	1..N	Data that describes one RI Service Stream. A stream is optionally associated with a device or subscriber group, has an associated IP address to which the RI Service messages are addressed and possibly a time interval indicating when the messages are broadcast. Contains the following attribute: ipAddress port	

				certificateChainUpdate Contains the following sub-elements: Schedule Target	
ipAddress	A	NO/TO	1	The IP address to which the RI stream messages are directed.	string
port	A	NO/TO	1	The port to which the RI stream messages are directed	unsignedShort
certificateChainUpdate	A	NO/TO	1	If the certificateChainUpdate field is set to true, this Rights Issuer Service Stream will contain Certificate Chain Updates within its schedule. These apply to all devices registered with the Rights Issuer using this Rights Issuer Service. The use of Certificate Chain Updates is described in Section 11.8.	boolean
Schedule	E3	NO/TO	0..N	The time interval(s) during which this RI Stream is broadcast. If this element is missing, the RI Stream is not scheduled and the device is expected to listen for service messages at any time. Contains the following attributes : validFrom validTo Contains the following subelement : RepeatInterval	
validFrom	A	NO/TO	1	The first moment when the transmission of messages over this RI stream is valid. This field expressed as the first 32bits integer part of NTP time stamps	unsignedInt
validTo	A	NO/TO	1	The last moment when the transmission of messages over this RI stream is valid. This field expressed as the first 32bits integer part of NTP time stamps	unsignedInt
RepeatInterval	E4	NO/TO	0..1	Indicates the interval time of the repeated distribution of the RI Service Stream messages for this target group or device.	duration

Target	E3	NO/TO	0..N	The data in this element define the range of devices to be addressed in any of the timeslots where this RIStream is scheduled. A device should evaluate the SubScriberGroupType, SubscriberGroup and subscriberGroupMask as described below to evaluate whether its Subscriber Group will be addressed in the timeslot associated with this RI Stream. If its group will be addressed, the device should then evaluate the DeviceAddress and deviceAddressMask fields to determine whether it will be addressed in the timeslot. If it will, the device should listen to the Rights Issuer Service Stream during that timeslot. If DeviceAddress and deviceAddressMask fields are not present, the timeslot may be used to address all devices in the group. If the SubScriberGroupType, SubscriberGroup and subscriberGroupMask fields are not present, but DeviceAddress and deviceAddressMask fields are, then the DeviceAddress and deviceAddressMask fields are absolute device addresses using the longform_udn(), as defined in this specification. Sub-elements: SubscriberGroupType SubscriberGroup DeviceAddress	
SubscriberGroupType	E4	NO/TO	0..1	If TRUE, Fixed Subscriber Groups are used; if FALSE, Flexible Subscriber Groups are used. The default value for this element is TRUE.	boolean
SubscriberGroup	E4	NO/TO	0..1	The Subscriber Group address for the group(s) to be addressed in this Rights Issuer Service Stream. A device can determine whether its Subscriber Group matches this value by bitwise ANDing its Subscriber Group address with the subscriberGroupMask (see below). If the result equals the value in this field, the groups match. The Subscriber Group address is as defined in this specification. Note that SubScriberGroup applies to both the Fixed and a Flexible Subscriber Group address that are defined in this specification as fixed_group_address and flexible_group_address respectively Contains the following attribute:	base64binary

				subscriberGroupMask	
subscriberGroupMask	A	NO/TO	1	A mask to be applied to the SubscriberGroup. See the definition for the SubscriberGroup element above.	base64binary
DeviceAddress	E4	NO/TO	0..1	The device address for the device(s) to be addressed in this scheduled Content item in a Rights Issuer Service. A device can determine whether its device address matches this value by bitwise ANDing its device address with the deviceAddressMask (see below). If the result equals the value in this field, the device addresses match. In case the SubscriberGroup element is not present, the DeviceAddress is in the form of a longform_udn(), as defined in this specification. In case the SubscriberGroupType and SubscriberGroup elements are present, the DeviceAddress is the fixed_position_in_group (for Fixed Subscriber Groups) or flexible_position_in_group" (for Flexible Subscriber Groups). Contains the following attribute: deviceAddressMask	base64binary
deviceAddressMask	A	NO/TO	1	A mask to be applied to the DeviceAddress. See the definition for the DeviceAddress element above.	base64binary

The complete XML schema implementing the data structure above is available as support document [DRM20-Broadcast-Extensions-RISD-XSD].

12.8 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 C.2.2 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 OMA BCAST Service Guide. 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.
- Furthermore, it is strongly recommended that a reference to at least the next certificate chain update is always carried in the OMA BCAST Service Guide.

- 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 chapter, 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.

12.9 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.

12.9.1 Resending of BCROs to Interactive Devices

For an interactive device, requests to resend BCROs can be made via the interaction 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 SHOULD resend the BCROs for that device.

12.9.2 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 7.4. 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.

12.10 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 chapter 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:

- MAY contain any number of Scheduled or Ad-hoc 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 BCROs required to receive services, to reduce the likelihood of a device not having the BCRO required to receive a service which it is entitled to receive.

12.11 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 interaction 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 (or line 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 BCROs. 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 Service for a Rights Issuer with which it is registered, the device SHALL listen to that service.

13.Adapted File Format

This section describes adaptations to the file formats DCF and PDCF that are needed to allow broadcast support to OMA DRM v2.0.

13.1 Common adaptations to DCF and PDCF

13.1.1 Key Info Box

The *ExtendedHeaders* field in the OMADRMCommonHeaders box MAY include one or more instances of the Key Info Box:

```
aligned (8) class OMABCASTKeyInfoBox extends FullBox('obki', version, flags) {
        unsigned int(8) KeyInfosNumber;                         // indicates the number of key infos that follow
        for (i=0;i<KeyInfosNumber;i++){
                bit(1) KeyIssuerPresent;                        // indicates that the key issuer URL is present
                bit(1) STKMPresent;                             // indicates that the STKM is present (only to be used for DCF)
                bit(1) TBKPresent;                              // indicates that the TerminalBindingKey information is present
                bit(1) TBKIssuerURLPresent;                     // indicates that the TBK issuer URL for TBK  is present
                bit(4) rfu;                                     // reserved for future use
                unsigned int(8) KeyIDType;                      // indicates the type of key id that follows
                unsigned int(8) KeyIDLength;                    // KeyID length in bytes
                byte KeyID[];                                   // key_id
                if(KeyIssuerPresent) {
                        unsigned int(16) KeyIssuerURLLength;    // KeyIssuer URL field length in bytes
                        char KeyIssuerURL[];                    // KeyIssuer URL string
                }
                if (STKMPresent) {                              // applies only to DCF, not PDCF
                        unsigned int(16) STKMLength;            // STKM field length in bytes
                        byte STKM[];                            // STKM
                }
                if (TBKPresent) {
                        unsigned int(32) TBK_ID;                // TerminalBindingKeyID
                        if (TBKIssuerURLPresent){
                                unsigned int(16) TBKIssuerURLLength;  // TBK Issuer URL field length in bytes
                                char TBKIssuerURL[];            // TBKIssuer URL string
                                }
                }
        }
}
```

The OMABCASTKeyInfoBox fields are described in Table 40.

Table 40: OMABCASTKeyInfoBox fields

Field name	Type	Purpose
KeyInfosNumber	unsigned int(8)	indicates that the number of key infos that follow
KeyIssuerPresent	bit	indicates that the key issuer URL is present
STKMPresent	bit	indicates that an STKM is present (only for DCF)
TBKPresent	bit	indicates that the TerminalBindingKey information is present

TBKIssuerURLPresent	bit	indicates that the TBK issuer URL for TBK is present
KeyIDType	unsigned int(8)	type of KeyID
KeyIDLength	unsigned int(8)	length of the Key ID in bytes
KeyID	byte[]	value of Key ID
KeyIssuerURLLength	unsigned int(16)	length of the KeyIssuerURL (optional)
KeyIssuerURL	char[]	Key Issuer URL (optional)
STKMLength	unsigned int(8)	length of the STKM in bytes (optional)
STKM	byte[]	STKM (optional)
TBK_ID	unsigned int(8)	TerminalBindingKeyID
TBKIssuerURLLength	unsigned int (16)	TBK Issuer URL field length in bytes
TBKIssuerURL	char[]	TBKIssuerURLstring

The *KeyIssuerURL* in the Key Info box SHALL be used first. If this fails or if the *KeyIssuerURL* is not present, the Device MAY try the RightsIssuerURL in the *OMADRMCommonHeaders* box.

For this version of the specification, the following values for the KeyIDType MUST be used:

Table 41: KeyIDType values

KeyID type	Value	Purpose
OMA BCAST DRM Profile	0x00	OMA BCAST DRM Profile KeyID as defined in [BCAST10-ServContProt]
OMA BCAST Smartcard Profile	0x01	OMA BCAST Smartcard Profile KeyID as defined in [BCAST10-ServContProt]
3GPP MBMS	0x02	3GPP MBMS KeyID as defined in [3GPP TS 33.246] Note this is one option given to MBMS. Other option is for MBMS to define their own box in the Extended Headers field. Both options will be suggested to 3GPP in an LS.

The field STKM_present_flag MAY only be set to 1 for DCF file delivery. In this case, the field STKM contains the key used to encrypt the DCF content. Refer to [BCAST10-ServContProt] for more details. In a PDCF file, this flag MUST be set to 0.

13.2 DCF

13.2.1 File Branding

The ISO base media file format defines a File Type box for identifying the major brand of the media file along with compatible brands. For DCF files conforming to this specification, the File Type box MUST be as defined in OMA DRM v2.0 [DRMCF-v2].

13.3 Adapted PDCF

This section allows a STKM stream (transmitted using Layer 3 of the 4-layer model for Service Protection and Content Protection of RTP streams using ISMACryp) to be stored within a PDCF. Recording is explained in [BCAST10-ServContProt].

The existing PDCF file format as defined in OMA DRM v2.0 [DRMCF-v2] allows audio video content to be stored in a file format together with the relevant OMA DRM information. Audio and video tracks can be encrypted as defined in [DRMCF-v2] using the appropriate CEK stored in a Generalised Rights Object (GRO).

Creating adapted PDCF recordings does not require a GRO. Playback of adapted PDCF recording is governed by the protection_after_reception flags in the key stream (see [BCAST10-ServContProt]) and, for certain values of the protection_after_reception_flags, by GROs.

Content can be streamed over RTP using ISMACryp. To allow storing this kind of streamed content in a PDCF file as samples and not as packets, a couple of adaptations to the PDCF file format are made. This modified PDCF file format is called Adapted PDCF.

Sections 13.3.2 and 13.3.3 explain how to store TEK stream information in Adapted PDCF. In the context of broadcast services, RTP streams can be encrypted at the content level (encrypting Access Units using ISMACryp as explained in [BCAST10-ServContProt]) using TEKs. This key is not the traditional CEK stored in an RO. In the broadcast context the CEK is a Service Encryption Key (SEK) or a Program Encryption Key (PEK) delivered using Layer 2. This SEK or PEK allows the TEK delivered in Traffic Encryption Key stream messages delivered in Layer 3 to be decrypted. The TEK is used to encrypt content transmitted in RTP packets using ISMACryp. As this key changes regularly, Adapted PDCF allows the storage of the relevant TEK stream information.

Section 13.4 specifies the AES_128_BYTE_CTR encryption algorithm. This algorithm is used in ISMACryp and is included in Adapted PDCF to allow the storage of ISMACryp protected AUs in a PDCF file, without re-encryption.

The informative Appendix C.18 describes the logical PDCF box structure.

13.3.1 File Branding

The ISO base media file format defines a File Type box for identifying the major brand of the media file along with compatible brands. Adapted PDCF Files conforming to this specification MUST include a File Type box with the adapted PDCF brand as compatible brand. The adapted PDCF brand is not recommended to be used as a major brand. Note: the major brand for the adapted PDCF should be the same as for the unprotected file. Unlike the File Type box defined for DCF, the File Type box in adapted PDCF does not have a fixed length of 20 bytes.

The adapted PDCF brand is 32 bits (4 octets) wide and MUST have the hexadecimal value 0x6F707832 (‘opx2’). If the adapted PDCF brand is used as major brand, this MUST be followed by the four-octet minor version indicator with the value 0 (0x00000000).

13.3.2 PDCF Adaptation for Key Stream Inclusion

This section details the modifications required in the PDCF file format of OMA DRM v2.0 [DRMCF-v2] so as to allow an OMA key stream to be stored in the PDCF.

The adapted PDCF file format is schematically shown in Figure 33 below in a simplified format, as per OMA DRM v2.0. The only difference between the diagram below and the original PDCF file format is the addition of an OMA STKM track in the Movie Box and the associated OMA STKM track data in the Media Data box. Full backward compatibility with the original PDCF file format is thus ensured.

Details on the PDCF file format, STKM track and details on how to link the STKM track to appropriate audio / video tracks are given in this specification in the sections below.

Supporting the adapted PDCF format defined in this specification is OPTIONAL for a Device, as is the case for the original PDCF format in OMA DRM v2.0.

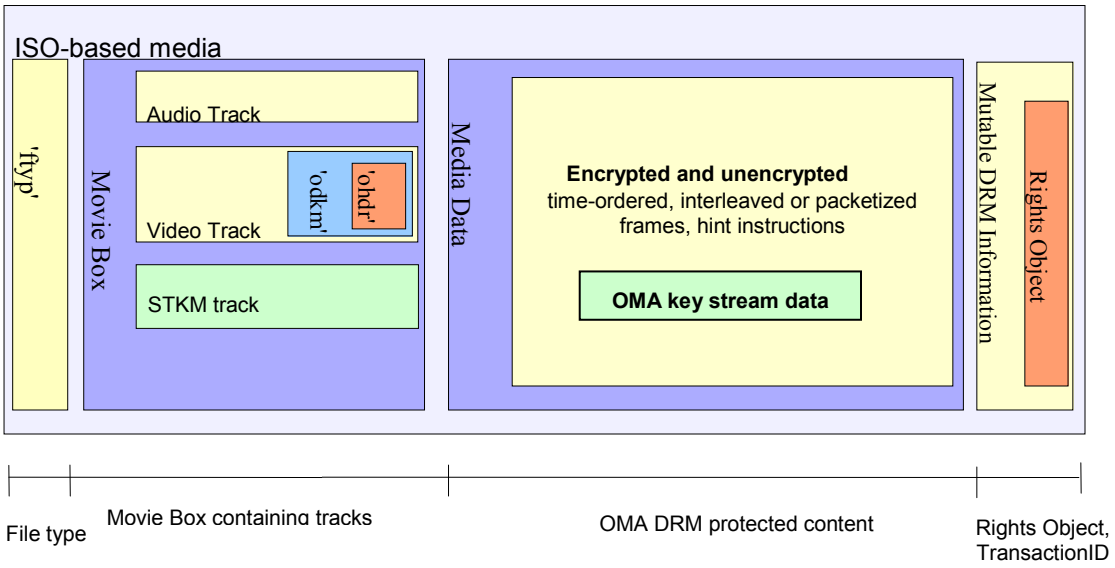


Figure 33: Example of a PDCF with a protected video track

13.3.3 STKM Tracks

A PDCF file, as any other "ISO File Format based" file, contains one or more audio or video tracks as defined in the ISO specification [ISO14496-12] and, for the storage of alternating TEKs, can additionally contain one or more STKM tracks as defined in this specification.

The STKM track is a Timed Metadata track as defined in [ISO14496-12:2005/Amd1].

Note that track references as defined in the ISO File Format provide the mechanism to relate the STKM track with the corresponding media data track. A track-reference of type 'cdsc' is used for linkage. The timing information (as provided as basic functionality in the ISO File Format) in the STKM track and the media data track are used to relate every STKM sample to the corresponding media sample.

The STKM track is defined by the OMAKeySampleEntry as extension of the MetadataSampleEntry as follows:

```
aligned(8) class OMAKeySampleEntry extends MetadataSampleEntry('oksd') {
        unsigned int(8) sample_version;                 // sample version
        unsigned int(8) sample_type;                    // sample type
        if( terminal_binding_flag_in_STKM == 1 ) {      // from the STKM
                unsigned int(32) TerminalBindingKeyID;  // from the SG
                unsigned int(16) RightsIssuerURILength; // Rights Issuer URI field length in bytes
                char RightsIssuerURI[];                 // Rights Issuer URI string
        }
}
```

The OMAKeySampleEntry box contains the following fields:

Table 42: OMAKeySampleEntry fields

Field name	Type	Purpose
sample_version	unsigned int (8)	Identifies OMA key sample version In this specification, sample_version contains 0x00.
sample_type	unsigned int (8)	Identifies the OMA key sample type:

		0x00 for STKM for DRM profile 0x01 for STKM using MIKEY for smartcard profile using (U)SIM 0x02 STKM for BCMCS using R-UIM

The terminal_binding_flag_in_STKM has by default a value of zero. For the smartcard profile from [BCAST10-ServContProt], the value of terminal_binding_flag_in_STKM equals the value of the terminal_binding_flag in the Short Term Key Message (STKM). If the flag contains a one, the fields TerminalBindingKeyID, RightsIssuerURILength and RightsIssuerURI are included in the OMAKeySampleDescriptionEntry box. See [BCAST10-ServContProt] for more details.

The sample description of each STKM track MUST contain exactly one OMAKeySampleEntry.

The samples in the STKM track have the following format:

```
aligned(8) class OMAKeySample {
        unsigned int(8*KeyIndicatorLength) keyIndicator        // key indicator
        unsigned int(8*STKMLength) STKM;                       // short term key message as defined in [BCAST-ServContProt]
}
```

The format of the STKM is described in [BCAST10-ServContProt]. Even though key indicator length and key indicator value are present in the STKM, for optimization purposes, these fields are placed also at the beginning of each OMAKeySample.

In order to provide maximum flexibility, the STKM track version and size are included. As needs evolve, new sample formats can be defined, identifying new formats with new STKM track sample version numbers. This approach ensures that future PDCF specifications will remain fully backward compatible.

13.3.4 OMA DRM Signalling Information

As specified in the OMA DRM v2.0 DCF specification [DRMCF-v2], the ISO ProtectionSchemeInfoBox ‘sinf’ with its sub-boxes is used to carry DRM key management system specific information, thus it is only a container box.

Table 43: PDCF scheme type for OMA DRM

scheme_type	Value	Semantics
OMA DRM	‘odkm’	OMA DRM is used for key management in the PDCF.

Table 44: PDCF scheme version for OMA DRM

scheme_version	Value	Semantics
OMA BCAST 1.0	0x00000300	OMA DRM version is 2.0 extended for BCAST (version 2.0 does not allow the STKM track)

For PDCF files conforming to this specification, in the ISO SchemeTypeBox ('schm') the SchemeType MUST be the 4CC ‘odkm’, and SchemeVersion MUST be 0x00000300. The file MUST contain at least one OMADRMKMSBox. A PDCF MUST support only OMA DRM for the key management system.

The ISO ProtectionSchemeInfoBox ('schi') is used to carry DRM key management system specific information. For this specification, this box MUST include exactly one OMADRMKMSBox (‘odkm’), as the first sub-box. The OMADRMKMSBox includes the OMADRMCommonHeaders box ‘ohdr’, as the first sub-box and the OMADRMAUFormatBox, as the second sub-box.For more details on the information included in the OMADRMCommonHeaders box refer to [DRMCF-v2]. In this version of the specification the EncryptionMethod field in the OMADRMCommonHeaders box is extended with the value 0x03 to contain the additional AES_128_BYTE_CTR algorithm.

In this version, the OMADRMAUFormatBox MUST be present. The value of the fields in the OMADRMAUFormatBox are described below.

```
aligned(8) class OMADRMAUFormatBox extends FullBox('odaf', 0, 0) {
        bit(1) SelectiveEncryption;
        bit(7) reserved;
        unsigned int(8) KeyIndicatorLength;
        unsigned int(8) IVLength;
}
```

Table 45: OMA sample format box fields

Field name	Type	Purpose
SelectiveEncryption	bit(1)	Indicate whether selective encryption is used or not
Reserved	bit(7)	Reserved, SHOULD be set to 0.
KeyIndicatorLength	unsigned int(8)	Size of the key indicator in bytes
IVLength	unsigned int(8)	Size of the IV in bytes

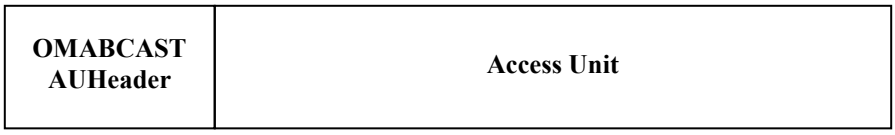
The SelectiveEncryption bit MAY be set either to 1 or to 0. If the selective encryption bit is set to 0 then all content to which the ISO ProtectionSchemeInformationBox applies is encrypted and no "encrypted" field is present in OMABCASTAUHeader. If the selective encryption bit is set to 1 then the OMABCASTAUHeader preceding Access Units indicates whether or not a particular AU is encrypted.

The KeyindicatorLength describes the size of the key indicator in bytes. In this version of the specification, the value of KeyIndicatorLength does not have to be set to 0.

13.3.4.1 OMABCASTAUHeader

The OMABCASTAUHeader specifies the format for each access unit protected by OMA DRM. This header MUST precede the codec-specific sample data in each access unit. It provides the OMA DRM information whose length is specified in the OMADRMAUFormatBox defined in Section 13.3.4. The OMABCASTAUHeader is placed before each AU as shown in Figure 34.

Figure 34: OMABCASTAUHeader and access unit



The OMABCASTAUHeader is defined as follows:

```
aligned(8) class OMABCASTAUHeader {
        if( SelectiveEncryption == 1) {                 // from the OMASampleFormatBox
                bit(1)      EncryptedAU;                // Encryption indicator
                bit(7)      reserved;                   // Must be zero
        }
        else EncryptedAU = 1;
        if( EncryptedAU == 1 ) {
                unsigned int(8 * IVLength) IV;
                unsigned int(8 * KeyIndicatorLength) KeyIndicator;
        }

}
```

Table 46: OMABCASTAUHeader fields

Field name	Type	Purpose
EncryptedAU	bit(1)	Encryption Indicator for the access unit.
IV	unsigned int(8)	IV preceding the access unit playload.
KeyIndicator	unsigned int(8)	Key indicator field preceding the access unit payload.

Table 47: Encryption indicator values

Encrypted	Value	Semantics
None	0	Access unit is not encrypted.
Encrypted	1	Access unit is encrypted.

A playing Device uses the header information for decryption purposes and is able to extract the actual sample(s).

13.4 AES counter encryption in byte mode and salt

To record an ISMACryp stream directly to a PDCF file, a couple of adaptations to the OMA DRM v2.0 PDCF file format [DRMCF-v2] are needed.

The AES counter mode algorithm as appears in [DRMCF-v2], AES_128_CTR, is slightly modified. This modified version will be referred to as AES_128_BYTE_CTR. Using the AES_128_BYTE_CTR algorithm allows the storing of ISMACryp AUs without re-encryption. The AES_128_BYTE_CTR algorithm corresponds to the encryption algorithm used in ISMACryp. The two AES counter mode algorithms are explained in more detail in Section 12.3.1.

In Section 12.3.2, makes the adaptations needed to signal that the AES_128_BYTE_CTR algorithm is used. This is done by adding a new possible value for the EncryptionMethod field in the OMADRMCommonheaders box.

Section 12.3.3 handles the adaptations needed for the use of a *Salt*. In the AES_128_BYTE_CTR algorithm, the Salt contains the 64 most significant bits of an Initialization Vector (IV) and is transmitted only once per track. The salt omits the need to send all the bits of the IV in each AU and therefore reduces the overhead in the AU Header.

13.4.1 Description of AES counter modes

In both AES counter mode algorithms, a block of plaintext is encrypted to a block of ciphertext by xoring it with a generated pseudorandom *KeyBlock* based on AES encryption, which is defined as follows:

$$\text{KeyBlock}_i = \text{AES-128-ENCRYPT}\{K\}(i),$$

where K is the key used to encrypt the content and i is a 128-bit integer. Each KeyBlock has a length of 16 bytes and uses a new value of i. The kth byte in a KeyBlock_i is denoted by $\text{KeyBlock}_i[k]$, where k=0 corresponds to the first byte. Similarly the nth byte of the ciphertext (in an AU) is denoted by C[n] and nth byte of the associated plaintext by P[n], where n=0 corresponds to the first byte.

The encrypter/decrypter has an internal variable CTR. This variable is used to calculate i in KeyBlock_i. The exact calculation of i depends on the counter mode. To calculate the first value of CTR, the cipher algorithms need an *Initialization Vector*. There is one Initialization Vector per AU.

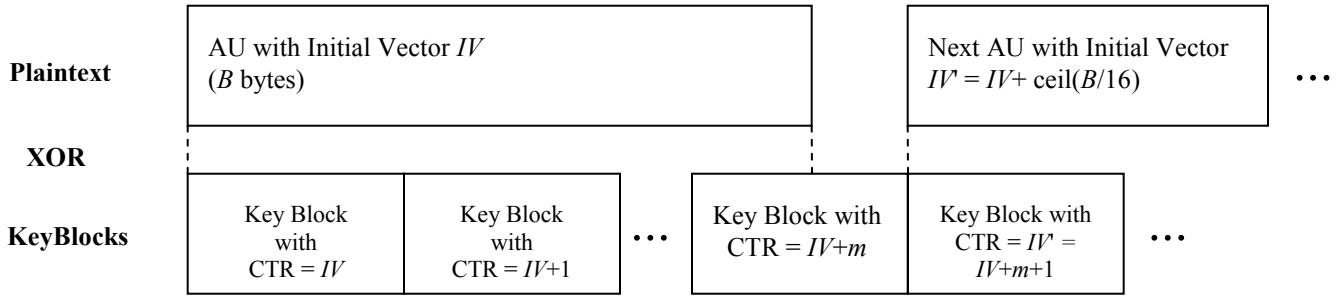
The basic difference between the two AES counter mode algorithms lies in the fact that for AES_128_ CTR the CTR is increased by 1 for each (16 byte) KeyBlock, whilst for AES_128_BYTE_CTR the CTR is increased by 1 for each byte. Furthermore, AES_128_BYTE_CTR uses a Salt, whereas AES_128_CTR does not.

13.4.1.1 AES_128_CTR

The AES_128_CTR algorithm is defined in [DRMCF-v2]. Using this algorithm, the initial value of CTR is equal to the value of the Initialization Vector *IV*. CTR is increased by one for each KeyBlock. The first byte of plaintext is encrypted using the first byte in KeyBlockCTR, with CTR=*IV*.

The plaintext on byte position *n*, P[*n*], is encrypted to the ciphertext on byte position *n*, C[*n*], as follows:
C[*n*] = P[*n*] xor $KeyBlock_{IV}$+floor($_n$/16)[*n* mod 16]. The decryption is similarly done as follows:
P[*n*] = C[*n*] xor $KeyBlock_{IV}$+floor($_n$/16)[*n* mod 16].

If this mode is used, it should be avoided to encrypt two different AUs using the same KeyBlock. Therefore encryption in this mode should always start with a fresh CTR value for each AU. This means that possibly unused bytes from the last KeyBlock used to encrypt the previous AU are discarded. The following figure illustrates this:

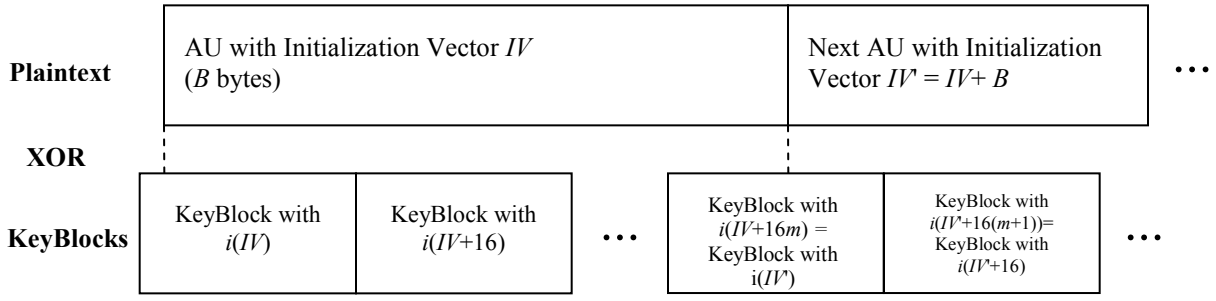


13.4.1.2 AES_128_BYTE_CTR

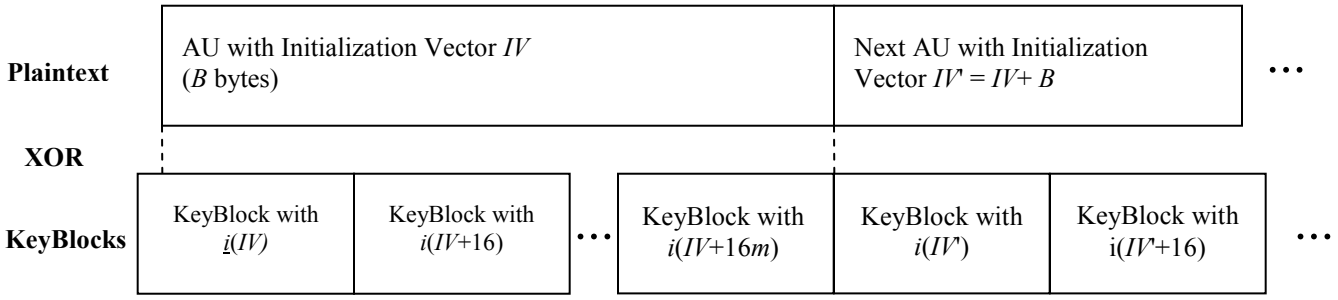
In the case of AES_128_BYTE_CTR, the IV derivation algorithm is technically identical to ISMACryp, so that the initial value of CTR is also equal to the value of the Initialization vector *IV*. CTR is increased by one for each byte of ciphertext/plaintext. CTR is used together with a 64-bit integer *Salt* to calculate the KeyBlock. The Salt is stored in the OMADRMSalt box in the ExtendedHeaders of the OMADRMCommonHeaders box. The 4 least significant bits of CTR contain the byte offset in the $KeyBlock_i$(CTR) with *i*(CTR) = ((Salt << 64) xor (CTR >> 4)). Notice that *i*(CTR) is a function *i* depending on CTR.

The plaintext on byte position *n*, P[*n*], is associated with a CTR value CTR = IV + *n*. P[*n*] is encrypted to the ciphertext on byte position *n*, C[*n*], as follows: C[*n*] = P[*n*] xor $KeyBlock_i$(CTR) [CTR & 0xF]. The decryption is similarly done as follows:
P[*n*] = C[*n*] xor $KeyBlock_i$(CTR)[CTR & 0xF].

For encryption in this mode, it is RECOMMENDED to increase the Initialization Vector continuously over the borders of AUs: when the Initialization Vector associated an AU has a value *IV* and the AU contains *B* bytes of ciphertext, then the Initialization Vector of the next AU has the value *IV*+*B*. This allows possibly unused bytes of the last KeyBlock of one AU to be used for the encryption of the first bytes of the next AU. The following figure illustrates this case:



If there are no unused KeyBlock bytes left, the next AU starts with a fresh KeyBlock, as is illustrated in the following figure:



The bitsize of CTR is the same as the bitsize of the Initialization Vector, IVLength. To ensure that the CTR does not overflow, the IV MUST be reset in due time. This can be avoided by choosing the IVLength big enough.

13.4.2 The EncryptionMethod field

Because of the addition of the AES_128_BYTE_CTR algorithm, the possible values in the EncryptionMethod field in the OMADRMCommonHeaders box are extended with the value 0x03. This value signals the use of the AES_128_BYTE_CTR algorithm. Table 48 summarizes the possible values for the EncryptionMethod field.

Table 48: Possible values for the EncryptionMethod field

Algorithm-id	Value	Semantics
NULL	0x00	No encryption for this object. NULL encrypted Content Objects may be used without acquiring a Rights Object. Value of the *PaddingScheme* field MUST be 0.
AES_128_CBC	0x01	AES symmetric encryption as defined by NIST [AES]. 128 bit keys. Cipher block chaining mode (CBC). 128 bit initialization vector prefixing the ciphertext (for non-streamable PDCF files this is included in the OMADRMBCASTHeader). Padding according to RFC 2630.
AES_128_CTR	0x02	AES symmetric encryption as defined by NIST [AES]. 128 bit keys. Counter mode (CTR). 128 bit initialization vector prefixes the ciphertext (for non-streamable PDCF files this is included in the OMABCASTAUHeader). For each **cipherblock** the counter is incremented by 1 (modulo 2^{128}). No padding.
AES_128_BYTE_CTR	0x03	AES symmetric encryption as defined by NIST [AES]. 128 bit keys. Counter mode (CTR). An initialization vector of minimal 8 bits and maximal 64 bits prefixes the ciphertext (for non-streamable PDCF files this is included in the OMABCASTAUHeader).

		For each **byte** of ciphertext the counter is incremented by 1. No padding.

13.4.3 The OMADRMSalt Box

Using the AES_128_BYTE_CTR encryption method, the ExtendedHeaders field in the OMADRMCommonHeaders box MUST include one instance of the OMADRMSalt box:

```
aligned (8) class OMADRMSalt extends Box('oslt') {
        unsigned int(8) SaltLength;          // Length of the Salt field in bits. MUST be 64
        unsigned int(SaltLength) Salt;       // Salt needed for AES_128_BYTE_CTR
}
```

The OMADRMSalt box contains the field *Salt*, which is needed for AES_128_BYTE_CTR encryption method.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
n/a	n/a	No prior version –or- No previous version within OMA
OMA-xxyyz-V1_0-20021001-A	01 Oct 2002	Initial document to address the basic starting point Ref TP Doc# OMA-TP-2002-1234-xxyyzForApproval
OMA-xxyyz-V1_1-20030405-A	05 Apr 2003	description of changed Ref TP Doc# OMA-TP-2003-0321-xxyyzV1_1forApproval

A.2 Draft/Candidate Version V1_0 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-DRM-XBS-V1_0	21 Feb 2005	n/a	First draft outline based on input to the joined committees (BAC-DLDRM and BAC-BCAST): OMA-BCAST-2005-0048-Joint-BCAST-DRM-Task-Workplan as well as discussions and contributions to the email reflector (prioritisation of work items).
	15 Mar 2005	6.3	OMA-BCAST-2005-0100R01-token-based-metering (approved at the Chicago BCAST/DLDRM joint meeting).
	17 Mar 2005	6.1 & 6.2 7	OMA-DLDRM-2005-0064-Broadcast-Rights-Object (approved in conference call 17 mar 2005) OMA-DLDRM-2005-0071R01-subscriber-group-addressing (approved in conference call 17 mar 2005)
	8 Apr 2005	5.1 5.1 5.1	OMA-DLDRM-2005-0085R01-offline-notification-of-detailed-device-data (approved in conference call 6 apr 2005) OMA-DLDRM-2005-0086-Push-binary-Device-Registration-data (approved in conference call 6 apr 2005) OMA-DLDRM-2005-0087R01-offline-notification-of-short-device-data (approved in conference call 6 apr 2005)
	11 May 2005	5.1.4, 7.2.3, 7.3.5 & 7.3.6	OMA-DLDRM-2005-0100R01-Broadcast-Extensions-Device-Registration (approved in Singapore BCAST/DLDRM joint meeting)
	21 Jun 2005	10 6.3, 7 9 6.3 8.3	OMA-BCAST-2005-0094R04-PDCF-adaptation-for-Traffic-Encryption-Key-stream OMA-BCAST-2005-0100R03-token-based-metering-specification-text OMA-DLDRM-2005-0098R03-Broadcast-Extensions-Key-Stream-Handling OMA-DLDRM-2005-0114R01-Broadcast-Extension-Key-Stream-Authentication-Key-Transfer OMA-DLDRM-2005-0169-Broadcast-encryption-key-derivation-functions
	11 Aug 2005	Many sections changed. Many sections moved to appendix A.	OMA-DLDRM-2005-0213-offline-notification-of-short-device-data OMA-DLDRM-2005-0214R01-update-ri-certificate OMA-DLDRM-2005-0215R01-update-DRM-time-via-broadcast OMA-DLDRM-2005-0216-update-contact-numbers-for-out-of-band-notification OMA-DLDRM-2005-0217-force-broadcast-reregistration OMA-DLDRM-2005-0220-domains-for-broadcast OMA-DLDRM-2005-0221-Token-handling-using-broadcast-channel OMA-DLDRM-2005-0224-RI-Services-for-broadcast OMA-DLDRM-2005-0229-authentication-for-broadcast OMA-DLDRM-2005-0232-BCRO_update OMA-DLDRM-2005-0243-BCRO-message-tag

Document Identifier	Date	Sections	Description
	10 Sep 2005	6.1.1 A.2 6.1.3 A.4 A.13 7.2.1 7.2.2	OMA-DLDRM-2005-0211R02-offline-notification-of-detailed-device-data OMA-DLDRM-2005-0212R02-push-device-registration-data-to-device-during-broadcast-registration OMA-DLDRM-2005-0223R02-OCSP-grace_-broadcast- OMA-DLDRM-2005-0254-BCRO-optimisations
	26 Sep 2005	7.2.2 + new Section 10.1.5 7.2.2	OMA-DLDRM-2005-0286R01-BCRO-permissions-category-support OMA-DLDRM-2005-0297-delete-old-table-in-sec7-2-2-of-XBS
	6 Oct 2005	Chapter 6 11.8 A.1.2 A.8 A.9.1 A.11.1.2	OMA-DLDRM-2005-0306-PDF-version-of-OMA-DLDRM-2005-0300R01-fix-broken-crosslinks-and-other-stuff
	21 Nov 2005	7.2.4 7.4 (new)	OMA-DLDRM-2005-0298R03-CR-XBS-REL-Save-Permission
	9 Dec 2005	7.2.1 12.2.2 & 12.3	OMA-DLDRM-2005-0254-BCRO-optimisations (one aspect of that CR forgotten in the September 10 edits). OMA-BCAST-2005-0470R01-BCast-PDCF-alignment
	23 Jan 2006	Many sections changed 7.3 & A.9.2.1	OMA-DLDRM-2005-0433-xRO-terminology-in XBS OMA-DLDRM-2005-0399R02-transport-of-ksm-authentication-seed-in-icros
	20 Mar 2006	A.12.4 (new) A.7.4 (new) A.1 (inserted) 7.2.1 7.2.2 (inserted) 12.2.1.3 7.2.1 7.4 3.2	OMA-BCAST-2005-0617R04-GKM-BCRO-Delivery OMA-BCAST-2005-0620R03-gkm-member-keynode-id OMA-BCAST-2006-0048-Security-Analysis-BCAST-Content-Protection OMA-BCAST-2006-0055R01-efficient-BCRO-addressing-to-subscriber-groups OMA-BCAST-2006-0144R01-PDCF-key-track-type OMA-BCAST-2006-0168R06-Sign-BCROS-revisited OMA-BCAST-2006-0223-CR-device-class-definitions
	22 Dec 2006		Incorporated the folloxing CR: OMA-BCAST-2007-0331

	26 Apr 2007	9.1.7	OMA-BCAST-2005-0121R03-CR_New_permission_called_ACCESS
		6.1.2.1.6	OMA-BCAST-2006-0283-20060326-DRM-XBS_Logical_Bug_Fix
		3.3	OMA-BCAST-2006-0284-20060326-XBS_Missing_Notations_Bug_Fix
		7.2.2.2.3	OMA-BCAST-2006-0285-20060326-DRM-XBS_Protection_keyset_Bug_Fix
		8.2.7.7	OMA-BCAST-2006-0286R01-DRM-XBS_System_Constraint_Descriptor
		7.4.1.6	OMA-BCAST-2006-0287-DRM-XBS_Token_request_Section
		3.3, 6.1, 6.2, 6.3, 6.4, 7.1, 7.2, 7.3, 9.3	OMA-BCAST-2006-0297R03-CR-XBS-wording-and-clerical-changes
		8.4.3.1, 8.4.4	OMA-BCAST-2006-0320R03-CR_Logical_Bug_fix
		3.3	OMA-BCAST-2006-0411-Missing_abreviations
		7.2.2.2.1	OMA-BCAST-2006-0471-Missing-broadcast-registration-items
		C.14.2.1	OMA-BCAST-2006-0551-XBS-bug-fix-authentication-seed-id
		9.1, 9.2, 9.3	OMA-BCAST-2006-0582-metering-comments-resolution
		7.2.2.2, 8.2, 8.2.3.2, 10.3.3, C.11	OMA-BCAST-2006-0592R04-CR-DVB-H-compatible-flexibility-in-XBS
		C.11.1, 0	OMA-BCAST-2006-0639-XBS_Tag_Length_Format_Bug_Fix
		7.1.2, C.13	OMA-BCAST-2006-0643R01-XBS-Message-Protocol-Versions
		C.13	OMA-BCAST-2006-0644-XBS-Encryptec-Keyset-Block-Integrity
		7.2.2.2.3	OMA-BCAST-2006-0645R01-CR_ReviewComment_DX108_tokenrequest_metering
		Error! Reference source not found., **Error! Reference source not found.**, 3.3, 5, 6	OMA-BCAST-2006-0646R02-CR_DRM_XBS_5_I_Auth_Key_Hierarchy
		5	OMA-BCAST-2006-0647R03-CR_DRM_XBS_5_II_Key_Hierarchy

		7.6.4.1	OMA-BCAST-2006-0656-CR_Cleanup_of_token_delivery_response_message
		Error! Reference source not found., 7.2, 7.7, 13, C.17	OMA-BCAST-2006-0666R01-CR_XBS_Missing_References
			OMA-BCAST-2006-0682-CR_Signalling_of_OFT
		Error! Reference source not found., 8.2, 10.3.3.2, C.11.1, 8.2.3.5	OMA-BCAST-2006-0683-CR_XBS_Optimisation_of_ECT
			OMA-BCAST-2006-0700-CR-Rename_Keytrack_STKMtrack
			OMA-BCAST-2006-0706R01-CR_Removing_Purchase
			OMA-BCAST-2006-0715R03-CR_Session_key_length_computation
		Many	
		7.4.1	OMA-BCAST-2006-0716R01-CR_SignatureFlag_must_be_two_bit
		Error! Reference source not found., 7.2, 7.7.4.3, C.12	OMA-BCAST-2006-0746R01-CR_Removal_of_Domain_Keyset
		7.5.2.1.2, 7.5.4.1.2, 7.5.5.1.2, 7.7.7.1	OMA-BCAST-2006-0759-CR_Domain_Generation_Clarification
			OMA-BCAST-2006-0760-CR_XBS_comment_DX023
		7.7.5.1	OMA-BCAST-2006-0761-CR_Clarification_for_subscriber_group_addressing
		C.1.5.2.4	OMA-BCAST-2006-0762R02-CR_Mixed_mode_definitions
		13	OMA-BCAST-2006-0768R01-CR_BCI_definitions_DX144
		10.3.2, 10.3.4	OMA-BCAST-2006-0774R01-CR_Notify_Time_Drift_DX110
		Error! Reference source not found., 7.7.2, Many	OMA-BCAST-2006-0782-CR_XBS_Encryption_Clarification
			OMA-BCAST-2006-0784R03-CR_XBS_SCR_Tables
		8.2.4, 11.1.1	OMA-BCAST-2006-0786R04-CR_sign_token_delivery_response
			OMA-BCAST-2006-0787-CR_sign_or_mac_bcros
		7.4.1.7	OMA-BCAST-2006-0788R01-CR_Relation_XBS_18Crypt
		8.4.3.3	OMA-BCAST-2006-0792-CR_Missing_message_tag_value
		Appendix B	OMA-BCAST-2006-0802-CR_Recording_using_adapted_PDCF
		7.6.4	
		8.1.1	
		Error! Reference source not found.	

		10.3.4.4, C.11.1, C.17	OMA-BCAST-2006-0844R01-CR_Comment_DX184_Remove_A.8.4
		8.2.6, 8.2.7	OMA-BCAST-2006-0859-CR_Comment_DX155_Timed_Count_Tokens_for_BCRO
		8.1.2	OMA-BCAST-2006-0874-CR_Comment_DX052
		10.1	OMA-BCAST-2006-0889-CR_Comment_DX055
		Many	OMA OMA-BCAST-2006-0892-CR_DX081_one_coding_style_in_XBS
		7.5.5.1.3	OMA-BCAST-2006-0899R02-CR_Comment_DX123_Explanation_Contact_Types
		Error! Reference source not found., 8, C.11	OMA-BCAST-2006-0905R01-CR_18Crypt_DRM_Profile_differences
		13.1.1	OMA-BCAST-2006-0954R04-CR_DRMv2.x_DCF_KeyID_box
		C.14	OMA-BCAST-2006-0959-CR_Comment_DX074_Make_A10_Normative
		7.7.1	OMA-BCAST-2006-0967R01-CR_Domain_Concept
		7.3, 10.2.2, C.4	OMA-BCAST-2006-0975-CR_ROAP_XML_schema_for_XBS
		C.14.2.1	OMA-BCAST-2006-1001-CR_XBS_Bug_fix_in_OMA_DRM_RO_Example
		7, C.13	OMA-BCAST-2006-1002R02-CR_XBS_Introduction_for_Broadcast_Device_and_Domain_Management
		8.2.1, 8.2.4	OMA-BCAST-2006-1004R01-CR_Purchase_Item_ID_DX143
		7.1.2, 7.2.1.2, 7.4.1, 7.7.4.3, 8.4.3, C.6.1, C.9, C.10	OMA-BCAST-2006-1011R01-CR_Comment_DX074_Add_Missing_normative_text
		1, **Error! Reference source not found.**, 5	OMA-BCAST-2006-1012R02-CR_CR_Comment_DX045_Relation_XBS_SPCP
		Error! Reference source not found., 3.3, 13	OMA-BCAST-2006-1036-CR_PDCF_byte_counter_mode_and_salt
		C.8	OMA-BCAST-2006-1102-CR_Comment_DX074_Annex_A.5

		8.2.2	OMA-BCAST-2007-0043-CR_XBS_Correction_of_examples
		Error! Reference source not found., 7, 8.1.3	OMA-BCAST-2007-0071-CR_XBS_RI_Service_Clarification OMA-BCAST-2007-0078R01-CR_Clarification_non_Streamable_PDCF
		13	OMA-BCAST-2007-0079R01-CR_STKM_Retrieval
		12.2.3	OMA-BCAST-2007-0119R01-CR_XBS_Device_Data_Inform_Message
		7.2.1.3.2	OMA-BCAST-2007-0120R01-CR_XBS_Local_Domain_Key_and_Local_Domain_Filter
		Many	OMA-BCAST-2007-0146R03-CR_XBS_consistent_use_of_DEK_and_IEK
		Error! Reference source not found., **Error! Reference source not found.**, 5.1, 8.1.2, 8.2.4, 10.2.1, 10.3.4	OMA-BCAST-2007-0150R01-CR_XBS_KeyInfoBox OMA-BCAST-2007-0165R04-CR_Announcement_of_RI_Service_related_data OMA-BCAST-2007-0171-CR_MIME_type_for_RightsIssuerServiceData OMA-BCAST-2007-0278R01-CR_TS_XBS_Clarification_of_Impulse_Pay_Per_View
		13	OMA-BCAST-2007-0284-CR_Remove_smartcard_profile_references
		12.7	OMA-BCAST-2007-0285R02-CR_SCR_table_entries_for_Subscriber_Groups
		C.19	OMA-BCAST-2007-0286-CR_Signalling_of_subscriber_group_parameters_in_XML
		Error! Reference source not found.	OMA-BCAST-2007-0288R01-CR_Specification_of_OFT
		Error! Reference source not found., 13.3	OMA-BCAST-2007-0291-CR_Figures_for_the_token_delivery_protocol OMA-BCAST-2007-0296-CR_Order_Figures_4_layer_key_hierarchy
		B.1, B.2	OMA-BCAST-2007-0298R01-CR_Use_of_ISO_Metadata_track_for_STKM_track
		7.3.2, 10.2.2	OMA-BCAST-2007-0350R02-CR_Adapted_PDCF_Substitute_copied_text_by_references
		3.3, 10.3.4.4, 13.4.1, C.17.3	OMA-BCAST-2007-0382R01-CR_CR_CID_and_BCI_to_URI_in_XBS
		9.2	OMA-BCAST-2007-0388-CR_remove_bsdaID_in_XBS
		5.3.1, 5.3.2	
		13.3.1, 13.3, C.18	

		13.2, 13.3 **Error! Reference source not found.** **Error! Reference source not found.**, 7.3.1, 7.3.2, 8.4.2, 9.2.1, C.3, C.4, C.5 9.1.7	OMA-BCAST-2007-0410-CR_XBS_DCF_and_PDCF_branding OMA-BCAST-2007-0434-CR_Cleanup_Reference_Table OMA-BCAST-2007-0445R01-CR_ROAP_Extensibility_XBS OMA-BCAST-2007-0501-CR_XBS_bugfix_Save_Permission_defined_in_ODRL

		Many	Comment DX003
		13	Comment DX017
		Many	Comment DX018
		12.2.1.3	Comment DX019
		13.3.4.1	Comment DX021
		3.4.21	Comment DX024
		13.4.3	Comment DX026
		Error! Reference source not found., C.1.1	Comment DX027 Comment DX049
		7.5, 8.2.1	Comment DX058 Comment DX059
		A1.3	Comment DX066
		C.1	Comment DX067
		Error! Reference source not found.	Comment DX070 Comment DX071
		Many	Comment DX073
		9.1	Comment DX076
		9.2.3.1	Comment DX079
		All	Comment DX080
		Error! Reference source not found., 7	Comment DX087 Comment DX098
		Many	Comment DX100
		Many	Comment DX101
		Error! Reference source not found., 12.6, C.1.5.2.2	Comment DX102 Comment DX104 Comment DX106
		7.2, 7.5	
		7.2.1.1	
		7.2.1.2.1	
		7.2.1.3.1	
		7.2.1.3.2	
		Many	

		7.4.1	Comment DX107
		Many	Comment DX109
		7.2.2.2.1, 7.7.4.1	Comment DX112
		7.2.2.2.1	Comment DX113
		7.2.2.2.1	Comment DX114
		Many	Comment DX116
		7.2.2.2.3, 7.7.4.3	Comment DX117
		Error! Reference source not found., 7.5.1	Comment DX122
		7.5.2	Comment DX124
		7.5.5.1.1	Comment DX127
		7.6.4.1, 8.2.1, 8.4.3.3	Comment DX130
		8.2.1	Comment DX140
		8.2.4	Comment DX145
		8.2.6	Comment DX147
		9.1.6	Comment DX156
		9.3	Comment DX159
		9.3	Comment DX161
		10.2.1	Comment DX163
		Error! Reference source not found., 11.1.1	Comment DX165
		12	Comment DX166
		12.1	Comment DX167
		12.5.1	Comment DX168
		12.9.1	Comment DX170
		12.11	Comment DX171
		A.1.2	Comment DX173
		C.2.1	Comment DX177
			Comment DX180

		8.2.1, C.13	Comment XBS001
		8.2.1, C.13	Comment XBS002
		13.4.2	Comment XBS004
		13.4.1	Comment XBS005
		Error! Reference source not found., 8.2.6, 8.2.7.8, 9, 9.2.3 C.16	Comment XBS006
		8.4.3.1	Comment XBS013
		8.4.3.1	Comment XBS014
		13.4.1.2	Comment XBS017
		Error! Reference source not found.	Comment XBS024
		7.3	Comment XBS027
		8.4.2	Comment XBS028
		8.4.4	Comment XBS034
		13.4	Comment XBS036
		13.4.3	Comment XBS037
		8.2.7.4	Comment XBS039
		8.2.6	Comment XBS040
		8.2.7	Comment XBS041
		8.2.1, C.11.1	Comment XBS043
		7.5.5.1.3	Comment XBS045
		7.4.1.4	Comment XBS052
		8.4.3.1	Comment XBS054
		8.4.2	Comment XBS055
	04 May 2007	All	Cleanup in preparation for Approval as Candidate
Candidate Version OMA-DRM-XBS-V1_0	29 May 2007	n/a	Status changed to Candidate by TP TP ref# OMA-TP-2007-0129R01-INP_BCAST_V1_0_ERP_for_Candidate_approval

	11 Jul 2007	C.19.1, 10.3.3.2	Incorporated the following agreed CRs: OMA-BCAST-2007-0561-CR_Rights_Issuer_Service_MIME_Security_IANA OMA-BCAST-2007-0615-CR_Editorial_bugfix_in_XBS
	05 Sep 2007	C.18	Incorporated the following agreed CR: OMA-BCAST-2007-0702-CR_XBS_Bug_fix_Adapted_PDCF_osfm
	07 Sep 2007	8.2.7.7, 13.3.4	Incorporated the following agreed CRs: OMA-BCAST-2007-0677-CR_BCRO_system_constraint_bug_fix OMA-BCAST-2007-0662R01-CR_Correction_to_schi_box_in_adapted_PDCF
	19 Dec 2007	All	Updated with the latest template Incorporated the following CR: OMA-BCAST-2007-0893R01
Candidate Version OMA-DRM-XBS-V1_0	26 Feb 2008	n/a	Status changed to Candidate by TP TP ref# OMA-TP-2008-0042-INP_BCAST_V1_0_ERP_for_Candidate_Re_approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [IOPPROC].

B.1 SCR for XBS Clients

Note: Section numbers in the Reference column include only those sub-sections that are not specifically referenced in other Items.

Note: BCAST adaptation specifications, in which it is specified how the BCAST 1.0 enabler is implemented over a specific BDS (Broadcast Distribution System), may overrule or adapt requirements from this SCR or provide additional requirements.

Item	Function	Reference	Status	Requirement
DRM-XBS-C-001	XBS client	1	M	DRM-XBS-C-002 OR DRM-XBS-C-003 OR DRM-XBS-C-004
DRM-XBS-C-002	Interactive Device	**Error! Reference source not found.**	O	DRM-XBS-C-023 DRM-XBS-C-044
DRM-XBS-C-003	Broadcast Device	**Error! Reference source not found.**	O	DRM-XBS-C-005 AND DRM-XBS-C-006 AND DRM-XBS-C-007 AND DRM-XBS-C-008 AND DRM-XBS-C-013 AND DRM-XBS-C-018 AND DRM-XBS-C-019 AND DRM-XBS-C-022 AND DRM-XBS-C-035 AND DRM-XBS-C-036 AND DRM-XBS-C-045 AND DRM-XBS-C-046 AND DRM-XBS-C-047 AND DRM-XBS-C-048 AND DRM-XBS-C-049 AND DRM-XBS-C-050 AND DRM-XBS-C-051 AND DRM-XBS-C-052 AND DRM-XBS-C-053 AND DRM-XBS-C-054
DRM-XBS-C-004	Mixed-Mode Device	**Error! Reference source not found.**	O	DRM-XBS-C-002 AND DRM-XBS-C-003
DRM-XBS-C-005	Authentication on traffic layer, key stream layer and rights management layer	6	O	
DRM-XBS-C-006	Broadcast Device and Domain Management	7	O	

Item	Function	Reference	Status	Requirement
DRM-XBS-C-007	Common fields for binary messages	7.1.2	O	
DRM-XBS-C-008	Device Registration	7.2	O	DRM-XBS-C-009 AND DRM-XBS-C-010 AND DRM-XBS-C-011
DRM-XBS-C-009	Offline notification of detailed device data	7.2.1	O	
DRM-XBS-C-010	Offline notification of short device data	7.4.1	O	
DRM-XBS-C-011	Broadcast registration	7.2.2	O	
DRM-XBS-C-012	On-line Registration for broadcast devices	7.3	O	
DRM-XBS-C-013	Inform Registered Device Protocol	7.5	O	DRM-XBS-C-014 AND DRM-XBS-C-015 AND DRM-XBS-C-016 AND DRM-XBS-C-017
DRM-XBS-C-014	Update RI certificate	7.5.3	O	
DRM-XBS-C-015	Update DRM time	7.5.4	O	
DRM-XBS-C-016	Update contact number	7.5.5	O	
DRM-XBS-C-017	Force re-registration	7.5.2	O	
DRM-XBS-C-018	Binary messages for token handling	7.6.4	O	
DRM-XBS-C-019	Domain Management	7.7	O	DRM-XBS-C-020 AND DRM-XBS-C-021
DRM-XBS-C-020	Domain joining and leaving	7.7.2	O	
DRM-XBS-C-021	Binary Domain messages	7.7.4, 7.7.5, 7.7.6, 7.7.7	O	
DRM-XBS-C-022	Format of the Broadcast Rights Object	8.2	O	
DRM-XBS-C-023	Acquisition of Rights Objects over an Interaction Channel	8.3	O	
DRM-XBS-C-024	Save Permission	8.4	O	
DRM-XBS-C-025	Token Management	9	O	
DRM-XBS-C-029	Subscriber Groups	10	O	DRM-XBS-C-030 OR DRM-XBS-C-031
DRM-XBS-C-030	Fixed Subscriber Groups	10.3.3.1	O	DRM-XBS-C-032 AND DRM-XBS-C-033
DRM-XBS-C-031	Flexible Subscriber Groups	10.3.3.2	O	DRM-XBS-C-032
DRM-XBS-C-032	Key derivation for domains, unique devices and whole Subscriber Groups	10.3.4.1, 10.3.4.2, 10.3.4.3	O	
DRM-XBS-C-033	Zero-Message Broadcast Encryption	10.3.4.4	O	
DRM-XBS-C-034	Broadcast Support	11	M	
DRM-XBS-C-035	RI Stream Packet Format	12.5.2	O	
DRM-XBS-C-036	Rights Issuer services reception by Devices	12.11	O	

Item	Function	Reference	Status	Requirement
DRM-XBS-C-037	PDCF Adapted File Format	13	O	DRM-XBS-C-039 AND DRM-XBS-C-040
DRM-XBS-C-038	Key Info Box	13.3.1	O	
DRM-XBS-C-039	PDCF adaptation for key stream inclusion	13.3.2	O	
DRM-XBS-C-040	AES counter encryption in byte mode and salt	13.4	O	DRM-XBS-C-041 AND DRM-XBS-C-042
DRM-XBS-C-041	AES_128_BYTE_CTR	13.4.1.2	O	
DRM-XBS-C-042	OMADRMSalt Box	13.4.3	O	
DRM-XBS-C-043	Security Considerations	C.2	M	
DRM-XBS-C-044	XML schema	C.4	O	
DRM-XBS-C-045	Checksum on ARC	C.6.1	O	
DRM-XBS-C-046	Checksum on UDN	C.6.2	O	
DRM-XBS-C-047	Status and Error Message Handling	C.7	O	
DRM-XBS-C-048	Time and Date Conventions	C.8	O	
DRM-XBS-C-049	RSA Signatures under PKCS#1	C.9	O	
DRM-XBS-C-050	Tag Length Format for keyset_block	C.11	O	
DRM-XBS-C-051	Message tags	C.13	O	
DRM-XBS-C-052	Authentication	C.14	O	
DRM-XBS-C-053	Authentication of the tokens_consumed field in the token consumption data	C.15	O	
DRM-XBS-C-054	Token management by devices	C.16.2	O	
DRM-XBS-C-056	One-Way Function Trees	C.17.3	O	

B.2 SCR for XBS Servers

Note: Section numbers in the Reference column include only those sub-sections that are not specifically referenced in other Items.

Note: BCAST adaptation specifications, in which it is specified how the BCAST 1.0 enabler is implemented over a specific BDS (Broadcast Distribution System), may overrule or adapt requirements from this SCR or provide additional requirements.

Item	Function	Reference	Status	Requirement
DRM-XBS-S-001	XBS Server	1	M	DRM-XBS-S-002 OR DRM-XBS-S-003
DRM-XBS-S-002	XBS Server for Interactive device	**Error! Reference source not found.**	O	DRM-XBS-S-023 AND DRM-XBS-S-027 AND DRM-XBS-S-028 AND DRM-XBS-S-044
DRM-XBS-S-003	XBS Server for Broadcast device	**Error! Reference source not found.**	O	DRM-XBS-S-005 AND DRM-XBS-S-006 AND DRM-XBS-S-008 AND

Item	Function	Reference	Status	Requirement
				DRM-XBS-S-013 AND DRM-XBS-S-018 AND DRM-XBS-S-019 AND DRM-XBS-S-022 AND DRM-XBS-S-034 AND DRM-XBS-S-036 AND DRM-XBS-S-047 AND DRM-XBS-S-050 AND DRM-XBS-S-051 AND DRM-XBS-S-053
DRM-XBS-S-005	Authentication on traffic layer, key stream layer and rights management layer	6	O	
DRM-XBS-S-006	Broadcast Device and Domain Management	7	O	
DRM-XBS-S-007	Common fields for binary messages	7.1.2	O	
DRM-XBS-S-008	Device Registration	7.2	O	DRM-XBS-S-009 AND DRM-XBS-S-010 AND DRM-XBS-S-011
DRM-XBS-S-009	Offline notification of detailed device data	7.2.1	O	
DRM-XBS-S-010	Offline notification of short device data	7.4.1	O	
DRM-XBS-S-011	Push Device Registration	7.2.2	O	
DRM-XBS-S-012	On-line Registration for broadcast devices	7.3	O	
DRM-XBS-S-013	Inform Registered Device Protocol	7.5	O	DRM-XBS-S-014 AND DRM-XBS-S-015 AND DRM-XBS-S-016 AND DRM-XBS-S-017
DRM-XBS-S-014	Update RI certificate	7.5.3	O	
DRM-XBS-S-015	Update DRM time	7.5.4	O	
DRM-XBS-S-016	Update contact number	7.5.5	O	
DRM-XBS-S-017	Force re-registration	7.5.2	O	
DRM-XBS-S-018	Binary messages for token handling	7.6.4	O	
DRM-XBS-S-019	Domain Management	7.7	O	DRM-XBS-S-020 AND DRM-XBS-S-021
DRM-XBS-S-020	Domain joining and leaving	7.7.2	O	
DRM-XBS-S-021	Binary Domain messages	7.7.4, 7.7.5, 7.7.6, 7.7.7	O	
DRM-XBS-S-022	Format of the Broadcast Rights Object	8.2	O	
DRM-XBS-S-023	Acquisition of Rights Objects over an Interaction Channel	8.3	O	
DRM-XBS-S-024	Save Permission	8.4	M	

Item	Function	Reference	Status	Requirement
DRM-XBS-S-026	Additions to the OMA DRM 2.0 REL for Token Management	9.1	M	
DRM-XBS-S-027	Extensions to ROAP to Issue Tokens	9.2	O	
DRM-XBS-S-028	Extensions for ROAP for Reporting	9.3	O	
DRM-XBS-S-029	Subscriber Groups	10	O	DRM-XBS-S-030 OR DRM-XBS-S-031
DRM-XBS-S-030	Fixed Subscriber Groups	10.3.3.1	O	
DRM-XBS-S-031	Flexible Subscriber Groups	10.3.3.2	O	
DRM-XBS-S-032	Key derivation for domains, unique devices and whole subscriber groups	10.3.4.1, 10.3.4.2, 10.3.4.3	O	
DRM-XBS-S-033	Zero-Message Broadcast Encryption	10.3.4.4	O	
DRM-XBS-S-034	Broadcast Support	11	O	
DRM-XBS-S-036	Rights Issuer Services	12	O	
DRM-XBS-S-043	Security Considerations	C.2	M	
DRM-XBS-C-044	XML schema	C.4	O	
DRM-XBS-S-047	Status and Error Message Handling	C.7	O	
DRM-XBS-S-050	Tag Length Format for keyset_block	C.11	O	
DRM-XBS-S-051	Message tags	C.13	O	
DRM-XBS-S-053	Authentication of the tokens_consumed field in the token consumption data	C.15	O	
DRM-XBS-S-055	Token management by RIs	C.16.1	O	
DRM-XBS-S-056	One-Way Function Trees	C.17.3	O	

Appendix C.

C.1 Security Considerations (Informative)

C.1.1 Background

BCAST DRM solutions in general need to meet a number of security requirements. In particular, three requirements any BCAST DRM solution must fulfill are:

- to offer the same or equivalent cryptographic protection on BCROs as is available for ROs obtained via the standard ROAP protocol. This includes authentication, integrity checking and confidentiality of encryption keys.
- Protected Content must only be accessed by properly authenticated and authorized DRM Agents
- Permissions on Protected Content must be honored by all DRM Agents

This specification along with its accompanying document [BCAST10-ServContProt] establishes the OMA BCAST DRM system. The OMA BCAST DRM system provides the means for the secure distribution and management of Protected Content in the OMA BCAST environment, and meets the requirements specified above.

C.1.2 Confidentiality

Confidentiality ensures that data is not accessible by an unauthorized party. As stated above, protected content must only be accessible by properly authenticated and authorized BCAST DRM Agents. To achieve this goal, protected content is encrypted with content encryption keys. BCROs contain OMADRMasset objects, which in turn contain Program or Service Encryption and Authentication keys (Long Term Keys) or Content encryption keys (Short Term Keys).

C.1.3 Authentication

Authentication is the process by which a party identifies itself to another party. In case of broadcast, authentication of BCROs can be classified into two categories, namely, source authentication and group authentication. In source authentication, the RI sending the BCROs would digitally sign to unequivocally establish the origin to the BCROs to the DRM agents on the BCAST clients. However, this is not supported, as digital signing BCROs is considered expensive. Instead, BCROs are integrity protected using a symmetric key. This provides group authentication; in other words members also have access to the symmetric key and thus can modify the contents of the BCROs. However, it is considered very difficult to send traffic in some broadcast environments.

In summary, the authentication of BCROs is dependent on the implausibility of BCAST clients being able to send traffic and on symmetric keys. Note that this is in contrast to DRM v2.0 where ROs are digitally signed. BCROs afford weaker protection than ROs.

C.1.4 Integrity Protection

Data integrity protection ensures the ability to detect unauthorized modification of data. In the OMA DRM, data integrity protection, when applicable, is achieved through digital signatures on ROAP messages and Rights Objects. In case BCAST DRM, integrity protection is via symmetric keys and as described earlier, BCRO integrity protection is weaker compared to that of ROs.

C.1.5 Threat Analysis

C.1.5.1 Threat Model

Any DRM system must protect against the threat of compromise of a DRM entity (Rights Issuer, DRM Agent, Content Issuer, CA, or OCSP responder), leading to unauthorized behavior. In particular, since it may be in the interest of the user of the DRM agent to bypass the security, the DRM Agent must be robust against the "reversed" threat model. Besides protecting against the threat of a DRM entity compromise, the DRM system must protect against passive as well as active attacks.

In the following, it is assumed that an attacker is able to:

- Listen to the communication channel between a DRM Agent and a Rights Issuer, and
- Read, modify, remove, generate and inject messages in this channel.

When applicable, the case of a compromised DRM entity is also discussed.

C.1.5.2 Active Attacks

C.1.5.2.1 Message Removal

An attacker may remove any message sent between a DRM Agent and an RI. In general, this constitutes a Denial of Service attack. BCROs are repeatedly sent to protect against this type of attack and also to cover cases of a mobile device being offline, out of coverage, and finally to cover the case of packet losses.

C.1.5.2.2 Message Modification

An attacker may modify any message sent between a DRM agent and an RI.

- BCROs are integrity protected using a symmetric key. Thus, if an outside entity modifies messages, the DRM agent can easily detect message modification.
- An insider attack is plausible since the symmetric key is available to DRM agents. Considering the case of the compromised DRM entity, such entities may be able to modify BCROs and send to other uncompromised DRM agents. However, broadcasting by BCAST Devices is considered very difficult in some broadcast distribution systems, e.g., DVB. BCRO security relies on assumptions on transmission capabilities as opposed to cryptographic techniques as in case of DRMv2.0 ROs.

C.1.5.2.3 Message Insertion

An attacker may at any point insert messages into the communication channel between an RI and a DRM Agent. The attacker may also record messages and try to replay them at a later point in time.

- BCROs contain timestamps for replay protection.

C.1.5.2.4 Entity Compromise

An attacker may attempt to, physically or otherwise, compromise an entity of the DRM system.

- A compromised DRM Agent may result in the disclosure of any of the following:
  i. The DRM agent's private key
  ii. Domain keys for any Domain the DRM Agent is a member of
  iii. Rights Object Encryption Keys
  iv. Content Encryption Keys
  v. Protected Content

  It may also result in loss of integrity protection of the DRM Agent's replay cache and/or loss of protection of Rights stored internally in the DRM Agent. Further it may result in loss of DRM Time, potentially allowing permissions to be overridden or compromised RIs to pose as uncompromised.

  Failure of DRM Agent implementations to protect the above assets may seriously compromise the security of the OMA DRM system and their protection is therefore critical.

  In addition, a compromised rendering application in the DRM Agent may also result in the loss of Protected Content. The DRM Agent implementation must therefore be robust and ensure that it only provides unprotected Protected Content to trusted rendering applications.

- A compromised Rights Issuer may result in the disclosure of any of the following:
  i. The Rights Issuer's private key

ii. Domain keys for any Domain administered by the RI

iii. Rights Object Encryption Keys

iv. Content Encryption Keys

v. Protected Content

Again, the protection of these assets in RI implementations is crucial to the correct functioning of the OMA DRM.

- The effects on a PKI of a compromised CA or OCSP Responder is discussed, e.g., in [RFC3280] and [RFC2560].

The OMA DRM system relies on certificate revocation for minimizing the damages of a compromised entity. DRM Agents and RIs must always verify that the entity they are communicating with has not been compromised by checking the entity's certificate status. Further, in Domain settings, RIs may protect against undetected DRM agent compromise by regularly upgrading Domain Generations as described in Section 8 of [DRM-v2].

C.1.5.2.5 Additional Impact of Entity Compromise due to Subgroup Key Management

Compromise of BCAST 1.0 DRM agent results in the compromise of services beyond services subscribed to by the DRM agent. This is due to the collusion vulnerability in the group key management system used in this specification. Specifically, two compromised DRM agents can recover the keys corresponding to the keys of all the DRM agents within a BCAST subgroup. More TBD.

C.2 Security Considerations

C.2.1 Handling Weak Keys

When applying a cryptographic algoritm, the use of weak keys SHOULD be avoided. At the time of this writing there are no specified weak keys for use in AES. This does not imply that weak keys do not exist. If, at some point, a set of weak keys for AES is identified, the use of these weak keys SHALL be avoided and rejected within the network.

C.2.2 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/she has acquired the GRO, 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/she 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.

- The device checks periodically a particular or all RI context for expiration.

1) 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")

2) Until this OCSPresponse 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 OMA BCAST Service Guide 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 GROs, 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.

3) 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 Device SHALL support secure DRM time.

    c. Rules in GROs SHALL have precedence over the OCSP response grace period usage.

4) 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 OMA BCAST Service Guide 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 GROs, 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 OMA BCAST Service Guide. The device SHOULD parse the received OMA BCAST Service Guide 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 Section 12.8 for more details.

2) Upon power down before update the device may display a warning message that the device needs to update it's 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).

C.3 ROAP XML schema extensibility (normative)

This appendix defines extension hooks on top of the DRM 2.0 ROAP XML schema which are in line with the extensibility mechanism proposed by DRM 2.1 for forward compatibility. The extensions in the XBS XML schema [DRM20-Broadcast-Extensions-ROAP-XSD] are based on these hooks.

In the sections below, additions to the DRM 2.0 XML schema are denoted by *red text in italics*.

C.3.1 The Response Type

The abstract **Response** type is defined in [DRM-v2] Section 5.3.5 as a basis to derive ROAP responses by extension from this type. All responses contain a ***status*** attribute that indicates whether the preceding request was successful or not. To enable future extension, the XML schema below changes the data type of the **status** attribute to "string". The currently specified status messages are defined in the **Status** type defined listed in Section 5.3.6 of [DRM-v2].

```
<complexType name="Response" abstract="true">
   <attribute name="status" type="string" use="required"/>
   <attribute name="errorMessage" type="string"/>
   <attribute name="errorRedirectURL" type="anyURI"/>
</complexType >
```

C.3.2 The ExtensionContainer type

The **ExtensionContainer** type inherits from the **Extension** type defined in [DRM-v2] Section 5.3.7 may be sent as an extension with any ROAP message, potentially next to other extensions (including other

ExtensionContainers). An ExtensionContainer may contain as child elements any currently optional or currently unknown (because defined in a subsequent versions of OMA DRM) ROAP features supported by a Device or an RI.

The extensions inside an **<ExtensionContainer>** may be a mixture of supported, unsupported and unknown extensions to the receiving party. If the **<ExtensionContainer>** is marked as **non-critical**, then the receiving party MUST disregard any unsupported or unknown children. Extensions inside a non-critical ExtensionContainer that are supported by the receiving party MUST be handled as specified in this document.

If the ExtensionContainer is marked as **critical** and it contains an unknown or unsupported child element, then:

- a receiving RI MUST respond with an UnknownCriticalExtension-status to the Device
- a receiving Device MUST discard the ROAP PDU

```
<complexType name="ExtensionContainer">
  <complexContent>
    <extension base="roap:Extension">
      <sequence>
        <any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

C.3.3 Extending the Rights Object Payload type

The **ROPayload** type defined in [DRM-v2] Section 5.3.7 is extended as given below.

Further elements MAY be included into the ROPayload after the **<encKey>** element. Devices MUST disregard any unknown elements.

```
<!-- Rights Object Definitions -->
<complexType name="ROPayload">
  <sequence>
    <element name="riID" type="roap:Identifier"/>
    <element name="rights" type="o-ex:rightsType"/>
    <element name="signature" type="ds:SignatureType" minOccurs="0"/>
    <element name="timeStamp" type="dateTime" minOccurs="0"/>
    <element name="encKey" type="xenc:EncryptedKeyType"/>
    <any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="version" type="roap:Version" use="required" />
  <attribute name="id" type="ID" use="required" />
  <attribute name="stateful" type="boolean"/>
  <attribute name="domainRO" type="boolean"/>
  <attribute name="riURL" type="anyURI"/>
</complexType>
```

C.3.4 Extending the ROAP Trigger type

The **ROAPTrigger** type has been defined in [DRM-v2] Section 5.2.1. In this section, the ROAP Trigger types are reformulated keeping the same semantics as the original schema but allowing for easier integration of future extensions. Based on the reformulation, an extensible trigger is added to this definition.

```
<complexType name="BasicRoapTrigger">
  <sequence>
    <element name="riID" type="roap:Identifier"/>
```

```
    <element name="riAlias" type="string" minOccurs="0"/>
    <element name="nonce" type="roap:Nonce" minOccurs="0"/>
    <element name="roapURL" type="anyURI"/>
  </sequence>
  <attribute name="id" type="ID"/>
</complexType>

<complexType name="DomainTrigger">
  <complexContent>
    <extension base="roap:BasicRoapTrigger">
      <sequence>
        <element name="domainID" type="roap:DomainIdentifier" minOccurs="0"/>
        <element name="domainAlias" type="string" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="ROAcquisitionTrigger">
  <complexContent>
    <extension base="roap:DomainTrigger">
      <sequence>
        <sequence maxOccurs="unbounded">
          <element name="roID" type="ID"/>
          <element name="roAlias" type="string" minOccurs="0"/>
          <element name="contentID" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="ExtendedRoapTrigger">
    <complexContent>
      <extension base="roap:BasicRoapTrigger">
        <sequence>
          <any  minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
        </sequence>
        <attribute name="type" type="string" use=”required”/>
      </extension>
    </complexContent>
</complexType>

<!-- ROAP trigger -->
<element name="roapTrigger" type="roap:RoapTrigger"/>
<complexType name="RoapTrigger">
  <annotation>
    <documentation xml:lang="en">
      Message used to trigger the device to initiate a Rights Object Acquisition Protocol.
    </documentation>
  </annotation>
  <sequence>
    <choice>
      <element name="registrationRequest" type="roap:RegistrationRequestTrigger"/>
      <element name="roAcquisition" type="roap:ROAcquisitionTrigger"/>
      <element name="joinDomain" type="roap:DomainTrigger"/>
      <element name="leaveDomain" type="roap:DomainTrigger"/>
```

```
    <element name="extendedTrigger" type="roap:ExtendedRoapTrigger"/>
  </choice>
  <element name="signature" type="ds:SignatureType" minOccurs="0"/>
  <element name="encKey" type="xenc:EncryptedKeyType" minOccurs="0"/>
 </sequence>
 <attribute name="version" type="roap:Version"/>
 <attribute name=”proxy” type=”boolean”/>
</complexType>
```

Future versions of OMA DRM MAY define additional ROAP triggers that can be received by implementations of this version of OMA DRM. In this case the **<roapTrigger>** element carries an **<extendedTrigger>** element, containing details of the requested protocol. The **<extendedTrigger>** element SHALL validate against the ExtendedRoapTrigger type defined in this specification. It SHALL contain a **type** attribute signalling the protocol that is triggerred by the **<extendedTrigger>** element. The **type** attribute is used to determine whether or not this trigger is known. Unknown triggers MUST be disregarded.

The **ExtendedRoapTrigger** type provides a forward-compatible structure forhe **ExtendedRoapTrigger** type defines the structure all future ROAP triggers SHALL be valid against in order to support forward compatibility. The **<any>** wildcard in this structure defines the location for all future extensions by additional elements. To signal the initiation of a new protocol, future specifications may introduce new extended ROAP triggers. For such future triggers, either the existing **ExtendedRoapTrigger** type contains all the needed information, or some additional elements are needed to be included. In the former case, the currently defined **ExtendedRoapTrigger** type can be re-used by defining a new fixed value for the **type** attribute to signal the triggered protocol. In the latter case, it is advised to derive a new type (e.g. by extending the **BasicRoapTrigger** type), adding the needed elements and defining a **type** attribute with a fixed value to signal the triggered protocol.

The definition of the **TokenAcquisitionTrigger** type in Section 9.2.1 provides an example of how to define extended triggers with additional elements.

C.4 XML schema (normative)

This specification reuses and extends the OMA DRM v2.0 ROAP protocol suite defined in [DRM-v2]. Extensions are compliant with the extensibility mechanism defined in Appendix C.3 of this specification.

ROAP PDUs used by or defined in this specification:

- SHALL conform to XML schema [DRM20-Broadcast-Extensions-ROAP-XSD].
- SHOULD include in top-level element the “xsi:schemaLocation” attribute associating "urn:oma:bac:dldrm:roap-1.0" namespace with a valid URL pointing to schema location of [DRM20-Broadcast-Extensions-ROAP-XSD] in OMNA repository

All ROAP messages conforming to this specification SHALL set the **version** attribute to “1.0”.
The ROPayload **version** attribute SHALL be set to “1.0”.
The REL messages SHALL have version “2.0”.

Note that [DRM20-Broadcast-Extensions-ROAP-XSD] imports from (and depends on) [DRM20-Broadcast-Extensions-OMADD-XSD], which holds the extensions to DRM REL v2.0.

C.5 Forward Compatibility (Informative)

It is expected that OMA will continue to develop its DRM enabler to enable new features on new devices and services. At the same time implementations of this version of the OMA DRM enabler will be used in the market. This means that users will own and use Devices that implement different versions of OMA DRM with the same services and/or in the same domain. For this purpose, this enabler specifies where the protocols and datatypes may be extended and the behaviour of the DRMAgent in case it encounters unknown extensions.

The XML schema defined in Appendix C.3 explicitly enables forward compatibility using wildcards at selected locations. Future version of OMA DRM and also of other enablers that use OMA DRM are advised to specify their extensions to OMA DRM ONLY at the location of these wildcards, using the types provided in the schema. In this way a message or data structure that is valid in the future version of OMA DRM or the other enabler, will also validate against the XML schema defined in Appendix C.3. Every conformant implementation will be able to parse the message or data structure and correctly deal with its content. The behavior of a DRM Agent that receives a message or data structure that does NOT validate against the XML schema specified in Appendix C.3 is undefined. It is possible that the message or data structure is discarded completely.

This appendix contains a number of examples of how the wildcards can be used for future extensions that are correctly handled by OMA DRM implementations. This appendix does not contain examples on how such future extensions may be specified in future versions by using XML-schema and/or specification text. The extensions to OMA DRM 2.0 defined in this specification (TokenAcquisitionTrigger, broadcastRegistration, broadcastRegistrationRequest) may serve as examples.

C.5.1 ROPayload with future extensions

This is an example of a possible future ROPayload. The future modifications are marked in a **bold** typeface. In this example, the ROPayload:

1. is of version 2.3
2. contains in addition to a play-permission (already define in OMA DRM 2.0) a “NewKindOfPermission”-permission (unknown in OMA DRM 2.0)
3. some additional elements in the ROPayload, appended after all elements known from OMA DRM 2.0.

Implementations of OMA DRM 2.0 are expected to disregard the unknown permission and additional elements but to correctly handle the ROPayload and thus potentially grant the known play permission.

```
<roap:protectedRO
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
 xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <roap:ro id="n8yu98hy0e2109eu09ewf09u" domainRO="true" version="2.3" riURL="http://www.ROs-r-us.com">
  <riID>
   <keyIdentifier xsi:type="roap:X509SPKIHash">
    <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
   </keyIdentifier>
  </riID>
  <rights o-ex:id="REL1">
   <o-ex:context>
    <o-dd:version>2.3</o-dd:version>
    <o-dd:uid>RightsObjectID</o-dd:uid>
   </o-ex:context>
   <o-ex:agreement>
    <o-ex:asset>
     <o-ex:context>
      <o-dd:uid>ContentID</o-dd:uid>
     </o-ex:context>
     <o-ex:digest>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>bLLLc+Um/5/NvmYKiHDLaErK0fk=</ds:DigestValue>
     </o-ex:digest>
     <ds:KeyInfo>
      <xenc:EncryptedKey>
```

```
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
         <ds:KeyInfo>
           <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
         </ds:KeyInfo>
         <xenc:CipherData>
          <xenc:CipherValue>EncryptedCEK</xenc:CipherValue>
         </xenc:CipherData>
        </xenc:EncryptedKey>
       </ds:KeyInfo>
     </o-ex:asset>
     <o-ex:permission>
       <o-dd:play/>
     </o-ex:permission>
     <o-ex:permission>
       <NewKindOfPermission/>
     </o-ex:permission>
   </o-ex:agreement>
  </rights>
  <signature>
   <ds:SignedInfo>
     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
           <ds:SignatureMethod
      Algorithm="http://www. rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsa-pss-default"/>
     <ds:Reference URI="#REL1">
       <ds:Transforms>
         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
       </ds:Transforms>
       <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <ds:DigestValue>sIo5hb+id8JtuOMNKs12=drf5+3df= </ds:DigestValue>
     </ds:Reference>
   </ds:SignedInfo>
   <ds:SignatureValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:SignatureValue>
   <ds:KeyInfo>
     <roap:X509SPKIHash>
       <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
     </roap:X509SPKIHash>
   </ds:KeyInfo>
  </signature>
  <encKey Id="K_MAC_and_K_REK">
   <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
   <ds:KeyInfo>
     <roap:domainID>Domain-XYZ-001</roap:domainID>
   </ds:KeyInfo>
   <xenc:CipherData>
     <xenc:CipherValue>32fdsorew9ufdsoi09ufdskrew9urew0uderty5346wq</xenc:CipherValue>
   </xenc:CipherData>
  </encKey>
  <NewUnknownFeatureDefined in 2.1/>
  <NewUnknownFeatureDefined in 2.3/>
</roap:ro>
<mac>
  <ds:SignedInfo>
   <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <ds:SignatureMethod
     Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
   <ds:Reference URI="#n8yu98hy0e2109eu09ewf09u">
```

```
      <ds:Transforms>
        <ds:Transform Algorithm=http://www.w3.org/2001/10/xml-exc-c14n#/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>slo5hb+id8JtuOMNKs12=drf5+3df=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
  </ds:KeyInfo>
 </mac>
</roap:protectedRO>
```

C.5.2 ROAP-PDU with future extensions

This is an example of a possible future joinDomainResponse. The modifications are marked in a **bold** typeface. In this example, the ROAP-PDU contains two new extensions, unknown in OMA DRM 2.0. Implementations of OMA DRM 2.0 are expected to recognize these extensions as extensions of an unknown type. Since one of the extensions is marked as critical, OMA DRM 2.0 implementation must discard the ROAP PDU.

```
<roap:joinDomainResponse
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 status="Success">
  <deviceID>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>vXENc+Um/9/NvmYKiHDLaErK0gk=</hash>
    </keyIdentifier>
  </deviceID>
  <riID>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
    </keyIdentifier>
  </riID>
  <nonce>32efd34de39sdwefqwer</nonce>
  <domainInfo>
    <notAfter>2004-12-22T03:02:00Z</notAfter>
    <roap:domainKey>
      <encKey Id="Domain-XYZ-001">
        <xenc:EncryptionMethod
     Algorithm="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsaes-kem-kdf2-kw-aes128"/>
        <ds:KeyInfo>
          <roap:X509SPKIHash>
            <hash>vXENc+Um/9/NvmYKiHDLaErK0gk=</hash>
          </roap:X509SPKIHash>
        </ds:KeyInfo>
        <xenc:CipherData>

<xenc:CipherValue>231jks231dkdwkj3jk321kj321j321kj423j342h213j321jh321jh2134jhk3211fdslfdsopfespjoefwo
pjsfdpojvct4w925342a</xenc:CipherValue>
        </xenc:CipherData>
      </encKey>
      <riID>
        <keyIdentifier xsi:type="roap:X509SPKIHash">
```

```
            <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
          </keyIdentifier>
        </riID>
        <mac>ewqrewoewfewohffohr3209832r3</mac>
      </roap:/domainKey>
    </domainInfo>
    <certificateChain>
      <certificate>MIIB223121234567</certificate>
      <certificate>MIIB834124312431</certificate>
    </certificateChain>
    <ocspResponse>miibewqoidpoidsa</ocspResponse>
    <extensions>
      <extension xsi:type="ExtensionContainer" critical=”true”>
                  <newCriticalUnknownExtensionElement/>
      </extension>
      <extension xsi:type="ExtensionContainer" critical=”false”/>
                  <newNonCriticalUnknownTypeofExtension>
      </extension>
    </extensions>
    <signature>d93e5fue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:joinDomainResponse>
```

C.5.3 ROAP Response with future status code

This is an example of a possible future leaveDomainResponse. The modifications are marked in a **bold** typeface. In this example, the leaveDomainResponse is unsuccesful, for a reason not specified by OMA DRM 2.0. RI implementations of this specification are expected to treat this as an “Abort” error code.

```
<roap:leaveDomainResponse
xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
status="NoSuccessForFutureReason">
</roap:leaveDomainResponse>
```

C.5.4 New type of ROAP Trigger

This is an example of a possible future ROAP Trigger. The modifications are marked in a **bold** typeface. Implementations of OMA DRM 2.0 are expected to disregard unknown triggers.

```
<roap:roapTrigger
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <extendedTrigger type=”someTriggerName”>
    <riID>
      <keyIdentifier xsi:type="roap:X509SPKIHash">
        <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
      </keyIdentifier>
    </riID>
    <roapURL>http://ri.example.com/ro.cgi?tid=qw683hgew7d</roapURL>
    <FirstAdditionalElement/>
    <SecondAdditionalElement/>
  </extendedTrigger>
  <signature>
```

```
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <ds:Reference URI="#de32r23r4">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue> slo5hb+id8JtuOMNKs12=drf5+3df=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:RetrievalMethod URI="#K_MAC"/>
  </ds:KeyInfo>
 </signature>
 <encKey Id="K_MAC">
  <xenc:EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
  <ds:KeyInfo>
    <roap:domainID>Domain-XYZ-001</roap:domainID>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>32fdsorew9ufdsoi09ufdskrew9urew0uderty5346wq</xenc:CipherValue>
  </xenc:CipherData>
 </encKey>
</roap:roapTrigger>
```

C.6 Checksum Algorithms

According to empirical research by [VERHOEF_1969] the likelihood of errors is expressed as:

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 sections explain a method to detect other errors.

C.6.1 Checksum on ARC

Definition:

The checksum on the ARC is calculated by F_{ARC}

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,...,*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,...,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 & S11
	3	**6**	**4**	**2**	**6**	**8**	**2**	**1**	**2**	**4**	**0**	**1**	line for C12 & 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***

C.6.2 Checksum on UDN

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 equitions: a string $(c1,c2...,cn)$ with elements from Zp is a codeword if and only if it satisfies the following equations:

for $i = 1,2,...r, \sum_{j=1}^{n} aj^{(i)} cj \equiv 0 (\mathrm{mod}\ p)$

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:

10*c1 + 1*c2 + 9*c3...+ 8*c17 = 0 (modulo 11)
0*c1 + 1*c2 + 0*c3...+ 7*c17 = 0 (modulo 11)
1*c1 + 0*c2 + 1*c3...+ 8*c17 = 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**
H3	1	0	1	0	1	0	1	0	1	0	1	2	3	4	5	7	8	1	0	0
	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,...,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 = - (10*c1 + 1*c2 + 9*c3 +...+ 8*c17) modulo 11

c19 = - (0*c1 + 1*c2 + 0*c3 +...+ 7*c17) modulo 11
c20 = - (1*c1 + 0*c2 + 1*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:

c18 = (10*c1 + 1*c2 + 9*c3 +...+ 1*c20) modulo 11
c19 = (0*c1 + 1*c2 + 0*c3 +...+ 1*c19) modulo 11
c20 = (1*c1 + 0*c2 + 1*c3 +...+ 1*c18) 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:

N.b.: following example illustrates the use of the algorithm on valid UDN as input number :

position (n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
inputnumber	**8**	**5**	**6**	**2**	**8**	**7**	**0**	**1**	**2**	**1**	**5**	**3**	**2**	**9**	**5**	**6**	**7**				
matrix H3	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**2**	**3**	**4**	**5**	**7**	**8**	**1**	**0**	**0**	line for C20 & S20
	0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**2**	**3**	**4**	**6**	**7**	**0**	**1**	**0**	line for C19 & S19
	10	**1**	**9**	**2**	**8**	**3**	**7**	**4**	**6**	**5**	**4**	**5**	**7**	**10**	**3**	**2**	**8**	**0**	**0**	**1**	line for C18 & S18

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	

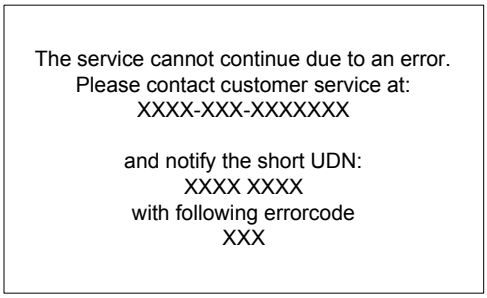
decoding																					checkdigit = +sum(n1..n18 or n19 or n20) mod 11
S18	80	5	54	4	64	21	0	4	12	5	20	15	14	90	15	12	56	0	0	2	***0***
S19	0	5	0	2	0	7	0	1	0	1	0	3	4	27	20	36	49	0	10	0	***0***
S20	8	0	6	0	8	0	0	0	2	0	5	6	6	36	25	42	56	9	0	0	***0***

C.7 Status and Error Message Handling

This section 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 user5.



An example dialogue showing an error

Figure 35: Sample notification display

Note: The error codes should be displayed as a three digit decimal number. Refer to Table 49 for an overview of possible error codes.

Table 49: Status/Error codes

Status / Error	value$_{(h)}$	comment
Success	0x00	
NotSupported	0x03	
DeviceTimeError	0x0B	
InvalidDomain	0x0D	
DomainFull	0x0E	
ForceInteractiveChannel	0x11	
ForceOobChannel	0x12	
Reserved for future use	0x11-0xFF	

NotSupported: indicates the Device made a request for a feature currently not supported by the RI.

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.

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.

ForceInteractiveChannel: indicates that the RI forces a Mixed-mode Device to exclusively use its interaction channel and not its OOB channel.

ForceOobChannel: indicates that the RI forces a Mixed-mode Device to exclusively use its OOB channel and not its interaction channel.

[5] 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).

C.8 Time and Date Conventions

C.8.1 Specification of the mjdutc format

The mjdutc format is a 40-bit field that represents date and time.

The first (left) 16 bits SHALL contain the 16 least significant bits of 'MJD' from Annex C of [ETSI EN 300 468] V1.7.1.

If the first (left) 16 bits of the mjdutc field are less than 15079, these 16 bits SHALL represent a value for MJD of 65536 + the value of the first (left) bits of the mjdutc field.

NOTE The first (left) 16 bits of the mjdutc field represent the inclusive dates 1900 March 1 to 2081 April 25.

The last (right) 24 bits of the mjdutc field represent time. They SHALL be coded as 6 digits in 4-bit Binary Coded Decimal (BCD).

EXAMPLE 93/10/13 12:45:00 is coded as "0xC079124500".

C.8.2 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 50: 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

C.9 RSA Signatures under PKCS#1

RSA signatures SHALL be made as described by the implementation guidelines of [PKCS #1] v2.1: RSA Cryptography Standard, RSA Laboratories, June 14, 2002.

The scheme SHALL be RSA + SHA1. There are two choices described in the [PKCS#1] 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, this specification SHALL also use RSASSA-PSS to sign the binary messages for broadcast mode of operation.

C.10 C-Style Types

Following abbreviated types are used in the document:

type name	description	remark
bslbf	bit serial leftmost bit first	
mjdutc	modified julian date UTC	Refer to Appendix C.8 for the specification of this type.

uimsbf	unsigned integer most significant bit first	

All fields marked as reserved for future use SHALL be set to the value 0, when not used.

All fields marked as reserved SHALL be set to value 0, and never to any other value.

C.11 Tag Length Format for keyset_block

C.11.1 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 51: 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	1000	
flexible_device_data()	1001	
FSGK block	1010	
reserved for future use	1011-1111	not used in this version of the spec

Note:

- The keyset items SHALL be included in the order of the table above.
- The keyset SHALL include only 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 in bytes.

- in case the preceding <tag> value indicates flexible_device_data this field represents the length of the flexible_device_data in bytes.
- in case the preceding <tag> value indicates an FSGK block, this field represents the length of the FSGK block in bytes. The <length> field indicates the type of the FSGKs as shown in Table 52.
    - If the zero-message broadcast encryption scheme is used, the FSGKs are stored in the FSGK block in descending order from root to leaf (the root itself not included). The maximum size of the keyset_item of 1023 bytes is sufficient to hold a maximum of 31 keys of length 256 bits each. This definition is valid for zero-message broadcast encryption method.
    - If the One-Way Function Tree (OFT) scheme is used (see Appendix C.17.3), the FSGK block contains the blinded keys and the device key. No blinded keys for the root level and the first level under the root are transmitted. The other blinded keys follow in the order from root to leaf, after which the device key follows.
    - When Flexible Subscriber Groups are used without a broadcast encryption scheme, the FGSK block contains only one FSGK, which is used as DEK where necessary.

If keyset_item == 0001 (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 = groupsize, then $n = \log_2 m$, where n is number of SGKs that have to be transmitted to the Device by the registration process. Possible positions for these SGKs in the tree are 2^{n+1}-2 (the root cannot contain an SGK). Therefore parameter "position" is expressed with 10 bits to express 1023 nodes in a tree. The MSB will be used as binary indicator to indicate if the SGK position is an internal node (MSB = 0) or a leaf (MSB = 1). Bit positions 2..10 (from left to right LSB) are used in binary format as an indication of the node and leaf position. Internal nodes and leafs SHALL be numbered according to Appendix C.17.1. The leaf keys are numbered from left to right, starting at the binary value 1000000000.

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 bits, as described in C.11.2.

length values:

This is a 3 bit field (bslbf) indicating the length of a keyset item. This field SHALL be present for all keyset items except for the LBDF keyset item and the flexible_device_data item.

Table 52: Defined length values

(key)length prescriber	Length $_{(b)}$	remark
128 bit AES	000	
192 bit AES	001	
256 bit AES	010	
5 byte Eurocrypt	011	
6 byte	100	SBDF
reserved for future use	101-111	not used in this version of the specification

Note: In case of the LBDF there is no extra length field, since the length value is indicated by the clarifier.

format of flexible_device_data

If a Device is assigned to a Flexible Subscriber Group, the flexible_device_data() structure is included. It contains information about the Flexible Subscriber Group and has the following format:

Table 53: Format of flexible_device_data()

Field	Length	Type
flexible_device_data() {		
flexible_group_address	**variable**	**OMADRMGroupAddress()**
flexible_position_in_group	**variable**	**OMADRMPositionInGroup()**
flexible_group_size_indicator	**5**	**uimsbf**
broadcast_encryption_scheme	**2**	**uimsbf**
}		

flexible_group_size_indicator: when the device is assigned to a Flexible Subscriber Group, this 5-bit field indicates the size of that Subscriber Group. When flexible_group_size_indicator contains a value *k*, the Subscriber Group has a size of 2^k Devices.

broadcast_encryption_scheme: indicates which broadcast encryption scheme is used by the RI. The number of Flexible Subscriber Group Keys (FSGKs) depends on the size of the Flexible Subscriber Group and the used broadcast encryption scheme. Table 54 explains this in more detail.

Table 54: The meaning of broadcast_encryption_scheme

value of broadcast_ encryption_ scheme	name of the broadcast encryption scheme used	number of FSGKs by a Flexible Subscriber Group of size 2^k.	value of flexible_bitmask_ present	value of nodenumber_ present
00	**no broadcast encryption scheme used (DEK equal to the sole FSGK)**	**1**	**TRUE**	**FALSE**
01	**zero-message broadcast encryption (DEK calculated by the method of Section 10.3.4.4)**	***k***	**TRUE**	**FALSE**
10	**one-way function tree (DEK calculated by the method of Appendix C.17.3)**	***k***	**FALSE**	**TRUE**
11	**reserved for future use**	**-**	**-**	**-**

TLF examples

E.g.1: A 5 byte Eurocrypt address implementing the UDF is coded as:

<0011> <011> <UDF>

E.g.2: A 48 bits SBDF address is coded as:

<0101> <100> <SBDF>

E.g.3: A LBDF address of 105 bytes is coded as:
<0110> <1101001000> <LBDF>

E.g.4: A 128 but AES key implementing the UGK is coded as:
<0000> <000> <UGK>

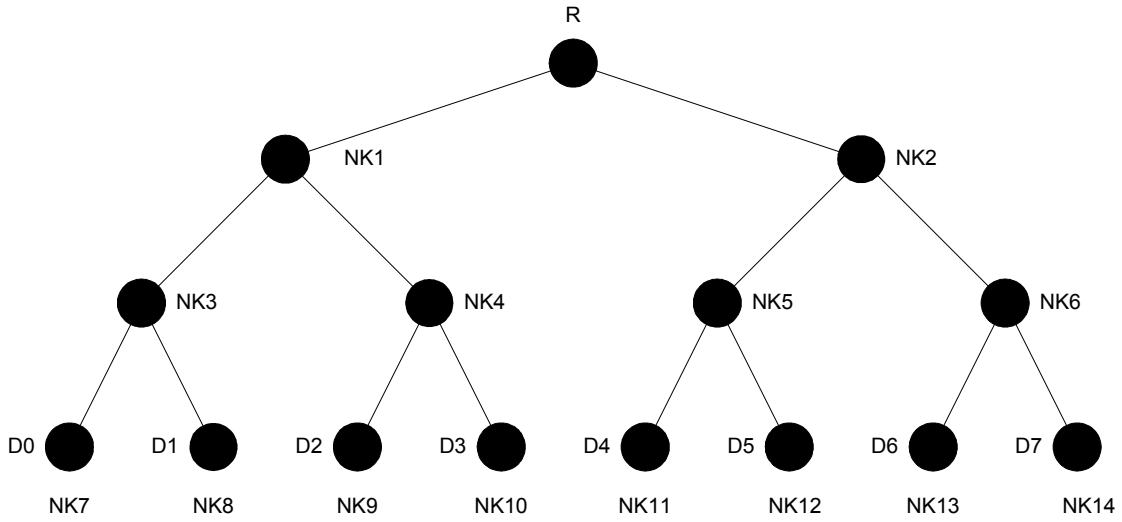


Figure 36: Sample tree with correct node and device numbering

E.g.5: A 128 bit AES key implementing the SGK on node position NK5 in Figure 36 is coded as:
<0001> <0000000101> <000> <SGK>

E.g.6: A 128 bit AES key implementing the SGK on node position NK7 (i.e. D0) in Figure 36 is coded as:
<0001> <1000000000> <000> <SGK>

E.g.7: A 128 bit AES key implementing the SGK on leaf position D300 in a Fixed Subscriber Group of size 512 is coded as
<0001> <1100101100> <000> <SGK>

C.11.2 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 [RFC 3629]. 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.

C.12 session_key length and surplus_block length computation (Informative)

The session_key is used in two registration messages (device_registration_response, domain_registration_response). The surplus_block is used in the device_registration_response message. This section provides details on the computation of their lengths during message generation and consumption.

The following are definitions of parameters that are used in these computations.

SKlen = length of session_key in bits (128, 196 or 256 bit)

SKBlen = length of encrypted sessionkey_block in bits (1024, 2048 or 4096 bit)

SKBPLlen = length of payload of sessionkey_block in bits (= length of unencrypted sessionkey_block)

SPBlen = length of encrypted surplus_block in bits

UKSBlen = length of unencrypted keyset_block in bits

KSBlen = length of encrypted keyset_block using NIST key-wrap in bits

Computation of surplus_block length during message generation

Using AES key wrap, encryption of n 64-bit plaintext blocks yields n+1 64-bit ciphertext blocks. Therefore:

KSBlen = (UKSBlen / 64) * 64 + 64 bit

The sessionkey_block uses PKCS#1 encryption, using the RSAES-OAEP algorithm and the SHA-1 hash. This means that a sessionkey_block with length SKBlen bits has a payload of:

SKBPLlen = SKBlen – 2*80 – 2*8 bit = SKBlen – 176 bit

Therefore, the length of the surplus_block in bits is:

SPBlen = 0 if (SKlen + KSBlen) ≤ SKBlen – 176

SPBlen = SKlen + KSBlen – (SKBlen – 176) if (SKlen + KSBlen) > SKBlen – 176

Computation of the session_key length during reception

The presence or absence of the surplus_block is signalled by the flag surplus_block_flag in the device_registration_response message. The following two paragraphs specify the computation of the length of the session key in both cases.

Surplus_block not present (SPBlen equal to 0)

PKCS#1 using the RSAES-OAEP algorithm accepts a byte string as input. If the number of bytes is less than the payload, byte padding occurs during the encryption process. The decryption process removes this byte padding, so the length in bytes of the input for encryption is known after decoding. Since SKBlen + KSBlen is always a multiple of 64 bits, this means that the exact value of SKBlen + KSBlen is known after PKCS#1 decryption. The registration message contains the key_set_block parameter, which is equal to KSBlen. With a simple subtraction, the length of the session key (Sklen) can be retrieved.

Surplus_block is present (SPBlen unequal to 0)

The total length of the registration message can be retrieved from the RI stream that carried the specific message, see Section 12.5.2. From the total length, the length of the surplus_block can be retrieved (SPBlen). The registration message contains the key_set_block parameter, which is equal to KSBlen. The length of the session_key can be computed as follows:

Sklen = SKBPLlen + SPBlen – KSBlen = SKBlen – 176 + SPBlen – KSBlen

See also the following figure.

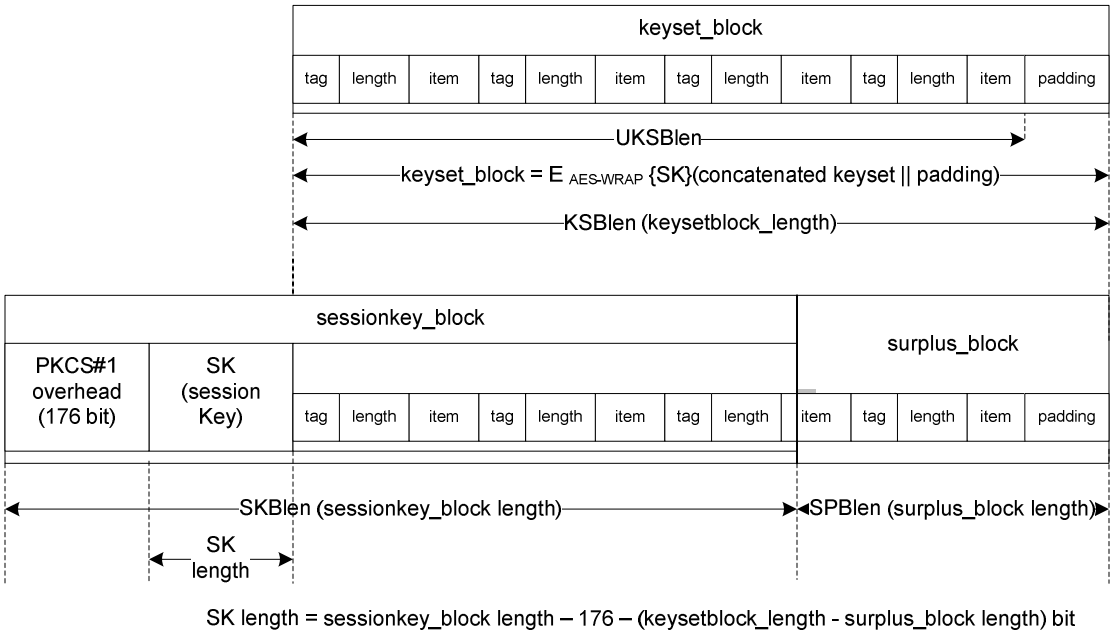


Figure 37: Diagram of keyset_block, session_key_block and surplus_block

C.13 Message Tag and Protocol Version Overview

The messages that are defined in this specification SHALL use following message_tag and protocol version values:

Table 55: message_tag and protocol_version overview

Message	message_tag	protocol_version	Section
device_registration_response()	0x01	0	7.2.2.2.2

domain_registration_response()	0x02	0	7.7.4.1.1
domain_update_response()	0x03	0	7.7.5.2
re_register_msg()	0x11	0	7.5.2.1.2
update_ri_certificate_msg()	0x12	0	7.5.3.1
update_drmtime_msg()	0x13	0	7.5.4.1.2
update_contact_number_msg()	0x14	0	7.5.5.1.2
join_domain_msg()	0x15	0	7.7.6
leave_domain_msg()	0x17	0	7.7.7
OMADRMBroadcastRightsObject	0x20	0	8.2.1
OMADRMBroadcastRightsObjectSigned	0x21	0	8.2.1
token_delivery_response()	0x30	0	7.6.4.2

C.14 Authentication

C.14.1 Authentication for IPsec

IPsec authentication is specified in [BCAST10-ServContProt]. It shares much functionality with the authentication specified for different purposes in this document.

C.14.2 Authentication for STKMs

STKM authentication is specified in [BCAST10-ServContProt]. It shares much functionality with the authentication specified for different purposes in this document.

C.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 AES-wrap [AES_WRAP], SHALL be transported in a RO as separate ds:KeyInfo elements in the <asset> fragment of the Rights Object. The relevant fragment of the <asset> element of a service RO is illustrated in the following figure:

```
<o-ex:asset o-ex:id="asset ID">

[…]

 <ds:KeyInfo>
  <xenc:EncryptedKey>
   <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
   <ds:KeyInfo>
    <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
   </ds:KeyInfo>
   <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"/>
   <ds:KeyInfo>
    <ds:RetrievalMethod URI="#K_MAC_and_K_REC"/>
   </ds:KeyInfo>
```

```
    <xenc:CipherData>
      <xenc:CipherValue>encrypted_service_authentication_seed</xenc:CipherValue>
    </xenc:CipherData>
   </xenc:EncryptedKey>
  </ds:KeyInfo>
</o-ex:asset>
```

Figure 38: <asset> fragment for a RO carrying SEK and SAS.

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 RO, which SHALL be constructed as follows:

service_authentication_seed_id = asset_idD + "_authSeed"

Similarly, the <asset> element of a program RO SHALL contain:

```
<o-ex:asset o-ex:id="asset ID">

[…]

  <ds:KeyInfo>
   <xenc:EncryptedKey>
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
    <ds:KeyInfo>
     <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
    </ds:KeyInfo>
    <xenc:CipherData>
     <xenc:CipherValue>encrypted_program_encryption_key</xenc:CipherValue>
    </xenc:CipherData>
   </xenc:EncryptedKey>
  </ds:KeyInfo>
  <ds:KeyInfo Id="program_authentication_seed_id" >
   <xenc:EncryptedKey>
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
    <ds:KeyInfo>
     <ds:RetrievalMethod Uri="#K_MAC_and_K_REK"/>
    </ds:KeyInfo>
    <xenc:CipherData>
     <xenc:CipherValue>encrypted_program_authentication_seed</xenc:CipherValue>
    </xenc:CipherData>
   </xenc:EncryptedKey>
  </ds:KeyInfo>
```

```
</o-ex:asset>
```

Figure 39: <asset> fragment for an RO carrying PEK and PAS.

encrypted_program_encryption_key =

$$E\{REK\}(SEK) = AES\_wrap\{REK\}(PEK)$$

encrypted_program_authentication_seed =

$$E\{REK\}(SAS) = AES\_wrap\{REK\}(PAS)$$

where PEK and PAS are both an AES key of 128 bits and program_authentication_seed_id is a unique identification of the authentication seed KeyInfo element within the RO, which SHALL be constructed as follows:

program_authentication_seed_id = asset_id + "_authSeed"

C.14.2.2 Transport of SEAK and PEAK in BCROs

The encryption keys and authentication keys (SEAK and PEAK) SHALL be 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 =

$$E\{DEK\}(SEAK) = AES\_CBC\{DEK\}(SEK << 128) \,||\, SAS)$$

where SEK and SAS are both an AES key of 128 bits.

and encrypted_program_encryption_authentication_key =

$$E\{DEK\}(PEAK) = AES\_CBC\{DEK\}(PEK << 128) \,||\, PAS)$$

where PEK and PAS are both an AES key of 128 bits.

C.14.3 Authentication of BCROs

BCROs MAY contain one MAC field which is used to authenticate the message and to protect the integrity of the message.

The authentication key SHALL be 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 Section 7.2.2.2.3 for details.

Refer to C.14.5 for details on f-auth.

The BAK SHALL be used in the MAC generation / verification of the BCRO if the BCRO is integrity protected with a MAC. The algorithm used to calculate the MAC field SHALL be HMAC-SHA1-96 according to [FIPS 198] and [RFC2104], using a authentication key of 160 bit.

C.14.4 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 SHALL be generated from the RIAK:

$$TDRMAK = f_{auth}\{RIAK\}(CONSTANT\_TDRM)$$

where:

CONSTANT_TDRM = 0x05050505050505050505050505050505 (120 bit)

Note: To obtain the RIAK the device needs have been equipped with a valid keyset. Refer to 0 for details.

Refer to C.14.5 for details on f-auth.

The TDRMAK SHALL be used in the MAC generation / verification of the token delivery response message. The algorithm used to calculate the MAC field SHALL be HMAC-SHA1-96 according to [FIPS 198] and [RFC2104], using a authentication key of 160 bit.

C.14.5 General Authentication Mechanism

The function F-auth SHALL consist of the following steps:

1. Denote by PRF{key}(text) as the AES-XCBC-MAC-PRF with output blocksize 128 bits as defined by IPsec WG in IETF. Please note:
   - Refer to [RFC 3566] for the AES-XCBC-MAC-PRF based key generation function.
   - Refer to [RFC 3664] 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:

   $$T1 = AES\_XCBC\_MAC\_PRF\{AS\}(CONSTANT \,||\, 0x01)$$

   $$T2 = AES\_XCBC\_MAC\_PRF\{AS\}(T1 \,||\, CONSTANT \,||\, 0x02)$$

   ....

   $$Tn = AES\_XCBC\_MAC\_PRF\{AS\}(T1 \,||\, CONSTANT \,||\, n)$$

   where *AS* is the appropriate authentication seed (be it TAS, PAS, SAS or RIAK) and *CONSTANT* is the appropriate constant as described in preceding sections. 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:

   $$AK = MSB_{160}(T1 \,||\, T2)$$

   The generated authentication key SHALL be applied as described in preceding sections.

C.15 Authentication of the tokens_consumed Field in the Token Consumption Data

Devices SHALL authenticate the tokens_consumed field and the message_seq_number of the token consumption report, see Section 7.4, in the way specified in this section. The hash function used here is one of the four secure hash functions from [SCHNEIER], page 449 (the upper left one in figure 18.9).

The maximum amount of tokens that can be reported as consumed is 9999. The amount of tokens, with the before mentioned restriction, is represented with a 14 bit uimsbf number and called tokens_consumed. The value of message_seq_number 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 message_seq_number. The resulting 18 bit number is right padded with 0x1 and right padded again with 109 binary zeroes (so 2109). 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 Section 7.6.4, 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.

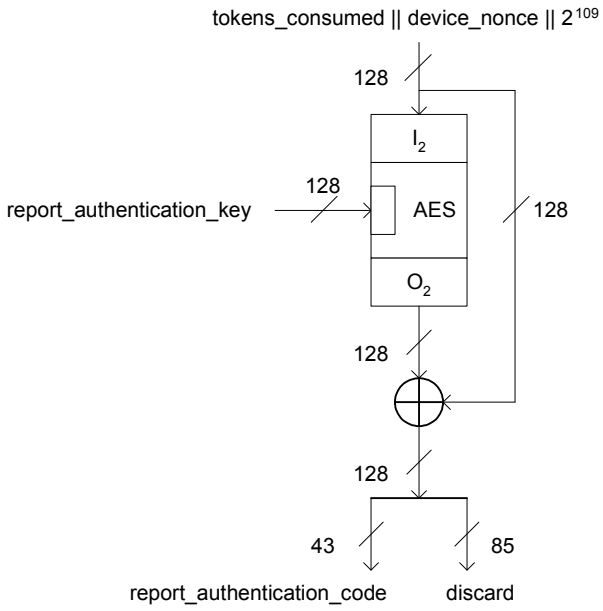


Figure 40: Computation of the report_authentication_code

C.16 Management of Tokens by RIs and Devices

C.16.1 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/she actually consumes. These tokens are post-paid tokens.

The tools to support these two business models are the token delivery response message, see Section 7.6.2 and the token reporting protocol, see Section 7.6.3. The next sections 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.

C.16.1.1 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.

C.16.1.2 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 tokens6.

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 Section C.16.1.3, 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, message_seq_number, 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".

C.16.1.3 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 section. The device will report the actual consumption of tokens that

---

6 Note that 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.

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 C.16.2. 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 Section C.16.2.

C.16.1.4 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 Section C.16.1.5 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.

C.16.1.5 Stopping the Post-Paid Token Business Model

When at a certain point of time, the user informs the RI that he/she does no longer wish to use post-paid tokens, not even the ones that are still in his/her 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.

C.16.2 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 message_seq_number which is one higher than the previously used message_seq_number. If the previously used message_seq_number was 9, the device uses 0 as the next message_seq_number.
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 stop 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.

C.17 Confidentiality in the Subscriber Group Concept

In this specification, there are 2 ways specified to deliver BCROs (see Section 10.3):

- Using fixed subscriber groups
- Using flexible subscriber groups

In order to deliver the BCROs efficiently and securely, 2 broadcast encryption schemes are used:

- Zero message broadcast encryption by fixed and flexible subscriber groups, see Section C.17.2.
- OFT by flexible subscriber groups only, see Section C.17.3.

Both broadcast encryption schemes use the same node numbering, see Section C.17.1.

The above mentioned broadcast encryption schemes have the following differences:

- the device interpretation of the delivered keys during the registration process;
- the derivation of the node keys in the subscriber group key derivation tree;
- the derivation of the Deduced Encryption Key and the complexity of the needed calculations (see Section 10.3.4.4);
- the number of BCROs instances that are needed to address a subset of a subscriber group.

Some of these differences are described in Section C.17.2 and C.17.3.

C.17.1 Node numbering

The nodes from the subscriber group key derivation tree are sequentially numbered per “level” from left to right starting from the root node (i.e. in a breadth-first manner). The root node has number 0. For a node with number i, the first child has a number 2i+1, and the second child a number 2i+2. Likewise, the parent of a child with number j has the number (j-1)>>1 (see also Figure 40).

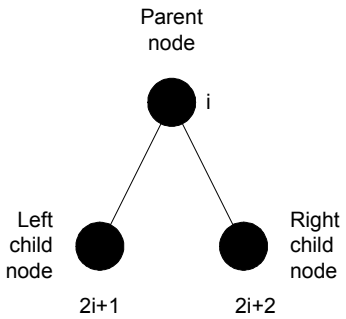


Figure 41: Subscriber group node numbering

The node number of a leaf can be calculated from the leaf number by adding 2k-1 to it, where k is the height of the tree.

For example, in Figure 42 the height of the tree is 3, therefore leaf D4 has number 4+23-1 = 11.

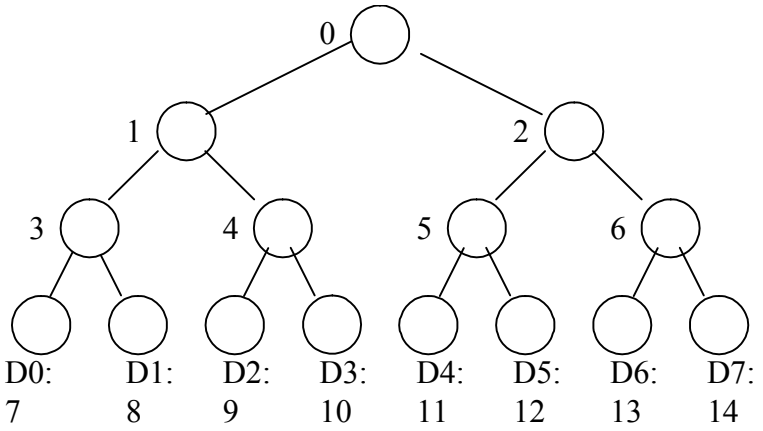


Figure 42: Example of a subscriber group key derivation tree of height 3.

C.17.2 BCRO delivery using zero message broadcast encryption scheme

This section provides some background information on zero-message broadcasting and key delivery.

C.17.2.1 Exponential Scheme

As there are 2n subsets of a group of n devices, a very inefficient way of implementing the scheme is to generate 2n distinct keys. Each device would be provided with the keys associated with all the subsets that include that device.

Group size	Number of subsets	Number of keys per device
1	2	1
2	4	2
4	16	8
8	256	128
16	65536	32768
32	4294967296	2147483648

This is for all practical purposes completely unusable.

C.17.2.2 Linear Scheme

An easy optimisation of the grossly impractical scheme is to generate an exclusion key unique per device part of the group. Each device is given all exclusion keys, except its own exclusion key. For any subset of the group that is to be allowed to access content, one can define the complement subset. If all the exclusion keys of the devices in the complement subset are used in a key derivation function, then only those devices in the complement subset cannot compute all the key material required: they lack the key associated with themselves.

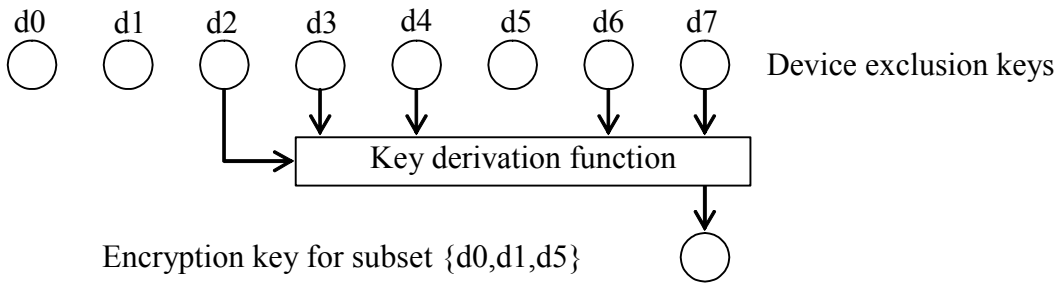


Figure 43: Derivation of an encryption key associated with a subset of the group

The figure shows the derivation of an encryption key for the addressed subset {d0, d1, d5}. The derivation function used is HMAC-SHA1-128 over the concatenation of the exclusion keys of the non-addressed devices. Each of the devices from the complement subset {d2, d3, d4, d6, d7} will find that its key is used in this derivation. Consequently neither of the devices from the complement subset can compute the encryption key. For example, device d4 cannot compute the required Deduced Encryption Key:

DEK = HMAC-SHA1-128{ DK2 || DK3 || DK4 || DK6 || DK7 } (BCI)

because it only knows DK0, DK1, DK2, DK4, DK5, DK6 and DK7.

Note that BCI is the Binary Content Identifier included in the BCRO and DKi is the exclusion key corresponding to device di.

The size of the key material to be distributed now scales linear with the size of the group. This is a big improvement over the exponential scaling of the naïve approach.

Group size	Number of subsets	Number of keys per device
1	2	0
2	4	1
4	16	3
8	256	7
16	65536	15
32	4294967296	31
64	1.84×10^{19}	63
128	3.40×10^{38}	127
256	1.16×10^{77}	255
512	1.34×10^{154}	511
1024	1.80×10^{308}	1023

This is a great improvement, and can make the scheme already practical for modest group sizes.

C.17.2.3 Logarithmic Scheme

In [FIAT_NAOR] the authors provide a scheme of hierarchical key derivations. Under this scheme, each device is provided key material that allows on-demand computing of the keys associated with all other devices in the group, except itself. The following picture shows schematically how this operates:

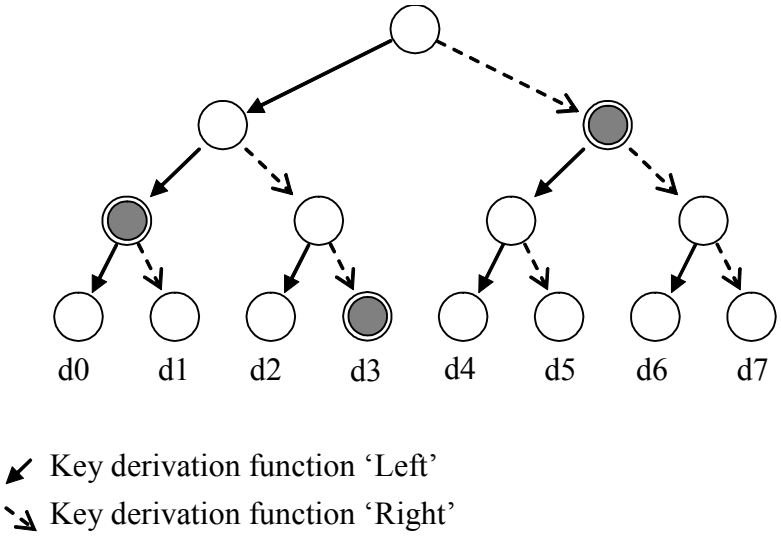


Figure 44: Fiat-Naor key derivation scheme

The figure shows the application of two similar, but different, key derivation functions. From a single key, two child keys can be derived using these two distinct functions. A tree hierarchy of keys can thus be formed. The complete tree is determined completely by the two key derivation functions and the single root key.

This scheme allows an efficient version of the linear scheme. Instead of distributing all keys (except its own) to a device, now only a few keys from the tree need to be distributed to each device. It can be shown that instead of n-1 keys, now it is sufficient to distribute log2n keys to each device.

Group size (*N* DEVICES)	**Total number of keys in the group**		**Number of keys per device**	
	Linear scheme n (n-1)	**Logarithmic scheme $n \log_2 n$**	**Linear scheme (n-1)**	**Logarithmic scheme $\log_2 n$**
1	0	0	0	0
2	2	2	1	1
4	12	8	3	2
8	56	24	7	3
16	240	64	15	4
32	992	160	31	5
64	4032	384	63	6
128	16256	896	127	7
256	65280	2048	255	8
512	261632	4608	511	9
1024	1047552	10240	1023	10
…				
1048576	1.10×10^{12}	20971520	1048575	20

A practical limit to the subscriber group size is given by the need to communicate which subset of the group is selected to access particular content. This is typically done with a bitvector, indicating which devices are included in the subset. For each communication to a specific subset, such a bitvector of n bits length must be added in order for the devices to determine the used encryption key.

It must be noted that if the subset of devices allowed to access content is the whole group, then the derivation of the content encryption key fails, because there is no device key at all to include in the key derivation algorithm. To address this issue, devices are provided with one additional special key, to be used when the whole group is addressed.

C.17.3 BCRO delivery using OFT

The OFT scheme is an optional broadcast encryption scheme that MAY only be used in Flexible Subscriber Groups. The OFT support is optional for both server and client. The size of a Flexible Subscriber Group can vary from 2^1 to 2^{31}, but is fixed after registration.

The OFT scheme is based on a *One-Way Function tree* (OFT) [OFT]. This key derivation tree is similar to the zero-message broadcast encryption key derivation tree. The node numbering in the OFT scheme is equal to the node numbering in the zero-message broadcast encryption scheme, see Section C.17.1.

A node in the OFT contains two entries: one 128-bit *node key* and one 128-bit *blinded node key*. Each node key can be used to create a Deduced Encryption Key:

DEK := HMAC-SHA1-128{ *node key* }(BCI)

The node keys and blinded node keys in thBe OFT are computed with the aid of a *one way function* and a *mixing function*. In this specification, these are defined as:

$\text{one\_way\_function}_i(a) := \text{AES-128-ENCRYPT}\{a\}((i+\text{LEFT\_CONSTANT}) \bmod 2^{128})$

mixing_function(a, b) := a XOR b

where a and b are the parameters to the functions, i is the node number of the node where the one way function is used and LEFT_CONSTANT is defined in Section 10.3.4.4. Notice that the one way function is chosen equal to one of the one way functions used in the zero-message broadcast encryption scheme.

The keys in the OFT are constructed recursively from the leaves to the root:

- The node keys of the leaves are generated randomly.
- When the node i has a node key NK_i , it has a blinded node key $BNK_i = \text{one\_way\_function}_i(NK_i)$.
- The parent's node key NK_p of two children nodes that have respectively blinded node keys BNK_i and BNK_j is derived as $NK_p = \text{mixing\_function}(BNK_i, BNK_j)$.

Figure 45 illustrates the keys of a parent node and its two children nodes.

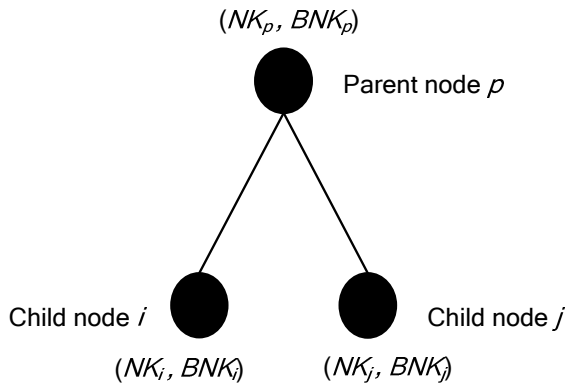


Figure 45: Keys in the OFT

To address a subset of the Flexible Subscriber Group by using the OFT encryption scheme, each device is assigned to a leaf of the OFT. The devices get the node key of their assigned leafs. In addition, each device is given all blinded node keys of the sibling nodes on the path from its leaf to the root node. With this information, a device can calculate the node keys of all nodes on this path, see Figure 46.

Notice that a device can calculate the appropriate node keys on the path from its leaf to the root node, but none of the other node keys.

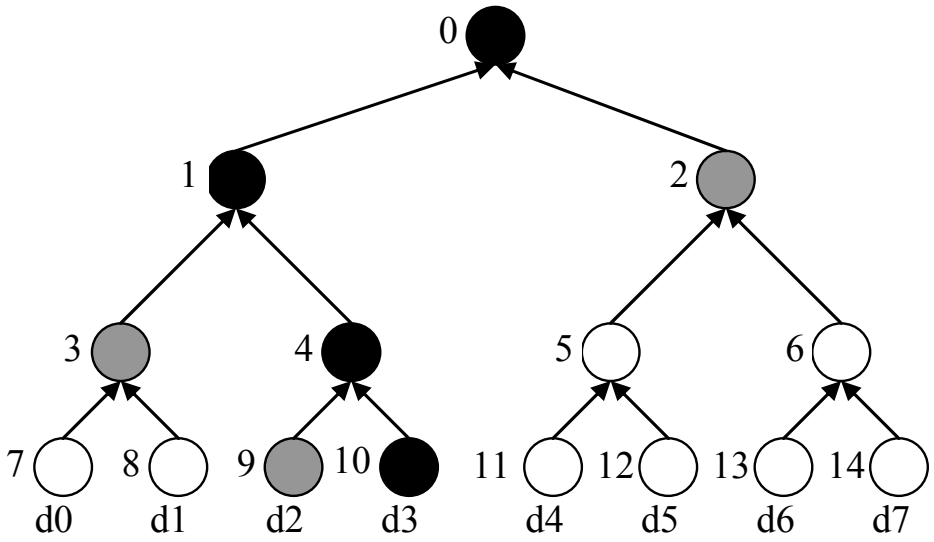


Figure 46: OFT for 8 devices with known keys of d3 marked.Black color means that d3 knows the node key, grey color that it knows the blinded key of the node.

To address a Flexible Subscriber Group subset, the Right Issuer creates a set of BCROs. The number of BCROs and which node keys are used for the associated DEKs depends on the subset: The RI constructs a minimal set of subtrees of which all leaves are addressed [NAOR02]. With each of these subtrees one BCRO is associated, whose DEK is derived from the node key in the root of the subtree. Notice that all devices in a subtree are able to compute the associated DEK and therewith can decrypt the PEK/SEK/CEK in the BCRO. In contrast, a non-subscribed device will not be able to calculate any of the DEKs.

Example:

In the Flexible Subscriber Group from Figure 46 the RI wants to address {d0, d1, d2, d3, d4}. Therefore two BCROs are needed. In the first one the PEK/SEK/CEK is encrypted with the DEK derived from the node key of node 1. Notice that {d0, d1, d2, d3} can calculate this key and decrypt the PEK/SEK/CEK since they know the node key of node 1. The second BCRO uses the DEK derived from the node key of node 11 as encryption key. Only {d4} can compute this DEK. Notice that although the non-subscribed devices might know the blinded key of node 1 or 11, they cannot calculate the appropriate node key, since it is infeasible to calculate the inverse of the one way function.

In most cases, more than one BCRO is needed to address the subscribed devices. The worst case occurs, when n/2 members are subscribed, one from each of the subtrees just above the leaf-level. The RI can try to minimize the number of BCROs by assigning the subscribers strategically to the nodes in the OFT. The minimum is achieved, if as many as possible subscribed devices share a common ancestor.

Note that neither the blinded key of the root nor the blinded keys of the two children of the root need to be transmitted. The blinded keys of the two children would only be needed to calculate the key of the root, which would only be needed for the addressing of the whole group. However, when addressing the whole group the UGK is used rather than the root key from this tree.

C.18 PDCF box structure example (Informative)

This informative section presents a non-exhaustive example of the PDCF box structure, including the boxes defined by ISO in [ISO14496-12] and the OMA boxes defined in [DRMCF-v2] and in this specification.

In both Tables below additional boxes may be necessary. The nesting order of the boxes is as follows: on the left is the parent and on the right, the child.

Table 56 shows an example of the 'ftyp' and 'moov' part of the PDCF box structure when a protected audio or video track is defined. Note, that the OMA information is specified per track. The file format structure corresponds to OMA DRM v2.0 [DRMCF-v2] and is not modified in this specification. It is fully ISO compliant.

Table 56: Partial box structure of a PDCF file with a single protected track

Data type/value											Field purpose
'ftyp'											ISO File header (fixed File Type box)
'moov'											ISO movie box
	'mvhd'										ISO movie header box
	'trak'										ISO track box
		'tkhd'									ISO track header
		'tref'									ISO track reference
		'mdia'									ISO media information box
			'mdhd'								ISO media header
			'hdlr'								ISO handler
			'minf'								ISO media information container
				'stbl'							ISO sample table box, container for the time/space map
					'stsd'						ISO sample descriptions 'soun' for audio tracks 'vide' for video tracks
						'encv or 'enca'					ISO protected sample entry
							'sinf'				ISO protection scheme information box (always present)
								'frma'			ISO original format (always present)
								'schm'			ISO SchemeTypeBox (when used to apply to single track)
								'schi'			ISO SchemeInformationBox

											(if applies to this ‘trak’ only)
									'odkm'		OMA DRM KMS box
										‘ohdr’	OMA DRM Common Headers box (when used to apply to single track)
										'odaf'	OMA DRM AU Format Box (when used to apply to single track)

Table 57 below shows an example of the OMA STKM track structure inside a PDCF file.

Table 57: Part of the box structure of a PDCF file showing OMA STKM track

Data type/value										Field purpose
	‘trak’									ISO track box
		‘tkhd’								ISO track header
		‘tref’								ISO track reference
		‘mdia’								ISO media information box
			‘mdhd’							ISO media header
			‘hdlr’							ISO handler
			‘minf’							ISO media information container
				‘stbl’						ISO sample table box, container for the time/space map
					‘stsd’					ISO sample descriptions 'meta' for OMA STKM track (Metadata track)
						‘oksd’				OMA key sample description box

C.19 MIME media types

C.19.1 Media-Type Registration Request for application/vnd.oma.drm.risd+xml

This section provides the registration request, as per [RFC 2048], to be submitted to IANA.

Type name: application

Subtype name: vnd.oma.drm.risd+xml

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations:

Rights Issuer services carry Rights Objects, Registration data and Certificate Chain Updates in messages over broadcast channels. BCAST Rights Issuer Service Data shall be passive, and do not generally represent a unique or new security threat. Some messages contain confidential and security critical fields. These are encrypted using AES-128-CBC, and are authenticated using HMAC-SHA-1-96 as described in the OMA BCAST specification referenced below. That means, the security of such critical fields is provided by the type itself and does not have to be provided externally. Furthermore, in sharing any kind of data, there is some risk that unintentional information may be exposed, and that risk applies to unencrypted fields contained in Rights Issuer Service Data as well.

Interoperability considerations:

This content type carries BCAST Rights Issuer Service Data within the scope of the OMA BCAST enabler. The OMA BCAST enabler specification includes static conformance requirements and interoperability test cases for this content.

Published specification:

OMA BCAST 1.0 Enabler Specification – OMA DRM v2.0 Extensions for Broadcast Support, especially Section 12.7.1. Available from http://www.openmobilealliance.org

Applications, which use this media type:

OMA BCAST Services

Additional information:

Magic number(s): none

File extension(s): none

Macintosh File Type Code(s): none

Person & email address to contact for further information:

Uwe Rauschenbach

Uwe.Rauschenbach@nsn.com

Intended usage: Limited use.

Only for usage with OMA DRM v2.0 Extensions for Mobile Broadcast Services, which meet the semantics given in the mentioned specification.

Author/Change controller: OMNA – Open Mobile Naming Authority, OMA-OMNA@mail.openmobilealliance.org